OS/390

UNIX System Services Command Reference

SC28-1892-09

OS/390

UNIX System Services Command Reference

SC28-1892-09

Note

Before using this information and the product it supports, be sure to read the general information under Appendix L, “Notices” on page 845.

Tenth Edition (March 2000)

This is a major revision of SC28-1892-08.

This edition applies to Version 2 Release 9 of OS/390 (5647-A01) and to all subsequent releases and modifications until otherwise indicated in new editions.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address below.

IBM welcomes your comments. A form for readers' comments may be provided at the back of this publication, or you may address your comments to the following address:

International Business Machines Corporation
Department 55JA, Mail Station P384
2455 South Road
Poughkeepsie, NY 12601-5400
United States of America

FAX (United States & Canada): 1+914+432-9405
FAX (Other Countries):
Your International Access Code +1+914+432-9405

IBMLink (United States Customers Only): IBMUSM10(MHVRCFS)
IBM Mail Exchange: USIB6TC9 at IBMMAIL
Internet e-mail: mhvrcfs@us.ibm.com
World Wide Web: http://www.ibm.com/s390/os390/

If you would like a reply, be sure to include your name, address, telephone number, or FAX number.

Make sure to include the following in your comment or note:

- Title and order number of this book
- Page number or topic related to your comment

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Information in this document has been adapted from the *InterOpen™/POSIX Shell and Utilities User Manual,* supplied by Mortice Kern Systems (MKS) Inc. for use by licensees of their InterOpen/POSIX Shell and Utilities source code product. © Copyright 1985, 1993 Mortice Kern Systems, Inc. © Copyright 1989 Software Development Group, University of Waterloo.

The information contained in the glossary section and tagged by the word [POSIX] is copyrighted information of the Institute of Electrical and Electronics Engineers, Inc., extracted from IEEE Std 1003.1-1990, IEEE P1003.0, and IEEE P1003.2. This information was written within the context of these documents in their entirety. The IEEE takes no responsibility or liability for and will assume no liability for any damages resulting from the reader's misinterpretation of said information resulting from the placement and context in this publication. Information is reproduced with the permission of the IEEE.

© Copyright International Business Machines Corporation 1996, 2000. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Permuted Index xv

About This Book xxxi
Who Should Use This Book xxxi
Finding More Information about Other Products xxxi
Where to Find More Information xxxi

Summary of Changes xxxv

Part 1. Introduction to Shell Commands and DBCS 1
Reading the Command Descriptions 1
Using the Doublebyte Character Set (DBCS) 7

Part 2. Shell Command Descriptions 11
alias — Display or create a command alias 11
ar — Create or maintain library archives 14
asa — Interpret ASA/FORTRAN carriage control 17
at — Run a command at a specified time 18
autoload — Indicate function name not defined 22
automount — Configure the automount facility 23
awk — Process programs written in the awk language 25
basename — Return the nondirectory components of a pathname 42
batch — Run commands when the system is not busy 43
bc — Use the arbitrary-precision arithmetic calculation language 44
bg — Move a job to the background 61
bpxmtext — Display reason code text 62
break — Exit from a loop in a shell script 63
c++ — Compile C++/MVS and C/MVS and create an executable file 64
c89 — Compile ANSI C/MVS source code and create an executable file 64
cal — Display a calendar for a month or year 94
calendar — Display all current appointments 95
cancel — Cancel print queue requests (stub command) 96
captoinfo — Print the terminal entries in the terminfo database 97
cat — Concatenate or display text files 98
cc — Compile common usage source code and create an executable file . . . 100
cd — Change the working directory 100
chaudit — Change audit flags for a file 103
chcp — Set or query ASCII/EBCDIC code pages for the terminal 105
chgrp — Change the group owner of a file or directory 108
chmod — Change the mode of a file or directory 109
chmount — Change the mount attributes of a file system 112
chown — Change the owner or group of a file or directory 113
chroot — Change the root directory for the execution of a command 114
cksum — Calculate and write checksums and byte counts 116
cmp — Compare two files 118
col — Remove reverse line feeds 119
: (colon) — Do nothing, successfully 120
comm — Show and select or reject lines common to two files 121
command — Run a simple command 123
compress — Lempel-Ziv file compression 124

© Copyright IBM Corp. 1996, 2000

confighfs — Invoke vfs_pfsctl HFS functions . . . 126
configstk — Configure the AF_UEINT stack . . . 127
continue — Skip to the next iteration of a loop in a shell script . . . 129
cp — Copy a file . . . 130
cpio — Copy in/out file archives . . . 140
cron daemon — Run commands at specified dates and times . . . 143
crontab — Schedule regular background jobs . . . 145
csplit — Split text files . . . 147
ctags — Create tag files for ex, more, and vi . . . 150
cu — Call up another system (stub only) . . . 152
cut — Cut out selected fields from each line of a file . . . 152
date — Display the date and time . . . 154
dbx — Use the debugger . . . 157
? subcommand for dbx: Search backward for a pattern . . . 160
/ subcommand for dbx: Search forward for a pattern . . . 161
alias subcommand for dbx: Display and assign subcommand aliases . . . 161
args subcommand for dbx: Display program arguments . . . 162
assign subcommand for dbx: Assign a value to a variable . . . 162
case subcommand for dbx: Change how dbx interprets symbols . . . 163
catch subcommand for dbx: Start trapping a signal . . . 163
clear subcommand for dbx: Remove all stops at a given source line . . . 164
cleari subcommand for dbx: Remove all breakpoints at an address . . . 165
condition subcommand for dbx: Display a list of active condition variables . . . 165
cont subcommand for dbx: Continue program execution . . . 166
delete subcommand for dbx: Remove traces and stops . . . 167
detach subcommand for dbx: Continue program execution without dbx control . . . 167
display memory subcommand for dbx: Display the contents of memory . . . 168
down subcommand for dbx: Move the current function down the stack . . . 170
dump subcommand for dbx: Display the names and values of variables in a procedure . . . 170
edit subcommand for dbx: Invoke an editor . . . 171
file subcommand for dbx: Change the current source file . . . 172
func subcommand for dbx: Change the current function . . . 172
goto subcommand for dbx: Run a specified source line . . . 173
gotoi subcommand for dbx: Change the program counter address . . . 173
help subcommand for dbx: Display a subcommand synopsis . . . 174
history subcommand for dbx: Display commands in a history list . . . 174
ignore subcommand for dbx: Stop trapping a signal . . . 174
list subcommand for dbx: Display lines of the current source file . . . 175
listfiles subcommand for dbx: Display the list of source files . . . 176
listfuncs subcommand for dbx: Display the list of functions . . . 177
listi subcommand for dbx: List instructions from the program . . . 177
map subcommand for dbx: Display load characteristics . . . 178
move subcommand for dbx: Change the next line to be displayed . . . 179
multproc subcommand for dbx: Enable or disable multiprocess debugging . . . 179
mutex subcommand for dbx: Display a list of active mutex objects . . . 180
next subcommand for dbx: Run the program up to the next source line . . . 181
nexti subcommand for dbx: Run the program up to the next machine instruction . . . 182
object subcommand for dbx: Load an object file . . . 182
onload subcommand for dbx: Evaluate stop/trace after dll load . . . 183
print subcommand for dbx: Print the value of an expression . . . 184
prompt subcommand for dbx: Change the dbx command prompt . . . 184

quit subcommand for dbx: End the dbx debugging session 185
readwritelock subcommand for dbx: Display a list of active read/write lock objects 185
record subcommand for dbx: Append user's commands to a file 186
registers subcommand for dbx: Display the value of registers 187
rerun subcommand for dbx: Begin running a program with the previous arguments 188
return subcommand for dbx: Continue running a program until a return is reached 188
run subcommand for dbx: Run a program 189
set subcommand for dbx: Define a value for a dbx variable 190
sh subcommand for dbx: Pass a command to the shell for execution 194
skip subcommand for dbx: Continue from the current stopping point 194
source subcommand for dbx: Read subcommands from a file 195
status subcommand for dbx: Display the active trace and stop subcommands 195
step subcommand for dbx: Run one or more source lines 196
stepi subcommand for dbx: Run one or more machine instructions 196
stop subcommand for dbx: Stop execution of a program 197
stopi subcommand for dbx: Stop at a specified location 198
thread subcommand for dbx: Display a list of active threads 199
trace subcommand for dbx: Print tracing information 200
tracei subcommand for dbx: Turn on tracing 201
unalias subcommand for dbx: Remove an alias 202
unset subcommand for dbx: Delete a variable 203
up subcommand for dbx: Move the current function up the stack 203
use subcommand for dbx: Set the list of directories to be searched 204
whatis subcommand for dbx: Display the type of program components 204
where subcommand for dbx: List active procedures and functions 205
whereis subcommand for dbx: Display the full qualifications of symbols 206
which subcommand for dbx: Display the full qualification of an identifier . . . 206
dd — Convert and copy a file 207
df — Display the amount of free space in the file system 212
diff — Compare two text files and show the differences 214
dircmp — Compare directories 219
dirname — Return the directory components of a pathname 220
. (dot) — Run a shell file in the current environment 221
dspcat — Display all or part of a message catalog 222
dspmsg — Display selected messages from message catalogs 223
du — Summarize usage of file space 224
echo — Write arguments to standard output 225
ed — Use the ed line-oriented text editor 227
egrep — Search a file for a specified pattern 235
env — Display or set environment variables for a process 235
eval — Construct a command by concatenating arguments 236
ex — Use the ex text editor 237
exec — Run a command and open, close, or copy the file descriptors 238
exit — Return to the shell's parent process or to TSO/E 240
expand — Expand tabs to spaces 241
export — Set a variable for export 242
expr — Evaluate arguments as an expression 243
exrecover daemon — Retrieve vi and ex files 245
extattr — Set, reset, and display extended attributes for files 248
false — Return a nonzero exit code 249
fc — Process a command history list 250

fg — Bring a job into the foreground . . . 253
fgrep — Search a file for a specified pattern . . . 254
file — Determine file type . . . 254
filecache — Manage file caches . . . 256
find — Find a file meeting specified criteria . . . 256
fold — Break lines into shorter lines . . . 261
functions — Display or assign attributes to functions . . . 262
fuser — List process IDs of processes with open files . . . 262
gencat — Create or modify message catalogs . . . 264
getconf — Get configuration values . . . 267
getopts — Parse utility options . . . 271
grep — Search a file for a specified pattern . . . 274
hash — Create a tracked alias . . . 276
head — Display the first part of a file . . . 277
history — Display a command history list . . . 279
iconv — Convert characters from one code set to another . . . 280
id — Return the user identity . . . 281
inetd daemon — Provide Internet Service Management . . . 283
infocmp — Compare or print the terminal description . . . 286
integer — Mark each variable with an integer value . . . 290
ipcrm — Remove message queues, semaphore sets, or shared memory IDs . 291
ipcs — Report status of the interprocess communication facility . . . 292
jobs — Return the status of jobs in the current session . . . 297
join — Join two sorted textual relational databases . . . 299
kill — End a process or job, or send it a signal . . . 301
[(left bracket) — Test for a condition . . . 304
let — Evaluate an arithmetic expression . . . 304
lex — Generate a program for lexical tasks . . . 306
line — Copy one line of standard input . . . 309
link — Create a hard link to a file . . . 310
lm — Starting login monitor for OCS support . . . 311
ln — Create a link to a file . . . 312
locale — Get locale-specific information . . . 316
localedef — Define the locale environment . . . 320
logger — Log messages . . . 322
logname — Return a user's login name . . . 324
lp — Send a file to a printer . . . 325
lpstat — Show status of print queues (stub command) . . . 326
ls — List file and directory names and attributes . . . 327
mail — Read and send mail messages . . . 332
mailx — Send or receive electronic mail . . . 335
make — Maintain program-generated and interdependent files . . . 355
man — Display sections of the online reference manual . . . 377
mesg — Allow or refuse messages . . . 380
mkcatdefs — Preprocess a message source file . . . 381
mkdir — Make a directory . . . 383
mkfifo — Make a FIFO special file . . . 385
mknod — Make a FIFO or character special file . . . 386
more — Display files on a page-by-page basis . . . 387
mount — Logically mount a file system . . . 391
mv — Rename or move a file or directory . . . 393
newgrp — Change to a new group . . . 402
nice — Run a command at a different priority . . . 404
nl — Number lines in a file . . . 405

nm — Display symbol table of object, library, or executable files 407
nohup — Start a process that is immune to hangups 410
obrowse — Browse an HFS file 411
ocsconfig — Configure, unconfigure, or query an OCS object 412
od — Dump a file in a specified format 419
oedit — Edit an HFS file 423
pack — Compress files by Huffman coding 424
passwd — Change user passwords 426
paste — Merge corresponding or subsequent lines of a file 427
patch — Change a file using diff output 429
pathchk — Check a pathname 432
pax — Interchange portable archives 433
pcat — Unpack and display Huffman packed files 445
pg — Display files interactively 446
pr — Format a file in paginated form and send it to standard output 450
print — Return arguments from the shell 453
printenv — Display the values of environment variables 455
printf — Write formatted output 456
ps — Return the status of a process 458
pwd — Return the working directory name 465
r — Process a command history list 465
read — Read a line from standard input 466
readonly — Mark a variable as read-only 468
renice — Change priorities of a running process 469
return — Return from a shell function or . (dot) script 470
rlogind — Validate rlogin requests 471
rm — Remove a directory entry 473
rmdir — Remove a directory 474
runcat — Pipe output from mkcatdefs to gencat 475
sed — Start the sed noninteractive stream editor 476
set — Set or unset command options and positional parameters 481
sh — Invoke a shell 485
shedit — Interactive command and history editing in the shell 513
shift — Shift positional parameters 519
sleep — Suspend execution of a process for an interval of time 520
sort — Start the sort-merge utility 521
spell — Detect spelling errors in files 527
split — Split a file into manageable pieces 530
stop — Suspend a process or job 531
strings — Display printable strings in binary files 532
strip — Remove unnecessary information from an executable file 533
stty — Set or display terminal options 534
su — Change the user ID associated with a session 540
sum — Compute checksum and block count for file 543
suspend — Send a SIGSTOP to the current shell 544
sysvar — Display static system symbols 545
tabs — Set tab stops 545
tail — Display the last part of a file 548
talk — Talk to another user 549
tar — Manipulate the tar archive files to copy or back up a file 551
tcsh — Invoke a C shell 556
% (percent) built-in command for tcsh: Move jobs to the foreground or background 600

alloc built-in command for tcsh: Show the amount of dynamic memory acquired . . . 601
bindkey built-in command for tcsh: List all bound keys . . . 601
builtins built-in command for tcsh: Prints the names of all built-in commands 603
bye built-in command for tcsh: Terminate the login shell . . . 603
chdir built-in shell command for tcsh: Change the working directory . . . 603
complete built-in command for tcsh: List completions . . . 603
dirs built-in command for tcsh: Print the directory stack . . . 607
echotc built-in command for tcsh: Exercise the terminal capabilities in args . 609
filetest built-in command for tcsh: Apply the op file inquiry operator to a file . . 609
glob built-in command for tcsh: Write each word TO STANDARD OUTPUT . 610
hashstat built-in command for tcsh: Print a statistic line on hash table effectiveness . . . 610
hup built-in command for tcsh: Run command so it exits on a hang-up signal 611
limit built-in command for tcsh: Limit consumption of processes . . . 611
log built-in command for tcsh: Print the watch tcsh shell variable . . . 612
login built-in command for tcsh: Terminate a login shell . . . 612
logout built-in command for tcsh: Terminate a login shell . . . 613
ls-F built-in command for tcsh: List files . . . 613
notify built-in command for tcsh: Notify user of job status changes . . . 614
onintr built-in command for tcsh: Control the action of the tcsh shell on interrupts . . . 615
popd built-in command for tcsh: Pop the directory stack . . . 615
pushd built-in command for tcsh: Make exchanges within directory stack . . . 616
rehash built-in command for tcsh: Recompute internal hash table . . . 617
repeat built-in command for tcsh: Execute command count times . . . 617
sched built-in command for tcsh: Print scheduled event list . . . 617
setenv built-in command for tcsh: Set environment variable name to value . . 618
settc built-in command for tcsh: Tell tcsh shell the terminal capability cap value . . . 619
setty built-in command for tcsh: Control tty mode changes . . . 619
source built-in command for tcsh: Read and execute commands from name . 620
telltc built-in command for tcsh: List terminal capability values . . . 620
uncomplete built-in command for tcsh: Remove completions whose names match pattern . . . 620
unhash built-in command for tcsh: Disable use of internal hash table . . . 621
unlimit built-in command for tcsh: Remove resource limitations . . . 621
unsetenv built-in command for tcsh: Remove environmental variables that match pattern . . . 621
watchlog built-in command for tcsh: Print the watch shell variable . . . 622
where built-in command for tcsh: Report all instances of command . . . 622
which built-in command for tcsh: Display next executed command . . . 622
tee — Duplicate the output stream . . . 623
test — Test for a condition . . . 624
tic — Put terminal entries in the terminfo database . . . 627
time — Display processor and elapsed times for a command . . . 628
times — Get process and child process times . . . 630
touch — Change the file access and modification times . . . 631
tput — Change characteristics of terminals . . . 633
tr — Translate characters . . . 635
trap — Intercept abnormal conditions and interrupts . . . 637
true — Return a value of 0 . . . 639
tso — Run a TSO/E command from the shell . . . 639
tsort — Sort files topologically . . . 644

tty — Return the user's terminal name . . . 645
type — Tell how the shell interprets a name . . . 646
typeset — Assign attributes and values to variables . . . 646
uconvdef — Create binary conversion tables . . . 649
ulimit — Set process limits . . . 650
umask — Set or return the file mode creation mask . . . 651
unalias — Remove alias definitions . . . 652
uname — Display the name of the current operating system . . . 653
uncompress — Undo Lempel-Ziv compression of a file . . . 655
unexpand — Compress spaces into tabs . . . 656
uniq — Report or filter out repeated lines in a file . . . 657
unlink — Removes a directory entry . . . 659
unmount — Remove a file system from the file hierarchy . . . 660
unpack — Decode Huffman packed files . . . 661
unset — Unset values and attributes of variables and functions . . . 663
uucc — Compile UUCP configuration files . . . 664
uucico daemon — Process UUCP file transfer requests . . . 665
uucp — Copy files between remote UUCP systems . . . 667
uucpd daemon — Invoke uucico for TCP/IP connections from remote UUCP systems . . . 671
uudecode — Decode a transmitted binary file . . . 672
uuencode — Encode a file for safe transmission . . . 673
uulog — Display log information about UUCP events . . . 674
uuname — Display list of remote UUCP systems . . . 676
uupick — Manage files sent by uuto and uucp . . . 677
uustat — Display status of pending UUCP transfers . . . 678
uuto — Copy files to users on remote UUCP systems . . . 682
uux — Request command execution on remote UUCP systems . . . 683
uuxqt daemon — Carry out command requests from remote UUCP systems . . . 686
vi — Use the display-oriented interactive text edit or . . . 688
wait — Wait for a child process to end . . . 719
wall — Broadcast a message to logged-in users . . . 721
wc — Count newlines, words, and bytes . . . 721
whence — Tell how the shell interprets a command name . . . 722
who — Display information about current users . . . 723
whoami — Display your effective user name . . . 725
write — Write to another user . . . 725
xargs — Construct an argument list and run a command . . . 727
yacc — Use the yacc compiler . . . 731
zcat — Uncompress and display data . . . 735

Part 3. TSO/E Commands . . . 737
BPXBATCH — Run shell commands, shell scripts, or executable files . . . 738
ISHELL — Invoke the ISPF shell . . . 739
MKDIR — Make a directory . . . 740
MKNOD — Create a character special file . . . 741
MOUNT — Logically mount a file system . . . 743
OBROWSE — Browse an HFS file . . . 747
OCOPY — Copy an MVS data set member or HFS file to another member or file . . . 748
OEDIT — Edit an HFS file . . . 753
OGET — Copy an HFS file into an MVS data set . . . 754
OGETX — Copy HFS files from a directory to an MVS PDS or PDSE . . . 758
OHELP — Display online OS/390 UNIX System Services publications . . . 762

OMVS — Invoke the OS/390 shell . . . 763
OPUT — Copy an MVS data set member into an HFS file . . . 776
OPUTX — Copy members from an MVS PDS or PDSE to an HFS directory . . . 779
OSHELL — Invokes BPXBATCH from TSO/E . . . 783
OSTEPLIB — Build a list of files . . . 784
UNMOUNT — Remove a file system from the file hierarchy . . . 785

Appendix A. OS/390 Shell Command Summary . . . 789
General Use . . . 789
Controlling Your Environment . . . 789
Daemons . . . 790
Managing Directories . . . 790
Managing Files . . . 791
Printing Files . . . 792
Computing and Managing Logic . . . 792
Controlling Processes . . . 793
Writing Shell Scripts . . . 793
Developing or Porting Application Programs . . . 793
Communicating with the System or Other Users . . . 794
Working with Archives . . . 794
Working with UUCP . . . 794

Appendix B. tcsh Shell Command Summary . . . 795
General Use . . . 795
Controlling Your Environment . . . 795
Managing Directories . . . 796
Computing and Managing Logic . . . 796
Managing Files . . . 796
Controlling Processes . . . 796

Appendix C. Regular Expressions (regexp) . . . 797
Summary . . . 800
Examples . . . 800

Appendix D. Running Shell Scripts or Executable Files under MVS Environments . . . 803
BPXBATCH . . . 803
Using OSHELL to Run Shell Commands and Scripts from MVS . . . 809

Appendix E. BPXCOPY: Copying a Sequential or Partitioned Data Set or PDSE Member into an HFS File . . . 811
BPXCOPY . . . 811

Appendix F. Localization . . . 817

Appendix G. Stub Commands . . . 819

Appendix H. File Formats . . . 821
cpio — Format of cpio archives . . . 821
magic — Format of the /etc/magic file . . . 822
pax — Format of pax archives and special header summary files . . . 824
queuedefs — Queue description for at, batch, and cron . . . 826
tags — Format of the tags file . . . 827
tar — Format of tar archives . . . 828

utmpx — Format of login accounting files 830
uucp — Format of UUCP working files 831

Appendix I. Setting the Local Time Zone with the TZ Environment Variable 837
TZ Environment Variable 837

Appendix J. Environment Variables 839

Appendix K. Specifying MVS Dataset Names in the Shell Environment . . 843
Utilities supporting MVS dataset names 844

Appendix L. Notices 845
Trademarks 846
Acknowledgments 847

Glossary 849

Index 875

Figures

1. Locales Supplied by OS/390 UNIX System Services 4
2. The Order of Operations for awk 30
3. **cp** Format: File to File and File ... (multiple files) to Directory 135
4. **cp** Format: File to File 136
5. **cp** Format: File... (multiple files) to Directory 136
6. Fields in the Configuration File (inetd daemon) 285
7. Explanation of the ipcs Command Listing 292
8. Internal Table Sizes (lex command) 307
9. **mv** Format: File to File and File ... (multiple files) to Directory 398
10. **mv** Format: File to File 398
11. **mv** Format: File... (multiple files) to Directory 399
12. Shell Operators (sh command) 499
13. Built-in Shell Variables (sh command) 507
14. Standard Input/Output Syntax for tcsh Shell 577
15. tcsh Built-in Shell Variables 587
16. tcsh Environment Variables 598
17. Various Formats of the OMVS CONVERT Command (OMVS command) 765
18. Locales, Their Conversion Tables, and Default Escape Characters (OMVS command) 766
19. Regular Expression Features (regexp) 800
20. Archive File: ASCII Header 822
21. Example of a Special Header Summary File 825
22. Archive File: UNIX-Compatible Format 828
23. Archive File: USTAR Format 829
24. Miscellaneous Environment Variables 839

© Copyright IBM Corp. 1996, 2000

Permuted Index

The permuted index is a guide to the command descriptions. It helps you explore the shell and TSO/E commands and what they can do for you.

Each line in the permuted index is taken from the title of a command—its "purpose line." The words of the purpose are shifted to get entries for the index. For example, the purpose of the **comm** shell command is:

comm — Show and select or reject lines common to two files

This produces the following permuted index entries:

and select or reject lines	common to two files. Show	**comm**
reject lines common to two	files. Show and select or	**comm**
Show and select or reject	lines common to two files.	**comm**
files. Show and select or	reject lines common to two	**comm**
to two files. Show and	select or reject lines common	**comm**
lines common to two files.	Show and select or reject	**comm**

The permuted index is sorted alphabetically in the middle column by each significant word in the title line. The first column contains whatever is left of the title line—either before or after the word or words in the middle column. The last column contains the place where you would go to find the information you are seeking. The purpose line of the title always begins after the command name. The purpose line always ends with a period.

The permuted index lets you look up commands according to significant words in their purpose lines. For example, suppose you want to show lines common to two files, and you don't know which command does this. Looking up the word *show* in the middle column of the permuted index, you find three entries:

lines common to two files.	Show and select or reject	**comm**
export attributes for variables, or	show currently exported variables. Set the	**export**
two text files and	show the differences. Compare	**diff**

Each index entry is associated with a command description (given in the third column). By looking at the purpose lines, you should be able to determine which command does what you want and go directly to the appropriate description—in this case, the **comm** command.

If you had looked in the permuted index under the word *file* or *files,* you would have found 47 entries. Again, by looking at the purpose lines, you should be able to determine which command does what you want. You could also have looked under *lines, reject,* or *select.* With any of these words, you would have found the appropriate command: **comm**.

a shell function or	**.** (dot) script. Return from	**return**
interrupts. Intercept	abnormal conditions and	**trap**
sent by uuto and uucp.	Manage files	**uupick**
Change the file	access and modification times.	**touch**
Configure the	AF_UEINT stack.	**configstk**
Create a tracked	alias.	**hash**
Remove	alias definitions.	**unalias**
create a command	alias. Display or	**alias**
Display	all current appointments.	**calendar**
Display	all or part of a message catalog.	**dspcat**
messages.	Allow or refuse	**mesg**

© Copyright IBM Corp. 1996, 2000

Call up	another system.	**cu**
Talk to	another user.	**talk**
Write to	another user.	**write**
current	appointments. Display all	**calendar**
language. Use the	arbitrary-precision arithmetic calculation	**bc**
up a file. Manipulate the tar	archive files to copy or back	**tar**
in/out file	archives. Copy	**cpio**
maintain library	archives. Create or	**ar**
portable	archives. Interchange	**pax**
a command. Construct an	argument list and run	**xargs**
expression. Evaluate	arguments as an	**expr**
shell. Return	arguments from the	**print**
output. Write	arguments to standard	**echo**
command by concatenating	arguments. Construct a	**eval**
Use the arbitrary-precision	arithmetic calculation language.	**bc**
Evaluate an	arithmetic expression.	**let**
Interpret	ASA/FORTRAN carriage control.	**asa**
Set or query	ASCII/EBCDIC code pages for the terminal.	**chcp**
values to functions.	Assign attributes and	**functions**
values to variables.	Assign attributes and	**typeset**
to functions. Assign	attributes and values	**functions**
to variables. Assign	attributes and values	**typeset**
Change the mount	attributes of a file system.	**chmount**
functions. Unset values and	attributes of variables and	**unset**
directory names and	attributes. List file and	**ls**
a file. Change	audit flags for	**chaudit**
Configure the	automount facility.	**automount**
written in the	awk language. Process programs	**awk**
tar archive files to copy or	back up a file. Manipulate the	**tar**
Schedule regular	background jobs.	**crontab**
job to the	background. Move a	**bg**
page-by-page	basis. Display files on a	**more**
Copy files	between remote UUCP systems.	**uucp**
Create	binary conversion tables.	**uconvdef**
a transmitted	binary file. Decode	**uudecode**
printable strings in	binary files. Display	**strings**
Compute checksum and	block count for file.	**sum**
into the foreground.	Bring a job	**fg**
to logged-in users.	Broadcast a message	**wall**
HFS file.	Browse an	OBROWSE
of files.	Build a list	OSTEPLIB
when system is not	busy. Run commands	**batch**
write checksums and	byte counts. Calculate and	**cksum**
words, and	bytes. Count newlines,	**wc**
an executable file. Compile	C/MVS source code and create	**c89/cc/c++**
and byte counts.	Calculate and write checksums	**cksum**
the arbitrary-precision arithmetic	calculation language. Use	**bc**
Display	calendar for a month or year.	**cal**
another system.	Call up	**cu**
from remote UUCP systems.	Carry out command requests	**uuxqt**
Interpret ASA/FORTRAN	carriage control.	**asa**
from other sites.	Carry out command requests	**uuxqt**
modify message	catalogs. Create or	**gencat**
part of the message	catalog. Display all or	**dspcat**
diff output.	Change a file using	**patch**
for a file.	Change audit flags	**chaudit**
terminals.	Change characteristics of	**tput**
of a running process.	Change priorities	**renice**
working directory.	Change the	**cd**
and modification times.	Change the file access	**touch**
a file or directory.	Change the group owner of	**chgrp**
a file or directory.	Change the mode of	**chmod**
of a file system.	Change the mount attributes	**chmount**
of a file or directory.	Change the owner or group	**chown**
for the execution of a command.	Change the root directory	**chroot**
new group.	Change to a	**newgrp**
passwords.	Change user	**password**

Create a	character special file.	MKNOD
a FIFO or	character special file. Make	**mknod**
Translate	characters.	**tr**
to another. Convert	characters from one code set	**iconv**
pathname.	Check a	**pathchk**
Compute	checksum and block count for file.	**sum**
Calculate and write	checksums and byte counts.	**cksum**
Get process and	child process times.	**times**
Wait for a	child process to end.	**wait**
Run a command and open,	close, or copy the file descriptors.	**exec**
Compile C/MVS source	code and create an executable file.	**c89/cc/c++**
Set or query	code pages for the terminal.	**chcp**
Convert characters from one	code set to another.	**iconv**
Display reason	code text.	**bpxmtext**
nonzero exit	code. Return a	**false**
files by Huffman	coding. Compress	**pack**
or create a	command alias. Display	**alias**
in the shell. Interactive	command and history editing	**shedit**
the file descriptors. Run a	command and open, close, or copy	**exec**
Run a	command at a different priority.	**nice**
Run a	command at a specified time.	**at**
arguments. Construct a	command by concatenating	**eval**
Request	command execution on remote UUCP systems.	**uux**
Run a TSO/E	command from the shell.	**TSO**
Process a	command history list.	**fc**
Process a	command history list.	**history**
Process a	command history list.	**r**
the shell interprets a	command name. Tell how	**whence**
parameters. Set or unset	command options and positional	**set**
Carry out	command requests from remote UUCP systems.	**uuxqt**
for the execution of a	command. Change the root directory	**chroot**
list and run a	command. Construct an argument	**xargs**
elapsed times for a	command. Display processor and	**time**
a simple	command. Run	**command**
Run shell	commands, shell scripts, or executable files.	BPXBATCH
is not busy. Run	commands when system	**batch**
and select or reject lines	common to two files. Show	**comm**
Report status of the interprocess	communication facility.	**ipcs**
directories.	Compare	**dircmp**
the terminal description.	Compare and print	**infocmp**
files.	Compare two	**cmp**
and show the differences.	Compare two text files	**diff**
terminfo database entries.	Compile term descriptions into	**tic**
configuration files.	Compile UUCP	**uucc**
and create an executable file.	Compile C/MVS source code	**c89/cc/c++**
the yacc	compiler. Use	**yacc**
Return the nondirectory	components of a pathname.	**basename**
Return the directory	components of a pathname.	**dirname**
spaces into tabs	Compress	**unexpand**
by Hoffman coding	Compress files	**pack**
Lempel-Ziv file	compression.	**compress**
Undo Lempel-Ziv	compression of a file.	**uncompress**
block count for file.	Compute checksum and	**sum**
a text file.	Concatenate or display	**cat**
a command by	concatenating arguments. Construct	**eval**
for a	condition. Test	**test**
Intercept abnormal	conditions and interrupts.	**trap**
Get	configuration values.	**getconf**
Compile UUCP	configuration files.	**uucc**
AF_EINT stack.	Configure the	**configstk**
query an OCS object.	Configure, unconfigure or	**ocsconfig**
automount facility.	Configure the	**automount**
Invoke uucico for TCP/IP	connections from remote UUCP systems.	**uucpd**
by concatenating arguments.	Construct a command	**eval**
and run a command.	Construct an argument list	**xargs**
ASA/FORTRAN carriage	control. Interpret	**asa**
a file.	Convert and copy	**dd**

Create binary	conversion tables	**uconvdef**
on remote UUCP systems.	Copy files to users	**uuto**
from one code set to another.	Convert characters	**iconv**
file.	Copy a	**cp**
Convert and	copy a file.	**dd**
an MVS data set.	Copy an HFS file into	OGET
member into an HFS file.	Copy an MVS data set	OPUT
member or HFS file to another member or file.	Copy an MVS data set	OCOPY
between remote UUCP systems.	Copy files	**uucp**
to users on remote UUCP systems.	Copy files	**uuto**
partitioned data set (PDS) or PDSE.	Copy HFS file from a directory to an MVS	OGETX
PDSE to an HFS directory.	Copy members from an MVS PDS or	OPUTX
file archives.	Copy in/out	**cpio**
standard input.	Copy one line of	**line**
Manipulate the tar archive files to	copy or back up a file.	**tar**
command and open, close, or	copy the file descriptors. Run a	**exec**
of a file. Merge	corresponding or subsequent lines	**paste**
checksum and block	count for file. Compute	**sum**
and bytes.	Count newlines, words,	**wc**
checksums and byte	counts. Calculate and write	**cksum**
special file.	Create a character	MKNOD
alias. Display or	create a command	**alias**
to a file.	Create a link	**ln**
C/MVS source code and	create an executable file. Compile	**c89/cc/c++**
alias.	Create a tracked	**hash**
tables.	Create binary conversion	**uconvdef**
to a file.	Create a hard link	**link**
library archives.	Create or maintain	**ar**
message catalogs.	Create or modify	**gencat**
ex, **more**, and **vi**.	Create tag files for	**ctags**
return the file mode	creation mask. Set or	**umask**
file meeting specified	criteria. Find a	**find**
Display all	current appointments.	**calendar**
shell file in the	current environment. Run a	(dot)
shell file in the	current environment. Run a	**.**
the name of the	current operating system. Display	**uname**
to the	current process. Send a SIGSTOP	**suspend**
of jobs in the	current session. Return the status	**jobs**
information about	current users. Display	**who**
each line of a file.	Cut out selected fields from	**cut**
file recovery.	daemon. **vi**	**exrecover**
display	data. Uncompress and	**zcat**
term descriptions into	database entries. Compile	**tic**
from a directory to an MVS partitioned	data set (PDS) or PDSE. Copy HFS files	OGETX
HFS file. Copy an MVS	data set member into an	OPUT
member or file. Copy an MVS	data set member or HFS file to another	OCOPY
file into an MVS	data set. Copy an HFS	OGET
sorted, textual relational	databases. Join two	**join**
Display the	date and time.	**date**
the	debugger. Use	**dbx**
binary file.	Decode a transmitted	**uudecode**
packed files	Decode Huffman	**unpack**
locale environment.	Define the	**localedef**
name not	defined. Indicate function	**autoload**
alias	definitions. Remove	**unalias**
print the terminal	description. Compare and	**infocmp**
Compile term	descriptions into terminfo database entries.	**tic**
close, or copy the file	descriptors. Run a command and open,	**exec**
in files.	Detect spelling errors	**spell**
type.	Determine file	**file**
Change a file using	diff output.	**patch**
files and show the	differences. Compare two text	**diff**
Run commands at a	different priority.	**nice**
Compare	directories.	**dircmp**
pathname. Return the	directory components of a	**dirname**
command. Change the root	directory for the execution of a	**chroot**
Remove a	directory entry.	**rm**

Removes a	directory entry.	**unlink**
the working	directory name. Return	**pwd**
List file and	directory names and attributes.	**ls**
(PDS) or PDSE. Copy HFS files from a	directory to an MVS partitioned data set	OGETX
the working	directory. Change	**cd**
of a file or	directory. Change the group owner	**chgrp**
of a file or	directory. Change the mode	**chmod**
group of a file or	directory. Change the owner or	**chown**
PDS or PDSE to an HFS	directory. Copy members from an MVS	OPUTX
a	directory. Make	**mkdir**
a	directory. Make	MKDIR
a	directory. Remove	**rmdir**
a file or	directory. Rename or move	**mv**
for a month or year	Display a calendar	**cal**
appointments.	Display all current	**calendar**
message catalog.	Display all or part of a	**dspcat**
from message catalogs.	Display selected messages	**dspmsg**
environment for a process.	Display environments, or set an	**env**
interactively.	Display files	**pg**
page-by-page basis.	Display files on a	**zcat**
about current users.	Display information	**who**
UUCP systems.	Display list of	**uuname**
about UUCP events.	Display log information	**uulog**
publications.	Display online	OHELP
a command alias.	Display or create	**alias**
in binary files.	Display printable strings	**strings**
times for a command.	Display processor and elapsed	**time**
text.	Display reason code	**bpxmtext**
online reference manual.	Display sections of the	**man**
symbols.	Display static system	**sysvar**
UUCP transfers.	Display status of pending	**uustat**
executable files.	Display symbol table of object, library, or	**nm**
space in the file system.	Display the amount of free	**df**
and time.	Display the date	**date**
of a file.	Display the first part	**head**
of a file.	Display the last part	**tail**
the current operating system.	Display the name of	**uname**
pending UUCP transfers.	Display the status of	**uustat**
user name.	Display your effective	**whoami**
environment variables.	Display the values of	**printenv**
Uncompress and	display data.	**zcat**
Unpack and	display Huffman packed files.	**pcat**
Set or	display terminal options.	**stty**
Concatenate or	display text files.	**cat**
Use the	display-oriented text editor.	**vi**
a specified format.	Dump a file in	**od**
output stream.	Duplicate the	**tee**
Cut out selected fields from	each line of a file.	**cut**
Integer value. Mark	each variable with an	**integer**
Set or query ASCII/	EBCDIC code pages for the terminal.	**chcp**
editor. Use the	ed line-oriented text	**ed**
HFS file.	Edit an	OEDIT
command and history	editing in the shell. Interactive	**shedit**
sed noninteractive stream	editor. Start the	**sed**
interactive text	editor. Use the display-oriented	**vi**
line-oriented text	editor. Use the ed	**ed**
Display your	effective user name.	**whoami**
text	editor. Use the ex	**ex**
command. Display processor and	elapsed times for a	**time**
or receive	electronic mail. Send	**mailx**
for safe transmission.	Encode a file	**uuencode**
or send it a signal.	End a process or job,	**kill**
child process to	end. Wait for a	**wait**
into terminfo database	entries. Compile term descriptions	**tic**
Print the terminal	entries in the termcap file.	**captoinfo**
a directory	entry. Remove	**rm**
a directory	entry. Removes	**unlink**

environments, or set an	environment for a process. Display	**env**
the locale	environment. Define	**localedef**
file in the current	environment. Run a shell	(dot)
file in the current	environment. Run a shell	**.**
for a process. Display	environments, or set an environment	**env**
Display the values of	environment variables.	**printenv**
Detect spelling	errors in files.	**spell**
arithmetic expression.	Evaluate an	**let**
an expression.	Evaluate arguments as	**expr**
about UUCP	events. Display log information	**uulog**
Create tag files for	**ex**, **more**, and **vi**.	**ctags**
Use the	ex text editor.	**ex**
C/MVS source code and create an	executable file. Compile	**c89/cc/c++**
information from an	executable file. Remove unnecessary	**strip**
object, library, or	executable files. Display symbol table of	**nm**
Run shell commands, shell scripts, or	executable files.	BPXBATCH
the root directory for the	execution of a command. Change	**chroot**
an interval of time. Suspend	execution of a process for	**sleep**
Request command	execution on remote UUCP systems.	**uux**
a nonzero	exit code. Return	**false**
in the shell script.	Exit from a loop	**break**
to spaces.	Expand tabs	**expand**
variable for	export. Set a	**export**
an arithmetic	expression. Evaluate	**let**
as an	expression. Evaluate arguments	**expr**
Set, reset, and display	extended attributes for files.	**extattr**
Configure the automount	facility.	**automount**
interprocess communication	facility. Report status of the	**ipcrm**
regular file	feeds. Remove	**col**
a file. Cut out selected	fields from each line of	**cut**
file. Make a	FIFO or character special	**mknod**
Make a	FIFO special file.	**mkfifo**
times. Change the	file access and modification	**touch**
and attributes. List	file and directory names	**ls**
Copy in/out	file archives.	**cpio**
Manage	file caches.	**filecache**
Lempel-Ziv	file compression.	**compress**
open, close, or copy the	file descriptors. Run a command and	**exec**
pattern. Search a	file for a specified	**egrep**
pattern. Search a	file for a specified	**fgrep**
pattern. Search a	file for a specified	**grep**
transmission. Encode a	file for ssafe	**uuencode**
file system from the	file hierarchy. Remove a	**unmount**
file system from the	file hierarchy. Remove a	UNMOUNT
format. Dump a	file in a specified	**od**
it to standard output. Format a	file in paginated form and send	**pr**
environment. Run a shell	file in the current	(dot)
environment. Run a shell	file in the current	**.**
set. Copy an HFS	file into an MVS data	OGET
Split a	file into manageable pieces.	**split**
criteria. Find a	file meeting specified	**find**
Set or return the	file mode creation mask.	**umask**
the mode of a	file or directory. Change	**chmod**
group owner of a	file or directory. Change the	**chgrp**
owner or group of a	file or directory. Change the	**chown**
or move a	file or directory. Rename	**mv**
vi	file recovery daemon.	**exrecover**
Summarize usage of	file space.	**du**
attributes of a	file system. Change the mount	**chmount**
of free space in the	file system. Display the amount	**df**
mount a	file system. Logically	**mount**
mount a	file system. Logically	MOUNT
Remove a	file system from the file hierarchy.	**unmount**
Remove a	file system from the file hierarchy.	UNMOUNT
printer. Send a	file to a	**lp**
an MVS data set member or HFS	file to another member or file. Copy	OCOPY
Process UUCP	file transfer requests.	**uucico**

Determine	file type.	**file**
Change a	file using diff output.	**patch**
an HFS	file. Browse	OBROWSE
flags for a	file. Change audit	**chaudit**
and create an executable	file. Compile C/MVS source code	**c89/cc/c++**
block count for	file. Compute checksum and	**sum**
display a text	file. Concatenate or	**cat**
copy a	file. Convert and	**dd**
a	file. Copy	**cp**
set member into an HFS	file. Copy an MVS data	OPUT
HFS file to another member or	file. Copy an MVS data set member or	OCOPY
link to a	file. Create a	**ln**
character special	file. Create a	MKNOD
hard link to a	file. Create a	**link**
from each line of a	file. Cut out selected fields	**cut**
transmitted binary	file. Decode a	**uudecode**
part of a	file. Display the first	**head**
part of a	file. Display the last	**tail**
an HFS	file. Edit	OEDIT
FIFO special	file. Make a	**mkfifo**
or character special	file. Make a FIFO	**mknod**
to copy or back up a	file. Manipulate the tar archive files	**tar**
subsequent lines of a	file. Merge corresponding or	**paste**
in a	file. Number lines	**nl**
in the termcap	file. Print the terminal entries	**captoinfo**
message source	file. Preprocess a	**mkcatdefs**
from an executable	file. Remove unnecessary information	**strip**
repeated lines in a	file. Report or filter out	**uniq**
of a	file. Undo Lempel-Ziv compression	**uncompress**
differences. Compare two text	files and show the	**diff**
UUCP systems. Copy	files between remote	**uucp**
Compress	files by Huffman coding.	**pack**
partitioned data set (PDS) or PDSE. Copy HFS	files from a directory to an MVS	OGETX
Display	files interactively.	**pg**
Create tag	files for **ex**, **more** and **vi**.	**ctags**
basis. Display	files on a page-by-page	**more**
Manage	files sent by uuto and uucp.	**uupick**
a file. Manipulate the tar archive	files to copy or back up	**tar**
Sort	files topologically.	**tsort**
on remote UUCP systems. Copy	files to users	**uuto**
Split	files.	**csplit**
list of	files. Build a	OSTEPLIB
two	files. Compare	**cmp**
configuration	files. Compile UUCP	**uucc**
Huffman packed	files. Decode	**unpack**
spelling errors in	files. Detect	**spell**
in binary	files. Display printable strings in	**strings**
library, or executable	Files. Display symbol table of object,	**nm**
processes with open files.	files. List process IDs of	**fuser**
and interdependent	files. Maintain program-generated	**make**
shell scripts, or executable	files. Run shell commands,	BPXBATCH
reject lines common to two	files. Show and select or	**comm**
Huffman packed files.	files. Unpack and display	**pcat**
a file. Report or	filter out repeated lines in	**uniq**
meeting specified criteria.	Find a file	**find**
file. Display the	first part of a	**head**
file. Change audit	flags for a	**chaudit**
job into the	foreground. Bring a	**fg**
Format a file in paginated	form and send it to standard output.	**pr**
and send it to standard output.	Format a file in paginated form	**pr**
in a specified	format. Dump a file	**od**
Write	formatted output.	**printf**
Interpret ASA/	FORTRAN carriage control.	**asa**
system. Display the amount of	free space in the file	**df**
Indicate	function name not defined.	**autoload**
Return from a shell	function or **.** (dot) script.	**return**
Assign attributes and values to	functions.	**functions**

vfs_pfsctl HFS	functions. Invoke	**confighfs**
attributes of variables and	functions. Unset values and	**unset**
mkcatdefs to	gencat. Pipe output from	**runcat**
for lexical tasks.	Generate a program	**lex**
directory. Change the owner or	group of a file or	**chown**
or directory. Change the	group owner of a file	**chgrp**
a new	group. Change to	**newgrp**
that is immune to	hangups. Start a utility	**nohup**
Create a	hard link to a file.	**link**
PDS or PDSE to an	HFS directory. Copy members from an MVS	OPUTX
data set. Copy an	HFS file into an MVS	OGET
Copy an MVS data set member or	HFS file to another member or file.	OCOPY
Browse an	HFS file.	OBROWSE
Edit an	HFS file.	OEDIT
data set member into an	HFS file. Copy an MVS	OPUT
partitioned data set (PDS) or PDSE. Copy	HFS files from a directory to an MVS	OGETX
Invoke vfs_pfsctl	HFS functions.	**confighfs**
from the file	hierarchy. Remove a file system	**unmount**
from the file	hierarchy. Remove a file system	UNMOUNT
Interactive command and	history editing in the shell.	**shedit**
a command	history list. Process	**fc**
a command	history list. Process	**history**
a command	history list. Process	**r**
a name. Tell	how the shell interprets	**type**
a command name. Tell	how the shell interprets	**whence**
Compress files by	Huffman coding.	**pack**
Decode	Huffman packed files.	**unpack**
Unpack and display	Huffman packed files.	**pcat**
the user	identity. Return	**id**
not defined.	Indicate function name	**autoload**
files. List process	IDs of processes with open	**fuser**
or shared memory	IDs. Remove message queues, semaphore sets,	**ipcrm**
a utility that is	immune to hangups. Start	**nohup**
Display	information about current users.	**who**
file. Remove unnecessary	information from an executable	**strip**
locale-specific	information. Get	**locale**
Display log	information on UUCP events.	**uucp**
of standard	input. Copy one line	**line**
locale-specific	information. Get	**locale**
archives. Copy	in/out file	**cpio**
line of standard	input. Copy one	**line**
line from standard	input. Read a	**read**
variable with an	integer value. Mark each	**integer**
in the shell.	Interactive command and history editing	**shedit**
files	interactively. Display	**pg**
Use the display-oriented	interactive text editor.	**vi**
and interrupts.	Intercept abnormal conditions	**trap**
archives.	Interchange portable	**pax**
program-generated and	interdependent files. Maintain	**make**
carriage control.	Interpret ASA/FORTRAN	**asa**
Tell how the shell	interprets a command name.	**whence**
Report status of the	interprocess communication facility.	**ipcrs**
how the shell	interprets a name. Tell	**type**
conditions and	interrupts. Intercept abnormal	**trap**
of a process for an	interval of time. Suspend execution	**sleep**
shell.	Invoke a	**sh**
tcsh shell.	Invoke a	**tcsh**
from remote UUCP systems.	Invoke uucico for TCP/IP connections	**uucpd**
HFS functions.	Invoke vfs_pfsctl	**confighfs**
ISPF shell.	Invoke the	ISHELL
shell.	Invoke the	OMVS
Invoke the	ISPF shell.	ISHELL
shell script. Skip to the next	iteration of a loop in a	**continue**
foreground. Bring a	job into the	**fg**
background. Move a	job to the	**bg**
signal. End a process or	job, or send it a	**kill**
Schedule regular background	jobs.	**crontab**

Return the status of	jobs in the current session.	**jobs**
textual relational databases.	Join two sorted,	**join**
in the awk	language. Process programs written	**awk**
arbitrary-precision arithmetic calculation	language. Use the	**bc**
file. Display the	last part of a	**tail**
of a file. Undo	Lempel-Ziv compression	**uncompress**
compression.	Lempel-Ziv file	**compress**
a program for	lexical tasks. Generate	**lex**
or maintain	library archives. Create	**ar**
Display symbol table of object,	library, or executable files.	**nm**
process	limits. Set	**ulimit**
Remove reverse	line feeds.	**col**
input. Read a	line from standard	**read**
out selected fields from each	line of a file. Cut	**cut**
standard input. Copy one	line of	**line**
Use the ed	line-oriented text editor.	**ed**
Show and select or reject	lines common to two files.	**comm**
Number	lines in a file.	**nl**
or filter out repeated	lines in a file. Report	**uniq**
lines. Break	lines into shorter	**fold**
Merge corresponding or subsequent	lines of a file.	**paste**
into shorter	lines. Break lines	**fold**
Create a hard	link to a file.	**link**
file. Create a	link to a	**ln**
command. Construct an argument	list and run a	**xargs**
names and attributes.	List file and directory	**ls**
Build a	list of files.	OSTEPLIB
Display	list of UUCP systems.	**uuname**
processes with open files.	list process IDs of	**fuser**
command history	list. Process a	**fc**
command history	list. Process a	**history**
command history	list. Process a	**r**
Define the	locale environment.	**localedef**
Get	locale-specific information.	**locale**
messages.	Log	**logger**
Broadcast a message to	logged-in users.	**wall**
UUCP events. Display	log information about	**uulog**
file system.	Logically mount a	**mount**
file system.	Logically mount a	MOUNT
support. Start	login monitor for OCS	**lm**
a user's	login name. Return	**logname**
Exit from a	loop in a shell script.	**break**
to the next iteration of a	loop in a shell script. Skip	**continue**
Read and send	mail messages.	**mail**
receive electronic	mail. Send or	**mailx**
Create or	maintain library archives.	**ar**
interdependent files.	Maintain program-generated and	**make**
directory.	Make a	**mkdir**
directory.	Make a	MKDIR
special file.	Make a FIFO	**mkfifo**
character special file.	Make a FIFO or	**mknod**
file caches.	Manage	**filecache**
sent by uuto and uucp.	Manage files	**uucpd**
Split a file into	manageable pieces.	**split**
copy or back up a file.	Manipulate the tar archive files to	**tar**
online reference	manual. Display sections of the	**man**
as read-only.	Mark a variable	**readonly**
with an integer value.	Mark each variable	**integer**
the file mode creation	mask. Set or return	**umask**
Find a file	meeting specified criteria.	**find**
Copy an MVS data set	member into an HFS file.	OPUT
set member or HFS file to another	member or file. Copy an MVS data	OCOPY
to another member or file. Copy an MVS data set	member or HFS file	OCOPY
to an HFS directory. Copy	members from an MVS PDS or PDSE	OPUTX
semaphore sets, or	memory IDs. Remove message queues,	**ipcrm**
lines of a file.	Merge corresponding or subsequent	**paste**
Display selected messages from	message catalogs.	**dspmsg**

Display all or part of a	message catalog.	**dspcat**
Create or modify	message catalogs.	**gencat**
Display selected	messages from message catalogs.	**dspmsg**
shared message IDs. Remove	message queues, semaphore sets, or	**ipcrm**
Preprocess a	message source file.	**mkcatdefs**
Broadcast a	message to logged-in users.	**wall**
Log	messages.	**logger**
refuse	messages. Allow or	**mesg**
send mail	messages. Read and	**mail**
Pipe output from	mkcatdefs to gencat.	**runcat**
or directory. Change the	mode of a file	**chmod**
Create or	modify message catalogs.	**gencat**
file access and	modification times. Change the	**touch**
Start login	monitor for OCS support.	**lm**
Display a calendar for a	month or year.	**cal**
Create tag files for **ex**,	**more**, and **vi**.	**ctags**
system. Logically	mount a file	**mount**
system. Logically	mount a file	MOUNT
file system. Change the	mount attributes of a	**chmount**
directory. Rename or	move a file or	**mv**
to the background.	Move a job	**bg**
an HFS file. Copy an	MVS data set member into	OPUT
another member or file. Copy an	MVS data set member or HFS file to	OCOPY
HFS file into an	MVS data set. Copy an	OGET
Copy HFS files from a directory to an	MVS partitioned data set (PDS) or PDSE.	OGETX
directory. Copy members from an	MVS PDS or PDSE to an HFS	OPUTX
online	publications. Display	OHELP
effective user	name. Display your	**whoami**
Indicate function	name not defined.	**autoload**
operating system. Display the	name of the current	**uname**
file and directory	names and attributes. List	**ls**
to a	new group. Change	**newgrp**
bytes. Count	newlines, words, and	**wc**
pathname. Return the	nondirectory components of a	**basename**
Start the sed	noninteractive stream editor.	**sed**
Return a	nonzero exit code.	**false**
function name	not defined. Indicate	**autoload**
Do	nothing, successfully.	(colon)
Do	nothing, successfully.	**:**
in a file.	Number lines	**nl**
or query an OCS	object. Configure, unconfigure,	**ocsconfig**
Display symbol table of	object, library, or executable files.	**nm**
or query an	OCS object. Configure, unconfigure,	**ocsconfig**
Start login monitor for	OCS support.	**lm**
Copy	one line of standard input.	**line**
publications. Display	online	OHELP
Display sections of the	online reference manual.	**man**
descriptors. Run a command and	open, close, or copy the file	**exec**
of processes with	open files. List process IDs	**fuser**
Display online	publications.	OHELP
Invoke the	&oshell;.	OMVS
name of the current	operating system. Display the	**uname**
Set or unset command	options and positional parameters.	**set**
utility	options. Parse	**getopts**
display terminal	options. Set or	**stty**
mkdefs to gencat. Pipe	output from	**runcat**
Duplicate the	output stream.	**tee**
using diff	output. Change a file	**patch**
in paginated form and send it to standard	output. Format a file	**pr**
formatted	output. Write	**printf**
to standard	output. Write arguments	**echo**
directory. Change the group	owner of a file or	**chgrp**
file or directory. Change the	owner or group of a	**chown**
Decode Huffman	packed files.	**unpack**
Unpack and display Huffman	packed files.	**pcat**
Display files on a	page-by-page basis.	**more**
Set or query ASCII/EBCDIC code	pages for the terminal.	**chcp**

standard output. Format a file in	paginated form and send it to	**pr**
command options and positional	parameters. Set or unset	**set**
positional	parameters. Shift	**shift**
Return to the shell's	parent process or to TSO/E.	**exit**
options.	Parse utility	**getopts**
Display all or	part of a message catalog.	**dspcat**
HFS files from a directory to an MVS	partitioned data set (PDS) or PDSE. Copy	OGETX
Change user	passwords.	**password**
a	pathname. Check	**pathchk**
components of a	pathname. Return the directory	**dirname**
for a specified	pattern. Search a file	**egrep**
for a specified	pattern. Search a file	**fgrep**
for a specified	pattern. Search a file	**grep**
directory. Copy members from an MVS	PDS or PDSE to an HFS	OPUTX
from a directory to an MVS	PDS or PDSE. Copy HFS files	OGETX
Display status of	pending UUCP requests.	**uustat**
Copy members from an MVS PDS or	PDSE to an HFS directory.	OPUTX
to an MVS partitioned data set (PDS) or	PDSE. Copy HFS files from a directory	OGETX
into manageable	pieces. Split a file	**split**
Configure the AF_UEINT	stack.	**configstk**
from mkcatdefs to gencat.	Pipe output	**runcat**
Display the status of	pending UUCP transfers.	**uustat**
Interchange	portable archives.	**pax**
Shift	positional parameters.	**shift**
unset command options and	positional parameters. Set or	**set**
message source file.	Preprocess a	**mkcatdefs**
Compare and	print the terminal description.	**infocmp**
in the termcap file.	Print the terminal entries	**captoinfo**
Display	printable strings in binary files.	**strings**
file to a	printer. Send a	**lp**
Change	priorities of a running process.	**renice**
at a different	priority. Run a command	**nice**
history list.	Process a command	**fc**
history list.	Process a command	**history**
history list.	Process a command	**r**
the awk language.	Process programs written in	**awk**
file transfer requests.	Process UUCP	**uucico**
process times. Get	process and child	**times**
time. Suspend execution of a	process for an interval of	**sleep**
open files. List	process IDs of processes with	**fuser**
to the shell's parent	process or to TSO/E. Return	**exit**
Set	process limits.	**ulimit**
it a signal. End a	process or job, or send	**kill**
process and child	process times. Get	**times**
for a child	process to end. Wait	**wait**
priorities of a running	process. Change	**renice**
an environment for a	process. Display environments, or set	**env**
status of a	process. Return the	**ps**
to a	process. Send a SIGSTOP	**stop**
to the current	process. Send a SIGSTOP	**suspend**
List process IDs of	processes with open files.	**fuser**
for a command. Display	processor and elapsed times	**time**
tasks. Generate a	program for lexical	**lex**
files. Maintain	program-generated and interdependent	**make**
awk language. Process	programs written in the	**awk**
OS/390	publications. Display online	OHELP
for the terminal. Set or	query ASCII/EBCDIC code pages	**chcp**
Configure, unconfigure, or	query an OCS object.	**ocsconfig**
Remove message	queues, semaphore sets, or shared memory IDs.	**ipcrm**
from standard input.	Read a line	**read**
mail messages.	Read and send	**mail**
variable as	read-only. Mark a	**readonly**
Display	reason code text.	**bpxmtext**
Send or	receive electronic mail.	**mailx**
vi file	recovery daemon.	**exrecover**
of the online	reference manual. Display sections	**man**
Allow or	refuse messages.	**mesg**

Schedule	regular background jobs.	**crontab**
files. Show and select or	reject lines common to two	**comm**
two sorted, textual	relational databases. Join	**join**
command requests from	remote UUCP systems. Carry out	**uuxqt**
files between	remote UUCP systems. Copy	**uucp**
files to users on	remote UUCP systems. Copy	**uuto**
for TCP/IP connections from	remote UUCP systems. Invoke uucico	**uucico**
command execution on	Remote UUCP systems. Request	**uux**
directory.	Remove a	**rmdir**
directory entry.	Remove a	**rm**
file system from the file hierarchy.	Remove a	**unmount**
file system from the file hierarchy.	Remove a	UNMOUNT
definitions.	Remove alias	**unalias**
shared memory IDs.	Remove message queues, semaphore sets, or	**ipcrm**
line feeds.	Remove reverse	**col**
an executable file.	Remove unnecessary information from	**strip**
directory entry.	Removes a	**unlink**
file or directory.	Rename or move a	**mv**
Report or filter out	repeated lines in a file.	**uniq**
lines in a file.	Report or filter out repeated	**uniq**
on remote UUCP systems.	Request command execution	**uux**
file transfer	requests. Process UUCP	**uucico**
Carry out command	requests from other sites.	**uuxqt**
Initiate UUCP	requests from TCP/IP from remote systems.	**uucpd**
interprocess communication facility.	Report status of the	**ipcrs**
exit code.	Return a nonzero	**false**
login name.	Return a user's	**logname**
of `0`.	Return a value	**true**
the shell.	Return arguments from	**print**
or **.** (dot) script.	Return from a shell function	**return**
user identity.	Return the	**id**
of a pathname.	Return the directory components	**dirname**
creation mask. Set or	return the file mode	**umask**
of a pathname.	Return the nondirectory components	**basename**
of a process.	Return the status	**ps**
in the current session.	Return the status of jobs	**jobs**
terminal name.	Return the user's	**tty**
directory name.	Return the working	**pwd**
parent process or to TSO/E.	Return to the shell's	**exit**
Remove	reverse line feeds.	**col**
of a command. Change the	root directory for the execution	**chroot**
simple command.	Run a	**command**
or copy the file descriptors.	Run a command and open, close,	**exec**
a different priority.	Run a command at	**nice**
in the current environment.	Run a shell file	(dot)
in the current environment.	Run a shell file	**.**
from the shell.	Run a TSO/E command	**tso**
system is not busy.	Run commands when	**batch**
shell scripts, or executable files.	Run shell commands,	BPXBATCH
an argument list and	run a command. Construct	**xargs**
Change priorities of a	running process.	**renice**
Encode a file for	safe transmission.	**eencode**
background jobs.	Schedule regular	**crontab**
in a shell	script. Exit from a loop	**break**
function or **.** (dot)	script. Return from a shell	**return**
of a loop in a shell	script. Skip to the next iteration	**continue**
Run shell commands, shell	scripts, or executable files.	BPXBATCH
a specified pattern.	Search a file for	**egrep**
a specified pattern.	Search a file for	**fgrep**
a specified pattern.	Search a file for	**grep**
manual. Display	sections of the online reference	**man**
editor. Start the	sed noninteractive stream	**sed**
to two files. Show and	select or reject lines common	**comm**
of a file. Cut out	selected fields from each line	**cut**
Display	selected messages from message catalogs.	**dspmsg**
Remove message queues	semaphore sets, or shared memory IDs.	**ipcrm**
to a printer.	Send a file	**lp**

to a process.	Send a SIGSTOP	**stop**
to the current process.	Send a SIGSTOP	**suspend**
a process or job, or	send it a signal. End	**kill**
file in paginated form and	send it to standard output. Format a	**pr**
Read and	send mail messages.	**mail**
electronic mail.	Send or receive	**mailx**
Manage files	sent by uuto and uucp.	**uupick**
jobs in the current	session. Return the status of	**jobs**
process. Display environments, or	set an environment for a	**env**
file. Copy an MVS data	set member into an HFS	OPUT
terminal options.	Set or display	**stty**
code pages for the terminal.	Set or query ASCII/EBCDIC	**chcp**
extended attributes for files.	Set, reset, and display	**extattr**
file mode creation mask.	Set or return the	**umask**
options and positional parameters.	Set or unset command	**set**
limits.	Set process	**ulimit**
stops.	Set tab	**tabs**
into an MVS data	set. Copy an HFS file	OGET
Remove message queues, semaphore	sets or shared memory IDs.	**ipcrs**
semaphore sets, or	shared memory IDs. Remove message queues,	**ipcrs**
or executable files. Run	shell commands, shell scripts,	BPXBATCH
current environment. Run a	shell file in the	(dot)
current environment. Run a	shell file in the	**.**
script. Return from a	shell function or **.** (dot)	**return**
Tell how the	shell interprets a name.	**type**
from a loop in a	shell script. Exit	**break**
iteration of a loop in a	shell script. Skip to the next	**continue**
Run shell commands,	shell scripts, or executable files.	BPXBATCH
TSO/E. Return to the	shell's parent process or to	**exit**
history editing in the	shell. Interactive command and	**shedit**
the ISPF	shell. Invoke	ISHELL
the OS/390	shell. Invoke	OMVS
a	shell. Invoke	**sh**
tcsh	shell. Invoke a	**tcsh**
from the	shell. Return arguments	**print**
from the	shell. Run a TSO/E command	**tso**
parameters.	Shift positional	**shift**
lines into	shorter lines. Break	**fold**
lines common to two files.	Show and select or reject	**comm**
two text files and	show the differences. Compare	**diff**
job, or send it a	signal. End a process or	**kill**
Send a	SIGSTOP to a process.	**stop**
process. Send a	SIGSTOP to the current	**suspend**
Run a	simple command.	**command**
from other	sites. Carry out command requests	**uuxqt**
a loop in a shell script.	Skip to the next iteration of	**continue**
Start the	sort-merge utility.	**sort**
databases. Join two	sorted, textual relational	**join**
topologically.	Sort files	**tsort**
executable file. Compile C/MVS	source code and create an	**c89/cc/c++**
Preprocess a message	source file.	**mkcatdefs**
usage of file	space. Summarize	**du**
Display the amount of free	space in the file system.	**df**
Expand tabs to	spaces.	**expand**
Compress	spaces into tabs.	**unexpand**
a character	special file. Create	MKNOD
a FIFO	special file. Make	**mkfifo**
FIFO or character	special file. Make a	**mknod**
a file meeting	specified criteria. Find	**find**
file in a	specified format. Dump a	**od**
file for a	specified pattern. Search a	**egrep**
file for a	specified pattern. Search a	**fgrep**
file for a	specified pattern. Search a	**grep**
command at a	specified time. Run a	**at**
Detect	spelling errors in files.	**spell**
into manageable pieces.	Split a file	**split**
files.	Split text	**csplit**

Configure the AF_UEINT	stack	**configstk.**
one line of	standard input. Copy	**line**
a line from	standard input. Read	**read**
arguments to	standard output. Write	**echo**
form and send it to	standard output. Format a file in paginated	**pr**
is immune to hangups.	Start a utility that	**nohup**
OCS support.	Start login monitor for	**lm**
sort-merge utility.	Start the	**sort**
Display	static system symbols.	**sysvar**
process. Return the	status of a	**ps**
current session. Return the	status of jobs in the	**jobs**
Display	status of pending UUCP transfers.	**uustat**
facility. Report	status of the interprocess communication	**ipcrm**
Set tab	stops.	**tabs**
the sed noninteractive	stream editor. Start	**sed**
the output	stream. Duplicate	**tee**
Display printable	strings in binary files.	**strings**
file. Merge corresponding or	subsequent lines of	**paste**
nothing,	successfully. Do	(colon)
nothing,	successfully. Do	(colon)
file space.	Summarize usage of	**du**
login monitor for OCS	support. Start	**lmu**
for an interval of time.	Suspend execution of a process	**sleep**
or executable files. Display	symbol table of object, library,	**nm**
Display static system	symbols.	**sysvar**
a file	system from the file hierarchy. Remove	**unmount**
a file	system from the file hierarchy. Remove	UNMOUNT
Run commands when	system is not busy.	**batch**
Display static	system symbols.	**sysvar**
another	system. Call up	**cu**
of a file	system. Change the mount attributes	**chmount**
free space in the file	system. Display the amount of	**df**
of the current operating	system. Display the name	**uname**
a file.	system. Logically mount	**mount**
a file	system. Logically mount	MOUNT
remote UUCP	systems. Carry out command requests from	**uucp**
remote UUCP	systems. Copy files between	**uucp**
remote UUCP	systems. Copy files to users on	**uucp**
from remote UUCP	systems. Invoke uucico for TCP/IP connections	**uucpd**
list of UUCP	systems. Display	**uuname**
remote UUCP systems.	systems. Request command execution on	**uux**
or executable files. Display symbol	table of object, library,	**nm**
Create binary conversion	tables.	**uconvdef**
Set	tab stops.	**tabs**
Compress spaces into	tabs.	**unexpand**
Set	tab stops.	**tabs**
Expand	tabs to spaces.	**expand**
Create	tag files for **ex**, **more**, and **vi**	**ctags**
another user.	Talk to	**talk**
back up a file. Manipulate the	tar archive files to copy or	**tar**
Invoke a	tcsh shell.	**tcsh**
terminal entries in the	termcap file. Print the	**captoinfo**
terminfo database entries. Compile	term descriptions into	**tic**
print the	terminal description. Compare and	**infocmp**
Print the	terminal entries in the termcap file.	**captoinfo**
the user's	terminal name. Return	**tty**
or display	terminal options. Set	**stty**
code pages for the	terminal. Set or query ASCII/EBCDIC	**chcp**
characteristics of	terminals. Change	**tput**
Compile term descriptions into	terminfo database entries.	**tic**
a condition.	Test for	**test**
reason code	text. Display	**bpxmtext**
display-oriented interactive	text editor. Use the	**vi**
ed line-oriented	text editor. Use the	**ed**
ex	text editor. Use the	**ex**
or display a	text file. Concatenate	**cat**
Split	text files.	**csplit**

the differences. Compare two	text files and show	**diff**
Join two sorted,	textual relational databases.	**join**
date and	time. Display the	**date**
at a specified	time. Run a command	**at**
process for an interval of	time. Suspend execution of a	**sleep**
Display processor and elapsed	times for a command.	**time**
access and modification	times. Change the file	**touch**
and child process	times. Get process	**times**
files	topologically. Sort	**tsort**
Create a	tracked alias.	**hash**
Process UUCP file	transfer requests.	**uucico**
pending UUCP	transfers. Display the status of	**uustat**
characters.	Translate	**tr**
for safe	transmission. Encode a file	**uuencode**
Decode a	transmitted binary file.	**uudecode**
Run a	TSO/E command from the shell.	**tso**
parent process or to	TSO/E. Return to the shell's	**exit**
Determine file	type.	**file**
display data.	Uncompress and	**zcat**
OCS object. Configure,	unconfigure, or query an	**ocsconfig**
of a file.	Undo Lempel-Ziv compression	**uncompress**
executable file. Remove	unnecessary information from an	**strip**
Huffman packed files.	Unpack and display	**pcat**
positional parameters. Set or	unset command options and	**set**
of variables and functions.	Unset values and attributes	**unset**
Call	up another system.	**cu**
Summarize	usage of file space	**du**
arithmetic calculation language.	Use the arbitrary-precision	**bc**
debugger.	Use the	**dbx**
text editor.	Use the display-oriented interactive	**vi**
line-oriented text editor.	Use the ed	**ed**
text editor.	Use the ex	**ex**
compiler.	Use the yacc	**yacc**
effective user name.	name. Display your	**whoami**
operating system. Display the	name of the current	**uname**
Change	user passwords.	**password**
to another	user. Talk to	**talk**
to another	user. Write to	**write**
message to logged-in	users. Broadcast a	**wall**
Copy files to	users on remote UUCP systems.	**uuto**
Return a	user's login name.	**logname**
Return the	user's terminal name.	**tty**
Parse	utility options.	**getopts**
to hangups. Start a	utility that is immune	**nohup**
the sort-merge	utility. Start	**sort**
remote UUCP systems. Invoke	uucico for TCP/IP connections from	**uucpd**
by uuto and	uucp. Manage files sent	**uuto**
Compile	UUCP configuraton files.	**uucc**
log information about	UUCP events. Display	**uulog**
Process	UUCP file transfer requests.	**uucico**
from remote	UUCP systems. Carry out command requests	**uuxqt**
remote	UUCP systems. Copy files between	**uucp**
remote	UUCP systems. Copy files to users	**uuto**
of	UUCP systems. Display list	**uuname**
TCP/IP connections from remote	UUCP systems. Invoke uucico for	**uucpd**
on remote	UUCP systems. Request command execution	**uux**
of pending	UUCP transfers. Display status	**uustat**
Manage files sent by	uuto and uucp.	**uuto**
Return a	value of 0.	**true**
variables and functions. Unset	values and attributes of	**unset**
Display the	values of environment variables.	**printenv**
Assign attributes and	values to functions.	**functions**
Assign attributes and	values to variables.	**typeset**
configuration	values. Get	**getconf**
Mark a	variable as read-only.	**readonly**
Set a	variable for export.	**export**
value. Mark each	variable with an integer	**integer**

values and attributes of	variables and functions. Unset	**unset**
and values to	variables. Assign attributes	**typeset**
values of environment	variables. Display the	**printenv**
ex, **more**, and	**vi**. Create tag files for	**ctags**
daemon.	**vi** file recovery	**exrecover**
Invoke	vfs_pfsctl HFS functions.	**confighfs**
process to end.	Wait for a child	**wait**
process from which the shell	was. Return to the parent	**exit**
to the parent process from	which the shell was. Return	**exit**
Count newlines,	words, and bytes.	**wc**
Return the	working directory name.	**pwd**
Change the	working directory.	**cd**
standard output.	Write arguments to	**echo**
counts. Calculate and	write checksums and byte	**cksum**
output.	Write formatted	**printf**
another user.	Write to	**write**
language. Process programs	written in the awk	**awk**
Use the	yacc compiler.	**yacc**
for a month or	year. Display a calendar	**cal**
Display	your effective user name.	**whoami**

About This Book

This book presents the information you need to use an OS/390 system with the shell and utilities feature as well as TSO/E (Time Sharing Option Extensions) commands for using OS/390 UNIX System Services (OS/390 UNIX). These features provide an application program interface (API) and a shell interface based on open systems standards.

OS/390 UNIX System Services (OS/390 UNIX) gives the OS/390 operating system an open standards interface. It consists of two features:

- **Shell and Utilities**, which you can use to enter shell commands, write shell scripts, and work with the file system.
- **Debugger**, which an application programmer can use to debug an OS/390 UNIX System Services application program written in C language.

This book describes how to use the shell commands, utilities, and TSO/E commands.

Who Should Use This Book

This book is for application programmers, system programmers, and end users working on an OS/390 system and using the shell.

Finding More Information about Other Products

Where to Find More Information

Where necessary, this book references information in other books about the elements and features of OS/390. For complete titles and order numbers for all OS/390 books, see *OS/390 Information Roadmap*.

Direct your request for copies of any IBM publication to your IBM representative or to the IBM branch office serving your locality.

There is also a toll-free customer support number (1-800-879-2755) available Monday through Friday from 6:30 a.m. through 5:00 p.m. Mountain Time. You can use this number to:

- Order or inquire about IBM publications
- Resolve any software manufacturing or delivery concerns
- Activate the program reorder form to provide faster and more convenient ordering of software updates

Softcopy Publications

The UNIX library is available on the *OS/390 Collection Kit*, SK2T-6700. This softcopy collection contains a set of OS/390 and related unlicensed product books. The CD-ROM collection includes the IBM Library Reader, a program that enables customers to read the softcopy books.

Softcopy OS/390 publications are also available for web-browsing at this URL:

© Copyright IBM Corp. 1996, 2000

http://www.ibm.com/s390/os390/

Select “Library.”

IBM Systems Center Publications

IBM systems centers produce redbooks that can be helpful in setting up and using UNIX services. You can order these publications through normal channels, or you can view them with a web browser from this URL:

http://www.redbooks.ibm.com

These books have not been subjected to any formal review nor have they been checked for technical accuracy, but they represent current product understanding (at the time of their publication) and provide valuable information on a wide range of UNIX topics. You must order them separately. A selected list of these books follows:

- *Selecting a Server — The Value of S/390,* SG24-4812
- *OS/390 Version 2 Release 6 UNIX System Services Implementation and Customization,* SG24-5178
- *OS/390 TCP/IP OpenEdition Implementation Guide,* SG24-2141. Written for OS/390 TCP/IP OpenEdition, a replacement for TCP/IP for MVS Version 3 Release 2 Application Feature.
- *Accessing OS/390 OpenEdition MVS from the Internet,* SG24-4721. Written for TCP/IP for MVS Version 3 Release 2 Application Feature.
- *MVS/ESA SP 5.2.2 OpenEdition MVS Installation and Customization Starter Kit,* SG24-4529
- *Porting Applications to the OpenEdition MVS Platform,* GG24–4473. This book was written for the OpenEdition MVS feature of MVS/ESA SP 5.1.

OS/390 UNIX Porting Information

There is a *Porting Guide* on the OS/390 UNIX porting page on the World Wide Web, at this URL:

http://www.ibm.com/s390/unix/bpxa1por.html

You can read the *Porting Guide* from the web or download it as a PDF file that you can view or print using Adobe Acrobat Reader. The *Porting Guide* covers a range of useful topics, including sizing a port, setting up a porting environment, ASCII-EBCDIC issues, performance, and much more.

The porting page also features a variety of porting tips, and lists porting resources that will help you in your port.

UNIX Courses

The following classroom courses are available:

- *UNIX System Services for OS/390 Implementation,* ESP25
- *Introduction to OS/390 UNIX Services*, ESP05
- *Exploiting OS/390 Through UNIX Apps,* ES74A
- *TCPIP Implementation for OS/390,* CB694
- *UNIX System Services for OS/390 DCE Implementation,* ESP26
- *Web Server Implementation on OS/390,* ES170

- *1999 OS/390 Expo and Performance Conference,* E5438

The availability of educational offerings changes. For current information on classroom courses and other offerings, see your IBM representative or call 1-800-IBM-TEACH (1-800-426-8322).

UNIX Home Page

The UNIX home page on the World Wide Web has the latest technical news, customer stories, tools, and FAQs (frequently asked questions). You can visit it at this URL:

http://www.ibm.com/s390/unix/

Some of the tools available from the web site are ported tools, and some are home-grown tools designed for UNIX. All this code works in our environment at the time we make it available, but is not officially supported. Each tool has a README file that describes the tool and any restrictions on its use.

The simplest way to reach these tools is through the UNIX home page. From the home page, click on **Tools and Toys**.

The code is also available from **www.ibm.com.s390** through **anonymous ftp**.

Restrictions

Because the tools are not officially supported,

- There are no guaranteed enhancements.
- No APARs can be accepted.

IBM TalkLink Conferencing Service

The UNIX forum, OPENMVS CFORUM, is available to customers through IBM's TalkLink offering. Customers and IBM developers participate in the discussion in this forum.

To preview or sign up for TalkLink, access this URL using a secure web browser:

http://www.ibmlink.ibm.com/talklink

Web Access:

1. Go to **http://www.ibmlink.ibm.com/talklink.**
2. Click on Logon.
3. After logon, select Martlink.
4. Choose S390 from the menu.
5. Choose MVS from the menu.
6. Choose MVSSYSTEM from the menu.
7. Choose MVSFORUMS from the menu.
8. Choose OPENMVS from the menu.

SNA Fastpath Access:

1. Logon to IBMLink.
2. From the IBMLink main menu, type TalkLink on the command line.
3. Type S390 on the command line.
4. Type OPENMVS on the command line.

Contact your IBM representative for information on TalkLink, DialIBM, or equivalent offerings for your country.

Discussion List

Customers and IBM participants also discuss UNIX on the **mvs-oe discussion list**. This list is not operated or sponsored by IBM.

To subscribe to the mvs-oe discussion so you can receive postings, send a note to:

listserv@vm.marist.edu

Include the following line in the body of the note, substituting your first name and last name as indicated:

```
subscribe mvs-oe first_name last_name
```

After you are subscribed, you will receive further instructions on how to use the mailing list.

Summary of Changes

Summary of Changes
for SC28-1892-09
OS/390 Version 2 Release 9

This book contains information previously presented in *OS/390 UNIX System Services Command Reference*, SC28-1892-08, which supports OS/390 Version 2 Release 8.

The following summarizes the changes to that information.

New Information

- New chapter on the **tcsh** command for the tcsh shell (see “tcsh — Invoke a C shell” on page 556); the following tcsh built-in commands are now documented:

%	**filetest**	**popd**	**uncomplete**
alloc	**glob**	**pushd**	**unhash**
bindkey	**hashstat**	**rehash**	**unlimit**
breaksw	**hup**	**repeat**	**unsetenv**
builtins	**limit**	**sched**	**watchlog**
bye	**login**	**setenv**	**where**
chdir	**logout**	**settc**	**which**
complete	**ls-F**	**setty**	
dirs	**notify**	**source**	
echotc	**onintr**	**telltc**	

- New **dbx** subcommands: **detach** and **readwritelock**
- New variables on **dbx set** subcommand: $dll_loads, $dll_loadstop, and $lv_events
- New shell commands: **fuser**, **sysvar**, **chmount**, **mount**, and **unmount**
- New usage note added to the MOUNT and UNMOUNT TSO commands
- New "Example" section added to **df** command and two additional fields (for sysplex) displayed on **df -v**
- New field on **ls –E**, **pax –E**, and **tar –E** which represents the setting of the shared library attribute
- New "Usage Note" section added to **ls**: adding trailing slash to a directory when using **ls** to display information on that directory
- Information added to **tar** "Description" and **pax -x** describing the need to use the USTAR (**-U**)format when saving and restoring external attributes
- New section added to **pax** command describing *special header files*
- New attribute, **l**, on the **extattr** command
- New modes on the **dbx: display memory** subcommand
- **ipcs –x** modified to return an additional field, **INFO** (information associated with message queue serialization using perform lock operation (PLO)); **ipcs –w** modified to display two additional fields, **MSGQPID** and **MSGQTYPE**
- New keywords added to BPXCOPY: SHARELIB|NOSHARELIB
- New alias BPXBATSL for BPXBATCH; information added to TSO BPXBATCH and BPXBATCH appendix
- New paramaters SYSNAME and AUTOMOVE|NOAUTOMOVE on the MOUNT TSO command (for shared HFS)
- New Appendix B: "tcsh Shell Command Summary"

© Copyright IBM Corp. 1996, 2000

Changed Information

- The following list contains existing OS/390 shell commands which have been updated with information pertaining to the tcsh shell:

: (colon)	**cd**	**exit**	**newgrp**	**shift**	**unalias**
@ (at)	**continue**	**fg**	**nice**	**stop**	**unset**
alias	**echo**	**history**	**nohup**	**suspend**	**wait**
bg	**eval**	**jobs**	**printenv**	**time**	
break	**exec**	**kill**	**set**	**umask**	

- **make** "Limits" section changed
- **chgrp –h** option description changed
- **chmod –h** option description changed
- **chown –h** option description changed
- **inetd** table for configuration file fields updated
- **man** command "Examples" section updated
- Changes made to UNMOUNT DRAIN and UNMOUNT REMOUNT descriptions (for sysplex)
- Appendix H, **tar** and **pax** format sections updated

Deleted Information

- Messages section removed from the **env** command description.

This book includes terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

Summary of Changes
for SC28-1892-08
OS/390 Version 2 Release 8

This book contains information previously presented in *OS/390 UNIX System Services Command Reference*, SC28-1892-07, which supports OS/390 Release 7.

The following summarizes the changes to that information.

New Information

- Support added to copy (**cp**) and move (**mv**) files from/to MVS datasets to/from UNIX files.
- Support has been added for double-square-bracket syntax (**[[**) for conditional expressions and autoloaded functions.
- New shell environment variable added: **FPATH**.
- New environment variable, recognized by **vi**, added: **TMP_VI**.
- Appendix J added: "Specifying MVS Dataset Names in the Shell Environment".
- Support added to pax and tar for reading and writing archives from and to MVS datasets
- Four new **dbx** subcommands added: **args**, **listfiles**, **listfuncs**, and **record**.
- New **dbx** command option: **–f**.
- New **su** command option: **–s**.
- For the **infocmp** command, information describing the option *term_names* has been added.
- Information describing the **j** absolute movement command added to **vi**.
- New **vi** Set Option Variable: **autoflash, af**
- Usage note added to TSO **UNMOUNT**.

- **–H** option added to the **ls** command.
- Five new keywords added to **tso** shell command.
- New fields added for **ipcs –x** and **–w**.

Changed Information

- "Long Output Format" section of **ls** command updated.
- **newgrp** command description updated.
- **vi** "Set Option Variables" section updated.
- The following shell pre-defined aliases have been given their own sections: **autoload**, **functions**, **history**, **integer**, **r**, **stop**, and **suspend**.
- **sh** "Built-In Shell Commands" section updated to include shell command aliases.
- **test** command options section rolled into description section.

Deleted Information

- Information deleted from **pre[serve]** description under **vi**.
- Information deleted from **newgrp** localization section.

There were also minor changes made to the following:

- TSO **MOUNT**
- Appendix D: BPXCOPY
- Appendix I: Environment Variables

Summary of Changes
for SC28-1892-07
as updated June, 1999
online only for SK2T-6700-13

The following changes appear only in the online version of this publication. A vertical bar (|) in the left margin indicates changes to the text.

Changed Information

The following summarizes changes in this edition.

confighfs was updated to clarify the **–l** option.

Summary of Changes
for SC28-1892-06
OS/390 Version 2 Release 7

This book contains information previously presented in *OS/390 UNIX System Services Command Reference*, SC28-1892-05, which supports OS/390 Release 6.

The following summarizes the changes to that information.

New Information

- Five new commands: **link**, **unlink**, **confighfs**, **chroot**, and **bpxmtext**
- New **onload** subcommand for **dbx**
- New **set** subcommand for **dbx** variables
- **ps** command now displays thread information
- New **pax** and **tar** command options for external links

- New signal: SIGDUMP
- New environment variable: _BPXK_MDUMP
- New BPXCOPY parameters: LINK, SYMLINK, SYMPATH, APF | NOAPF, PROGCTL | NOPROGCTL, SHAREAS | NOSHAREAS, UID, GID
- New MOUNT parameters: SECURITY | NOSECURITY
- Two new keyword options on PARM statement of MOUNT command: SYNC(t) and NOWRITEPROTECT
- Use of the **man** command to view descriptions of TSO/E commands and **dbx** subcommands
- New **automount** keyword: **security**
- Examples added to **ln** command description
- Usage notes section added to **tar** command
- Usage notes added to **su** command
- Usage note added to **cpio** command
- Usage note added to **pax** command
- Usage note added to **dot** command
- Option *-v* added to the **rm** command

Changed Information

- **dbx** command description updated
- **sh** built-in variables chart updated
- **ln** options section updated
- UNMOUNT command description updated
- **su** command description updated
- **sort** "Sorting Key" section updated
- **chown** and **chgrp** "Portability" sections updated
- **tic** command description, "Options", and "Examples" sections updated
- **exrecover daemon** "Environment Variables" section updated
- **rlogind** "Usage Notes" section updated
- **chmod** "Portability" section updated
- Appendix A: "Shell Command Summary" updated

There were also minor changes made to the following OS/390 UNIX commands:

- **patch**
- **kill**
- **passwd**

Summary of Changes
for SC28-1892-05
OS/390 Version 2 Release 6

This book contains information previously presented in *OS/390 UNIX System Services Command Reference* SC28-1892-04, which supports OS/390 Release 5.

The following summarizes the changes to that information.

Name Change

As part of the name change of OpenEdition to OS/390 UNIX System Services, occurrences of OS/390 OpenEdition have been changed to OS/390 UNIX System Services or its abbreviated name, OS/390 UNIX. OpenEdition may continue to appear in messages, panel text, and other code with OS/390 UNIX System Services.

New Information

- New BPXBATCH environment variables: **_BPX_BATCH_UMASK** and **_BPX_BATCH_SPAWN**
- **locale** command keyword list
- Support has been added for IEEE. Complete support for IEEE binary floating-point requires PTFs to OS/390 R6 and specific releases of some software. See *OS/390 Planning for Installation* for the software requirements of IEEE floating point.
- Information added to **dbx** subcommand **set** and to the **od** shell command for IEEE requirements.

Changed Information

- BPXBATCH max parameter length increased
- Information on **_BPX_USERID**, **_CEE_ENVFILE** and **_BPX_PTRACE_ATTACH** environment variables added to Appendix I
- **passwd** command description updated
- **inetd** usage notes section updated
- **TSO/E MKDIR** format updated
- **sh** "Enviroment Varibles" section changed to "Shell Variables" and section was updated
- **stty** command description and operand descriptions updated
- **su** command description updated

There were also minor changes made to the following OS/390 UNIX commands:

- **c89**
- **expr**
- **file**
- **find**
- **mailx**
- **nm**
- **pax**
- **rlogind**
- **rm**
- **tso**
- **umask**
- **uname**
- **uulog**
- **wall**
- **wc**
- **write**
- BPXCOPY

Summary of Changes
for SC28-1892-04
OS/390 Version 2 Release 5

This book contains information previously presented in *OpenEdition Command Reference* SC28-1892-03, which supports OS/390 Release 4.

The following summarizes the changes to that information.

New Information

- **passwd** command
- **wall** command
- **whoami** command
- A new option (**–h**) for the **chgrp**, **chmod**, and **chown** commands
- Updates and new examples for the **pax** command

TCP/IP OpenEdition commands have been included in Part 2, which contains the descriptions of the shell commands. The description of those commands contain a reference to the appropriate TCP/IP book.

Summary of Changes
for SC28-1892-03
OS/390 Version 2 Release 4

This book contains information previously presented in *OpenEdition Command Reference* SC28-1892-02, which supports OS/390 Release 3.

The following summarizes the changes to that information.

New Information

- **extattr** command
- **filecache** command
- New operand RUNOPTS added to the OMVS command
- New option (**–E**) added to the **ls** command

Summary of Changes
for SC28-1892-02
OS/390 Release 3

This book contains information previously presented in *OpenEdition Command Reference* SC28-1892-01, which supports OS/390 Release 2.

The following summarizes the changes to that information.

New Information

- **configstk** command
- **uconvdef** command
- New environment variable (**_BPXACCT_DATA**)

Changed Information

- An usage note was added to the MOUNT command.
- An usage note was added to the **newgroup** command.
- An usage note was added to the UNMOUNT command.

Summary of Changes
for SC28-1892-01
OS/390 Release 2

This book contains information previously presented in *OpenEdition Command Reference* SC28-1892-00, which supports OS/390 Release 1.

The following summarizes the changes to that information.

New Information

- Appendix containing a partial list of environment variables
- OS/390 Release 2 compiler support (including IPA optimization and run-time library support
- OSHELL TSO/E command
- **printenv** shell command
- **uucc** shell command
- **uucico** shell command
- **uucp** file format
- **uucpd** daemon
- **uulog** shell command
- **uuname** shell command
- **uupick** shell command
- **uuxqt** daemon

Changed Information

- **uucp** shell command
- **uustat** shell command
- **uuto** shell command
- **uux** shell command

/etc/init is now **/usr/sbin/init**.

Summary of Changes
for SC28-1892-00
OS/390 Release 1

This book contains information previously presented in *OpenEdition MVS Command Reference* SC28-3014-02, which supports MVS/ESA Version 5.

The following summarizes the changes to that information.

New Information

- **nm**
- **ctags**
- **exrecover**

Changed Information

- **c89/cc/c++**
- **set**
- **tso**

Part 1. Introduction to Shell Commands and DBCS

This part is an introduction to the shell commands and the doublebyte character set (DBCS).

Reading the Command Descriptions

Each shell command appears in alphabetic order. The description for each command is divided into several sections, which are explained in the following paragraphs. Some of these sections apply only to a few command descriptions. Also, some command descriptions include special sections that are not explained here.

Format Section

The *Format* section provides a quick summary of the command's format, or syntax. The syntax was chosen to conform to general UNIX usage. For example, here is the format of the **ls** command:

ls [**–AabCcdFfgiLlmnopqRrstuWx1**] [*pathname* **...**]

The format takes the form of a command line as you might type it into the system; it shows what you can type in and the order in which you should do it. The parts enclosed in square brackets are *optional*; you can omit them if you choose. Parts outside the square brackets *must* be present for the command to be correct.

The format begins with the name of the command itself. Command names always appear in **bold Courier (typewriter)** font.

After the command name comes a list of options, if there are any. A typical OS/390 shell command option consists of a dash (**–**) followed by a single character, usually an uppercase or lowercase letter. For example, you might have **–A** or **–a**.

Note: The case of letters is important; for example, in the format of **ls**, **–a** and **–A** are *different* options, with different effects.

If you are going to specify several options for the same command, you can put all the option characters after the same dash. Or you can put each option after its own dash. Or you can rearrange the order of options. For example,

ls -A -a ls -Aa ls -a -A ls -aA

are all equivalent.

The format line shows options like in **bold Courier (typewriter)** font. In the description of **ls**, all options are shown in one long string after the single dash. But another common option form is:

-x *value*

where **–x** is a dash followed by a character, and *value* provides extra information for using that option. For example, here is the format for the **sort** command, which takes unsorted input and sorts it:

© Copyright IBM Corp. 1996, 2000

sort [**–cmu**]
[**–o** *outfile*]
[**–t** *char*]
[**–y***n*]
[**–z***n*]
[**–bdfiMnr**]

[**-k** *startpos*[**,***endpos*]] ...
[*file* ...]

sort [**–cmu**]
[**–o** *outfile*]
[**–t***char*]
[**–y***n*]
[**–z***n*]
[**–bdfiMnr**]

[**+***startposition*[**–***endposition*]] ...
[*file* ...]

You can see that there are two possibilities here; you would need to choose which of the two versions of **sort** met your requirements. In either possibility, however, we have the option:

–o *outfile*

This option tells the **sort** command where to save its sorted output. The form of the option is **–o**, followed by a space, followed by *outfile*. In a command format, anything appearing in *italic serif* font is a *placeholder* for information that you are expected to supply. Sometimes after the format, the kind of information expected in place of the placeholder is explained. In our **sort** example, *outfile* stands for the name of a file where you want **sort** to store its output. For example, if you wanted to store the output in the file **sorted.dat**, you would specify:

```
sort -o sorted.dat
```

(followed by the rest of the command).

The format for **sort** also contains an option of the form:

–t*char*

This is similar to the option form we were just discussing, except that there is no space between the **–t** and *char*. As before, *char* in italics is a placeholder; in this case, it stands for any single character. If you want to use the **–t** option for **sort**, you just type **–t** followed immediately by another character, as in:

```
sort -t:
```

In this case, we use a colon (**:**) in the position of the placeholder *char*.

The end of the **sort** format is:

[*file* ...]

This means a list of one or more filenames; the ellipsis (....) stands for repetitions of whatever immediately precedes it. Since there are square brackets around the previous list, you can omit the list if you like.

The format of **ls** ended in:

[*pathname* ...]

As you might guess, this means that an **ls** command can end with an optional list of one or more pathnames. What's the difference between this and our **sort** example? A pathname (specified with *pathname*) can be the name of either a file or a directory; a filename (specified with *file*) is always the name of a file.

The order of items on the command line is important. When you type a command line, you should specify its parts in the order they appear in the command format. The exceptions to this are options marked with a dash (**–**); they do not have to be given in the exact order shown in the format. However, all the **–** options must appear in the correct *area* of the command line. For example, you can specify:

```
ls -l -t myfiles
ls -t -l myfiles
```

but you won't get correct results if you specify:

```
ls myfiles -l -t          ***incorrect***
```

or:

```
ls -l myfiles -t          ***incorrect***
```

and so on. If you enter the last example, for instance, **ls** interprets **–t** as the pathname of a file or directory, and the command will try to list the characteristics of that item.

As a special notation, most OS/390 shell commands let you specify two dashes (**––**) to separate the options from the nonoption arguments; **––** means: “There are no more options.” Thus, if you really have a directory named **–t**, you could specify:

```
ls –– –t
```

to list the contents of that directory.

Description Section

The *Description* section describes what the command does. For a particularly complex command, this section may be divided into a large number of subsections, each dealing with a particular aspect of the command.

The Description section often mentions the *standard input* (**stdin**) and the *standard output* (**stdout**). The standard input is usually the workstation keyboard; the standard output is usually the display screen. The process of *redirection* can change this. Redirection is explained in *OS/390 UNIX System Services User's Guide*.

The shell differentiates between hex, octal, and decimal as follows:

- Any number that starts with 0x is hex.
- Any number that starts with 0 is octal.
- Any number that does not start with 0x or 0 is decimal.

Inside the Description section, the names of files and directories are presented in **normal bold** font. The names of environment variables are also presented in **NORMAL BOLD** font, capitalized.

Options Section

The *Options* section describes each of the options used by the command.

Examples Section

The *Examples* section is present in many command descriptions, giving examples of how the OS/390 shell can be used. This book tries to give a mix of simple examples that show how the commands work on an elementary level, and more complex examples that show how the commands can perform complicated tasks.

Trying the Examples in This Book

Before you try to run any of the examples in this book, you need to know that the OS/390 shell uses the EBCDIC *Latin1/Open System Interconnection Code Page* 01047. Characters entered on a workstation keyboard and passed to the shell by OS/390 do not have the same hexadecimal encoding as the code page the shell uses. You may need to customize your keyboard so that those characters have the encoding the shell uses. See *OS/390 UNIX System Services User's Guide*for more information about code page conversion, about using a keyboard with customized characters, and for a copy of code page 01047.

Environment Variables Section

The *Environment Variables* section lists the environment variables that affect the command, if any, and describes the purposes that those variables serve. For example, the **ls** command description lists two environment variables— **COLUMNS** and **TZ**—and informs you that **COLUMNS** is the terminal width and that **TZ** contains information about the local time zone.

Localization Section

All shell commands are affected by the following special localization variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

The *Localization* section describes how the locale-related environment variables affect the behavior of the command. These environment variables allow you to access *locale* information, including alternate character sets; alternate numeric, monetary, and date and time formats; and foreign language translations of common messages. Locales make it easier for users around the world to use the shell and utilities.

OS/390 UNIX System Services supports the IBM-supplied locales listed in Figure 1. User-generated locales using code page 1047 are also supported.

Figure 1 (Page 1 of 2). Locales Supplied by OS/390 UNIX System Services

Country	Language	Locale Name
Bulgaria	Bulgarian	Bg_BG.IBM-1025
Czech Republic	Czech	Cs_CZ.IBM-870
Denmark	Danish	Da_DK.IBM-277
Denmark	Danish	Da_DK.IBM-1047
Switzerland	German	De_CH.IBM-500
Switzerland	German	De_CH.IBM-1047
Germany	German	De_DE.IBM-273

Figure 1 (Page 2 of 2). Locales Supplied by OS/390 UNIX System Services

Country	Language	Locale Name
Germany	German	De_DE.IBM-1047
Greece	Ellinika	El_GR.IBM-875
United Kingdom	English	En_GB.IBM-285
United Kingdom	English	En_GB.IBM-1047
Japan	English	En_JP.IBM-1027
United States	English	En_US.IBM-037
United States	English	En_US.IBM-1047
Spain	Spanish	Es_ES.IBM-284
Spain	Spanish	Es_ES.IBM-1047
Finland	Finnish	Fi_FI.IBM-278
Finland	Finnish	Fi_FI.IBM-1047
Belgium	French	Fr_BE.IBM-500
Belgium	French	Fr_BE.IBM-1047
Canada	French	Fr_CA.IBM-037
Canada	French	Fr_CA.IBM-1047
Switzerland	French	Fr_CH.IBM-500
Switzerland	French	Fr_CH.IBM-1047
France	French	Fr_FR.IBM-297
France	French	Fr_FR.IBM-1047
Croatia	Croatian	Hr_HR.IBM-870
Hungary	Hungarian	Hu_HU.IBM-870
Iceland	Icelandic	Is_IS.IBM-871
Iceland	Icelandic	Is_IS.IBM-1047
Italy	Italian	It_IT.IBM-280
Italy	Italian	It_IT.IBM-1047
Israel	Hebrew	Iw_IL.IBM-424
Japan	Japanese	Ja_JP.IBM-939
Japan	Japanese	Ja_JP.IBM-1027
Korea	Korean	Ko_KR.IBM-933
Belgium	Dutch	Nl_BE.IBM-500
Belgium	Dutch	Nl_BE.IBM-1047
Netherlands	Dutch	Nl_NL.IBM-037
Netherlands	Dutch	Nl_NL.IBM-1047
Norway	Norwegian	No_NO.IBM-277
Norway	Norwegian	No_NO.IBM-1047
Poland	Polish	Pl_PL.IBM-870
Brazil	Brazilian	Pt_BR.IBM-037
Brazil	Brazilian	Pt_BR.IBM-1047
Portugal	Portugese	Pt_PT.IBM-037
Portugal	Portugese	Pt_PT.IBM-1047
Romania	Romanian	Ro_RO.IBM-870
Russia	Russian	Ru_RU.IBM-1025
Serbia	Serbian(Latin)	Sh_SP.IBM-870
Slovakia	Slovak	Sk_SK.IBM-870
Slovenia	Slovenian	Sl_SI.IBM-870
Serbia	Serbian(Cyrillic)	Sr_SP.IBM-1025
Sweden	Swedish	Sv_SE.IBM-278
Sweden	Swedish	Sv_SE.IBM-1047
Turkey	Turkish	Tr_TR.IBM-1026
People's Republic of China	Simplified Chinese	Zh_CN.IBM-935
Taiwan	Traditional Chinese	Zh_TW.IBM-937

For more information on locales, see Appendix F.

Files Section

The *Files* section lists any supplementary files (files not specified on the command line) that the command refers to. Such files usually provide information the command needs; the command accesses these files during its operation. If the files cannot be found, the command issues a message to this effect.

Files documented in this section may be temporary files, output files, databases, configuration files, and so on.

The OS/390 C runtime library supports a file naming convention of // (the filename can begin with exactly two slashes). However, OS/390 UNIX System Services *does not* support this convention. Do not use this convention (//) unless it is specifically indicated (as in the description for the **c89** command). OS/390 UNIX System Services does support the POSIX file naming convention, where the filename can be selected from the set of character values excluding the slash and the null character.

Usage Notes Section

The *Usage Notes* section gives additional notes for those using the shell. The purpose of the Usage Notes section is similar to that of the *Caution* section (see "Caution Section")—to provide important information that the reader should not overlook. However, the Usage Notes section usually deals with issues that are more benign than what the Caution section deals with.

Exit Values Section

The *Exit Values* section presents the error messages that the shell may display, along with a description of what caused the message and a possible action you can take to avoid getting that message. Occasionally, this section refers you to another command description for more information on an error message.

This section also contains information about the exit status returned by the command. You can test this status to determine the result of the operation that the command was asked to perform.

Limits Section

The *Limits* section lists any limits on the operation of the shell. Some limits are implicit rather than explicit and may be lower than the explicitly stated limit.

Portability Section

The *Portability* section includes two types of information:

- Availability of a version of the command on existing UNIX systems (System V, BSD)
- Compatibility with industry standards—for example, the POSIX.2 Draft Standard or the X/Open Portability Guide, Issue 4 (XPG4**).

Caution Section

The *Caution* section contains important advice for users. In OS/390 shell documentation, the *Caution* section is often aimed at those who are familiar with UNIX systems. Since the OS/390 shell primarily conforms to the emerging POSIX standards, its behavior may not precisely match the corresponding UNIX commands. The

Caution section may point out discrepancies in behavior that may catch experienced POSIX or UNIX users by surprise.

Related Information

The *Related Information* section refers to other command descriptions that may contain information relevant to the command description you have just read. For example, consider the **head** command; by default, **head** displays the first 10 lines of each file given on the command line. Its *Related Information* section refers you to **tail**, the command that displays the last 10 lines of a file.

Using the Doublebyte Character Set (DBCS)

OS/390 UNIX supports the doublebyte character set (DBCS). It also supports a DBCS locale. The name of the IBM-supplied DBCS locale is **Ja_JP**. This locale uses the IBM-939 coded-character set, which is a doublebyte character set.

This section discusses the following:

- Requirements for using DBCS
- When you must use SBCS characters and not DBCS characters
- When you can use DBCS characters
- Byte sequences that are not permitted in DBCS strings
- Displaying DBCS characters
- Switching locales
- Problems with DBCS filenames containing DBCS characters

Requirements for Using DBCS

If you plan to use DBCS interactively, you must work at a terminal that supports DBCS, such as a PS/55, and follow the procedures for the terminal emulator being used. It is not necessary, however, to be at a terminal that supports DBCS if you just want to use files that contain DBCS data.

To use DBCS, you need to do the following:

1. Specify special logmodes to access TSO/E and VTAM support for DBCS. Typically, the system programmer has already set these up and provided you with instructions.
2. Issue the TSO/E PROFILE PLANGUAGE(JPN) command, if required, to receive TSO/E messages in the Japanese language.
3. On the OMVS command, use the null character conversion table (the default) for character conversion. You do not need to specify the CONVERT operand on the OCOPY, OGETX, OPUT, and OPUTX commands.
4. Access the shell using the OMVS command with the DBCS operand (which is the default setting).

   You can also access the shell by using the rlogin program as well as OCS (Outboard Communication Server). The default conversion is from ISO8859-1 to IBM-1047; users can change their conversion to use different code pages by using the **chcp** command.
5. Define singlebyte escape characters for typing escape sequences.
6. Enable the shell and utilities for the DBCS locale, including having all shell and utility messages in Japanese, by entering the following commands:

```
export LC_ALL=Ja_JP
exec sh
```

To receive shell and utility messages in Japanese, but *not* put your terminal in DBCS mode, enter the following command:

```
export  LC_MESSAGES=Ja_JP
```

When You Must Use SBCS and Not DBCS Characters

You must use the singlebyte character set (SBCS) when specifying the following:

- User names.
- System, device, group, and terminal names.
- User names and passwords.
- Shell command-line options.
- Shell commands and their operands.
- Environment variables (DBCS characters are not exportable).
- Delimiters such as space, slash (/), curly brackets {}, tab, parentheses, dot (**.**), and any other shell special characters.
- Encoding for <newline> or null cannot be embedded in a DBCS character's code. There are other rules that define valid DBCS data:
  - The DBCS blank is 0x4040.
  - The first byte of the code defining the DBCS character must be in the range 0x41 to 0xFE.
  - The second byte must be in the range 0x41 to 0xFE.

  All others are invalid. This effectively covers the <newline> and null escape sequences, since they cannot be part of a valid DBCS character.

  For more information on invalid DBCS characters, see “Byte Sequences That Are Not Permitted in DBCS Strings” on page 9.
- Although filenames with DBCS characters are tolerated, you should not create filenames with DBCS characters. Doing so makes the file nonportable across locales, and problems may occur if filenames are subsequently used in a singlebyte locale. Instead, use the POSIX portable filename character set and singlebyte filenames.

  IBM will not support any customer problems with DBCS filenames.

  For more information on DBCS filenames, see “Problems with Filenames Containing DBCS Characters” on page 10.

When You Can Use DBCS Characters

When in the DBCS locale, you can use DBCS to specify the following:

- **sh** command-line arguments, although arguments expressed as numeric values must use SBCS characters.
- Text in data files. Files containing DBCS text are processed correctly by the shell and the utilities (such as **ed** and **grep**) if the DBCS locale is active. These files can be either DBCS text or mixed text (combinations of SBCS and DBCS). Both types of file can exist in the file system along with files that contain only singlebyte text.

Byte Sequences That Are Not Permitted in DBCS Strings

If you create invalid DBCS text, you may see an "illegal byte sequence" message when processing that text. The shell or command issues this error message, and the command stops processing in most cases.

Valid DBCS strings must start with "shift out" (SO [0x0E]) and end with "shift in" (SI [0x0F]). The first byte of the code defining the DBCS character must be in the range 0x41 to 0xFE. The second byte must be in the range 0x41 to 0xFE. The exception is that DBCS blank is 0x4040. All others codes are invalid.

Normal terminal operations do not produce invalid DBCS strings. To prevent invalid DBCS characters and strings:

- *Do not* use commands that operate on the data as byte strings instead of character strings. For example, **head** is a utility that could truncate a DBCS string or character in an inappropriate place, thus creating an invalid DBCS string. Using pipes between utilities can also result in invalid DBCS strings unless you pay attention to how each command handles the data.
- *Do not* edit text in nontext mode such as having the TSO/E editor with in HEX ON mode.

If the shell command is operating on a character string and not on a byte string, and the shell is in a locale that supports DBCS, and if the utility encounters an invalid DBCS string, such as the ones described in this section—you get an "illegal byte sequence" message and the utility may fail.

Note: <newline> (\n [0x15]) causes the shift state of any subsequent character sequence to start in the initial state (shifted into the SBCS mode). This may apply when a command is processing a DBCS string and encounters <newline> before a "shift in."

For information on rules for creating DBCS data, refer to *DBCS Design Guide—System/370 Software*, GG18-9095.

Displaying DBCS Characters

In a doublebyte environment, column positions are always based on the width of narrow characters. Normally, characters are "thin"; they take up only one column position when displayed. In contrast, some DBCS characters are "thick"; they take up two column positions when displayed.

The number of actual characters that are displayed by the command in the column area depends on the thickness of the characters. This applies to such commands as **ls**, **fold**, and **pr**, which display DBCS characters in column positions.

Switching Locales

By default, the shell starts in the POSIX locale and cannot handle DBCS text until the locale is changed, typically with the shell command export LC_ALL=Ja_JP. This **export** command affects the current shell environment with the following exception: if you change the locale to DBCS, the shell's **LC_CTYPE** locale category remains in the locale until is replaced via the **exec** command (**exec /bin/sh**).

Even if you change the locale to DBCS by using **export** LC_ALL=Ja_JP, the shell's LC_CTYPE variable remains in the previous locale (initially POSIX) until the shell is exec'd again with **exec sh**.

Always follow the **export** LC_ALL=*your locale* with **exec sh** to be sure the shell and utilities are running in the desired locale. This is true even if you place the **export** LC_ALL=*your_locale* in your login profile.

Problems with Filenames Containing DBCS Characters

The file system treats all filenames as if they contained SBCS characters. However, when you use the shell in the DBCS locale, filename and pathname comparison is performed in wide mode. That is, all the characters in the name are converted to wide characters before comparison. By doing this, the shift codes are removed from the comparison and, therefore, a match can be found with the filenames.

For example, if you have such DBCS filenames as:

```
db/so dbfile1 si
db/so dbfile2 si
```

where `so` and `si` are the shift codes that shift out to DBCS and back to SBCS, then when in the DBCS locale (Ja_JP),

```
ls db/so file si *
```

lists both files.

When in the POSIX locale, DBCS strings are treated as byte strings. Comparison is performed byte by byte. For example:

```
ls db/so file si *
```

shows the comparison string ending with an “e si”. The files in the directory would have to end with an “e si” in order to find a match. Neither of the filenames in the example would be found.

Part 2. Shell Command Descriptions

This part describes all the commands for the OS/390 shell. The descriptions are listed in alphabetic order. For instructions on how to read the command descriptions, see “Reading the Command Descriptions” on page 1.

The OS/390 shell is based on the KornShell that originated on a UNIX system. As implemented for OS/390 UNIX System Services, this shell conforms to POSIX standard 1003.2-1992.

Note: This book assumes that your OS/390 system includes the Resource Access Control Facility (RACF). Instead of RACF, your system could have an equivalent security product.

alias — Display or create a command alias

Format

alias [**–tx**] [*name*[=*value*] ...]
alias –r

tcsh shell: **alias** [*name* [*wordlist*]]

Description

When the first word of a shell command line is not a shell keyword, **alias** causes the shell to check for the word in the list of currently defined *aliases*. If it finds a match, the shell replaces the alias with its associated string value. The result is a new command line that might begin with a shell function name, a built-in command, an external command, or another alias.

When the shell performs alias substitution, it checks to see if *value* ends with a blank. If so, the shell also checks the next word of the command line for aliases. The shell then checks the new command line for aliases and expands them, following these same rules. This process continues until there are no aliases left on the command line, or recursion occurs in the expansion of aliases.

Calling **alias** without parameters displays all the currently defined aliases and their associated values. Values appear with appropriate quoting so that they are suitable for reinput to the shell.

Calling **alias** with parameters of the form *name*=*value* creates an alias for each *name* with the given string *value*.

If you are defining an alias where *value* contains a backslash character, you must precede it with another backslash. The shell interprets the backslash as the escape character when it performs the expansion. If you use double quotes to enclose *value*, you must precede each of the two backslashes with an additional backslash, because the shell escapes characters—that is, the shell does not interpret the character as it normally does—both when assigning the alias and again when expanding it.

© Copyright IBM Corp. 1996, 2000

To avoid using four backslashes to represent a single backslash, use single quotes rather than double quotes to enclose *value*, because the shell does not escape characters enclosed in single quotes during assignment. As a result, the shell escapes characters in single quotes only when expanding the alias.

Calling **alias** with *name* without any value assignment displays the function name (*name*) and its associated string value (*value*) with appropriate quoting.

DBCS Recommendation: We recommend that you use singlebyte characters when specifying an alias name, because the POSIX standard states that alias names must contain only characters in the POSIX portable character set.

alias in the tcsh shell

Without arguments, **alias** in the tcsh shell prints all aliases. With *name*, **alias** prints the alias for name. With *name* and *wordlist*, **alias** assigns *wordlist* as the alias of *name*. *wordlist* is command and filename substituted. *name* may not be *alias* or *unalias*.

See also “unalias in the tcsh shell” on page 653.

Options

–r Removes all tracked aliases.

–t Makes each *name* on the command line a tracked alias. Each tracked alias resolves to its full pathname; the shell thus avoids searching the **PATH** directories whenever you run the command. The shell assigns the full pathname of a tracked alias to the alias the first time you invoke it; the shell reassigns a pathname the first time you use the alias after changing the **PATH** variable.

When you enter the command:

```
set –h
```

each subsequent command you use in the shell automatically becomes a tracked alias. Running **alias** with the **–t** option, but without any specified names, displays all currently defined tracked aliases with appropriate quoting.

–x Marks each alias *name* on the command line for export. If you specify **–x** without any names on the command line, **alias** displays all exported aliases. Only exported aliases are passed to a shell that runs a shell script.

Several aliases are built into the shell. Some of them are:

```
alias autoload="typeset –fu"
alias functions="typeset –f"
alias hash="alias –t"
alias history="fc –l"
alias integer="typeset –i"
alias nohup="nohup "
alias r="fc –s"
alias stop="kill –STOP"
alias suspend="stop \$\$"
```

You can change or remove any of these aliases, and the changes will remain in effect for the current shell and any shell scripts or child shells invoked implicitly from the command. These aliases are reset to their default built-in values each time a new shell is invoked from the command line.

Example

The command:

```
alias ls="ls –C"
```

defines **ls** as an alias. From this point onward, when you issue an **ls** command, it produces multicolumn output by default.

alias in the tcsh shell examples

To alias the **!!** history command, use **\!-1** instead of **\!\!**. For example:

```
alias mf 'more \!-1$'
```

creates an alias for looking at the file named by the final argument of the previously entered command. Example output would be the following:

```
alias mf 'more \!-1$'
echo "We love tcsh." > file1
mf


We love tcsh.
"file1" (EOF)
```

where **mf** pulls the last argument of the previous command (**file1**), and then displays that file using the **more** command.

Localization

alias uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F, “Localization” on page 817 for more information.

Usage Notes

1. **alias** is a built-in shell command.
2. Because exported aliases are only available in the current shell environment and to the child processes of this environment, they are not available to any new shell environments that are started (via the **exec sh** command, for example). To make an alias available to all shell environments, define it as a nonexported alias in the **ENV** file, which is executed whenever a new shell is run.

Exit Values

0 Successful completion
1 Failure because an alias could not be set
2 Failure because of an incorrect command-line option

If you define **alias** to determine the values of a set of names, the exit value is the number of those names that are not currently defined as aliases.

Portability

POSIX.2 User Portability Extension, UNIX KornShell.

The **–t** and **–x** options are extensions to the POSIX standard.

Related Information

fc, **hash**, **nohup**, **set**, **sh**,**typeset**, **unalias**, **tcsh**

---

ar — Create or maintain library archives

Format

ar –d[–llv] *archive member...*
ar –m[–ablilsv] *[posname] archive member ...*
ar –p[–llsv] *archive member...*
ar –q[–clsv] [**–F** *format] archive member ...*
ar –r[abclilsuv] [**–F** *format] [posname*] *archive member ...*
ar –t[llsv] *archive[member...*]
ar –u[–abcliklsv] [**–F** *format*] [*posname*] *archive member ...*
ar –x[–CllsTv] *archive [member...*] *...*

Description

ar maintains archive libraries. The archive library is a collection of files, usually object files. Using **ar**, you can create a new library, add members to an existing library, delete members from a library, extract members from a library, and print a table of contents for a library.

A library member is an arbitrary file. Normally, these files are object files, suitable for use by a linkage editor.

If any members of a library are object files, **ar** creates and maintains an external symbol index for link-editing.

Member names in an archive are only the final component of any pathname. When creating a new library member (*member*) as given on the command line, **ar** uses the full pathname given. When storing the member name in the library, or comparing a member name, **ar** uses only the final component.

Options

The format shows the main functions of **ar**, which are defined as follows:

–d Deletes each named *member* from the archive and regenerates the symbol table.

–m Moves the named archive member in the archive. The new position is specified by **–a**, **–b**, **i**, or *posname*. If a location is not specified, the member is moved to the end of the archive.

–p Displays each *member* specified to the standard output (**stdout**). If you did not specify any members, **ar** displays all members.

–q Quickly appends the specified *file* to the archive. With this option, **ar** does not check to see if *file* is already a member of the archive.

–r Replaces or adds *file* to *archive*. If *archive* does not exist, **ar** creates it and prints a message. When **ar** replaces an existing member, the archive order is not changed. If *file* is not replacing a member, it is added to the end of the archive unless **–a**, **–b**, or **–i** is used. This option regenerates the symbol table.

–t Displays a table of contents that lists members, or every member if *member* is not specified. **ar** prints a message for each member it doesn't find. By default, **ar** prints the member name for all selected members. With the verbose (**–v**) option, **ar** prints more information for all selected members.

–x Extracts each specified *member* from the archive and copies it to a file. If *member* is specified as a full pathname, it is copied to that pathname. If no *member* is specified, all members are extracted. The archive remains unchanged.

The following options change the behavior of the main functions:

–a Places *file* in the archive after the member specified by *posname*. If no member is named, *file* is added to the end of the archive.

–b Places *file* in the archive before the member specified by *posname*. If no member is named, *file* is placed at the beginning of the archive.

–C Prevents **ar** from overwriting existing files with extracted files. This option is used only with extraction (**–x**).

–c Suppresses the message normally printed when **ar** creates a new archive file. You can use this only in conjunction with the **–r** and **–q** options.

–F *format*
Specifies the archive format to be used by a new archive. You can use this option only when creating a new archive with the **–r** and **–q** options.

–I Ignores the case of letters when searching the archive for specified member names. Normally, the case is significant.

–i Inserts *file* into the archive before the member specified by *posname*. If *posname* isn't specified, **ar** inserts *file* at the beginning of the archive. This option is the same as **–b**.

–l This option is ignored. It requests that temporary files generated by **ar** be put in the directory rather than in the default temporary file directory. It is provided for backward compatibility with other versions of **ar**

–s Regenerates the external symbol table regardless of whether the command modifies the archive.

–T When used with **–x**, allows extraction of members with names longer than the file system supports. Normally this is an error, and **ar** does not extract the file. Most file systems truncate the filename to the appropriate length.

–u Replaces the archive member only if the *member* file's modification time is more recent than the archive member time. **–u** implies **–r**, so it isn't necessary to specify **–r** also.

–v Gives verbose output. With **–d**, **–q**, **–r**, and **–x**, this option prints the command letter and the member name affected before performing each operation. With **–t**, **ar** prints more information about archive members using a format similar to **ls –l**. With **–p**, **ar** writes the name of the member to **stdout**, before displaying the contents of the file.

Operands

archive Specifies the pathname of the archive file.

member Specifies the pathname of the file that is to be acted upon (placed, deleted, searched for, and so on) in the archive library.

Examples

1. To add a member **fioacc.o** to the archive file **/u/turner/bin/cliserpgm.a**, specify:

   ```
   ar –rc /u/turner/bin/cliserpgm.a fioacc.o
   ```

2. To display the members of the archive file **/u/turner/bin/cliserpgm.a**, specify:

   ```
   ar –tv /u/turner/bin/cliserpgm.a
   ```

3. To delete the member **repgen.o** from the archive file **/u/turner/bin/cliserpgm.a** and regenerate the external symbol table for the archive, specify:

   ```
   ar –ds /u/turner/bin/cliserpgm.a repgen.o
   ```

Environment Variables

ar uses the following environment variable:

TMPDIR The pathname of the directory being used for temporary files. If it is not set, OS/390 UNIX uses **/tmp**.

Localization

ar uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_TIME**
- **NLSPATH**

See Appendix F, “Localization” on page 817 for more information.

Files

ar creates temporary files in the working directory and in the directory named by the **TEMPDIR** environment variable. These files are intermediate versions of the archive file being created or updated. Consequently, they normally are the same size as the archive file being manipulated.

Usage Note

Within the external symbol table, all symbols for a given member are kept together. Symbols of more recently added or modified members are located before symbols of older (not as recently modified) members in the archive. The modification time of an archive member determines its relative age.

Exit Values

0 Successful completion

1 Failure due to any of the following:

- Inability to create the extracted file
- An error writing to the extracted file
- The requested module not found on appending
- An error opening the module on appending
- An incorrect module on appending
- Inability to access the module on appending
- A module not found on table or extraction

2 Incorrect command-line arguments or options

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

For backward compatibility, you can omit the dash (**–**) preceding the options if the options appear only as the first argument after the command name.

The following options are XPG extensions to the POSIX standard: **–a**, **–b**, **–C**, **–i**, **–l**, **–m**, **–q**, **–s**, and **–T**.

The **–F** and the **–l** options are extensions to the POSIX and XPG standards.

Related Information

c89, **make**, **nm**

---

asa — Interpret ASA/FORTRAN carriage control

Format

asa [*file ...*]

Description

Historically, printouts created by programs use the first character of each line to control the spacing between that line and the previous one. For example, if the first character is a space, the rest of that line immediately follows the previous line; if it is a 1, that line should begin on a new page, and so on.

asa reads input in this format and writes it out in a normal text format, using newlines, formfeeds, and carriage returns to achieve the same effects as the carriage control characters.

If you specify files on the command line, **asa** reads input from these files; otherwise, it reads the standard input (**stdin**). **asa** writes output to the standard output (**stdout**).

It does not copy newline characters in the input to the output. Instead, it uses the first character of each line to determine how to print the rest of the line. **asa** interprets the first character as follows:

Space Outputs the rest of the line without change.

0 Outputs a newline character before printing *line*.

1 Outputs a formfeed (start a new page) sequence before printing *line*.

+ Outputs a carriage return sequence so that *line* is output over the previous *line*. If **+** starts the first line, it's treated as a space.

Localization

asa uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F, “Localization” on page 817 for more information.

Exit Values

0 Successful completion

1 Failure due to any of the following:

- Write error on **stdout**
- Inability to open the input file

2 Unknown command-line option

Portability

POSIX.2, X/Open Portability Guide, UNIX System V.

---

at — Run a command at a specified time

Format

at [**–m**] [**–f** *file*] [**–q** *queue*] **–t** *time*
at [**–m**] [**–f** *file*] [**–q** *queue*] *timespec*
at –r [**–q** *queue*] *at_job ...*
at –l [**–q** *queue*] [*at_job ...*]

tcsh shell:

@
@ *name* = expr
@ *name*[*index*] = expr
@ *name*+ + |– –
@ *name*[*index*+ + |– –

Description

at lets you set up a series of commands to be run later. It reads the commands from the standard input (**stdin**) or from a file specified with the **–f** option. When the commands run, they have the same environment variables, working directory, file creation mask, and so on that are set up when you run the **at** command; however, **at** does not usually preserve open file descriptors, traps, or priority inherited from the working environment.

Usually, you redirect the standard output (**stdout**) from these commands to files so you can read the files after the system runs the commands. **at** mails the standard output (**stdout**) and standard error output (**stderr**) to you if you do not redirect them.

The **at** command displays an *at-job* identifier when you submit commands, along with the time that the system is to run the commands.

at, **batch**, and **crontab** submit jobs to **cron**; the data in these jobs may contain doublebyte characters. When the jobs are run, the data in the jobs are interpreted in the locale that **cron** is using. Since it is strongly recommended that **cron** be started in the POSIX locale, doublebyte characters in the job may not be interpreted correctly. You can get around this by calling **setlocale()** in the job itself.

@ (at) in the tcsh shell

@ (at) in the tcsh shell prints the value of tcsh shell variables. **@** alone prints the values of all tcsh shell variables.

@ in the tcsh shell supports the following options:

name **= expr**
: Assigns the value of expr to name.

*name***[***index***] = expr**
: Assigns the value of expr to the *index*'th component of *name*. Both *name* and its *index*'th component must already exist.

  For both *name* **= expr** and *name*[*index*] **= expr**, **expr** may contain the operators *, +, etc. as in C. If **expr** contains <, >, &, or " then at least part of **expr** must be placed within (). The syntax of **expr** has nothing to do with that described under “Expressions” on page 578.

  expr must evaluate to a numeric expression. Therefore, use **set** instead of **@** to assign array variables.

*name***+ + |– –**
: Increments (++) or decrements (– –) *name*.

*name***[***index***]+ + |– –**
: Increments (++) or decrements (– –) *name*'s *inde*x'th component.

The space between **@** and name is required. The spaces between name and = and between = and **expr** are optional. Components of **expr** must be separated by spaces.

Options

–f *file*
Reads commands from *file* rather than from standard input (**stdin**).

–l Reports on standard output (**stdout**) all jobs you have scheduled and when the system is to run them if you do not specify *at_job*. If you specify *at_job*s, this option reports information on only those jobs.

–m Sends you mail after your job has finished running. If you did not redirect the **stdout** and **stderr**, **at** also mails these to you. If **stdout** or **stderr** is non-null, **at** mails this output to you even if you do not specify **–m**.

–q *queue*
Specifies the queue your **at** job is to be recorded in or removed from. *queue* can be any singlebyte character except a space, a tab, a null character, or a number sign (#). By default, **at** stores all its jobs in a queue called `a`, and **batch** stores all its jobs in a queue called `b`. If used with this option, **–l** reports information only on **at** jobs in *queue*.

–r *at_job*
Removes previously scheduled **at** jobs. The *at_job* arguments must be the identifiers assigned to the jobs when you set them up with **at**.

–t *time*
Specifies the time for the system to run the job. You specify *time* in the same format as the time argument for **touch**.

When you do not use the **–t** option, you can use a *timespec* argument to specify the time. A *timespec* argument consists of three parts: a time, a date, and an increment (in that order). You must always specify the time, but you can omit the date, the increment, or both. Following are possible time formats:

Format	Meaning
hhmm	*hh* hours, *mm* minutes, 24-hour clock
hh:mm	*hh* hours, *mm* minutes, 24-hour clock
h:mm	*h* hours, *mm* minutes, 24-hour clock
h:m	*h* hours, *m* minutes, 24-hour clock
hh:mm zone	*zone* is time zone
hh:mm`am`	Morning, 12-hour clock
hh:mm`am` *zone*	Morning, 12-hour clock in given time zone
hh:mm`pm`	Afternoon, 12-hour clock
hh:mm`pm` *zone*	Afternoon, 12-hour clock in given time zone
`noon`	Noon
`midnight`	Midnight
`next`	Current time, next day that meets date and increment
`now`	Current time today

All minute specifications are optional. For example, to specify an **at** job to run at 1:00 p.m., you can enter

```
at 1pm
```

Currently, the OS/390 shell only supports the time zones GMT, CUT, UTC, and ZULU, all of which stand for Coordinated Universal Time (often called Greenwich Mean Time). If you do not specify a zone, **at** interprets times with respect to the **TZ** variable.

Appendix I, “Setting the Local Time Zone with the TZ Environment Variable” on page 837 explains how to set the local time zone with the **TZ** environment variable.

Possible date formats are shown in the following list:

Format	Meaning
month day	*month* is the full name, or the three-letter abbreviation (as in `January` or `Jan`)
month day, year	*day* and *year* given as appropriate numbers
weekday	*weekday* is the full name or the three-letter abbreviation (as in `Monday` or `Mon`)
`today`	Current day
`tomorrow`	Next day

The increment is added to the time and date you specify with the preceding parts of *timespec*. It has the format + *`n units`* where *n* is a number and *units* is one of the following:

```
minute     minutes    hour     hours
day        days       week     weeks
month      months     year     years
```

Here are some sample time specifications:

```
0655
1855
18:55
6:55pm
6:55 pm Jan 10
now + 3 hours
noon tomorrow
midnight Friday
```

Environment Variables

at uses the following environment variables:

SHELL Contains the name of the shell used to invoke the **at** job.

TZ Specifies the default time zone for all times given on the command line. If you include a time zone as part of *time* or *timespec*, it overrides the value of **TZ**.

Appendix I, “Setting the Local Time Zone with the TZ Environment Variable” on page 837 explains how to set the local time zone with the **TZ** environment variable.

Usage Note

at jobs that contain a line consisting of just the string `"!!!ATEOF!!!"` fail with unexpected results.

Localization

at uses the following localization variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_TIME**
- **NLSPATH**

The keywords *midnight*, *noon*, *today*, and *tomorrow* are valid only in the POSIX locale. See Appendix F, “Localization” on page 817 for more information.

Exit Values

`0` Successful completion
`>0` Returned if the command fails for any reason

If an error occurs, **at** does not schedule, remove, or list the job.

Portability

POSIX.2 User Portability Extension, UNIX systems.

Related Information

batch, **bg**, **cron**, **crontab**, **touch**, **tcsh**

Appendix I, “Setting the Local Time Zone with the TZ Environment Variable” on page 837 also explains how to set the local time zone with the **TZ** environment variable.

---

autoload — Indicate function name not defined

Format

autoload *name ...*

Description

autoload is an alias for **typeset –fu**. Like **typeset –fu**, **autoload** indicates that the functions named in the command line are not yet defined.

See “typeset — Assign attributes and values to variables” on page 646 and “Command Execution” on page 493 for more information.

Related Information

typeset, **functions**, **sh**

automount — Configure the automount facility

Format

automount [**–s**] [*Master filename*]

Description

automount is used to configure the automount facility. This facility can automatically mount file systems at the time they are accessed, and also unmount them later. (For information on setting up the automount facility, refer to *OS/390 UNIX System Services Planning*.)

When run with no arguments, **automount** reads the **/etc/auto.master** file to determine all directories that are to be configured for automounting and the filenames that contain their configuration specifications.

If you run **automount** with the [*master filename*] argument, that filename is used instead of **/etc/auto.master**.

automount should be run from the **/etc/rc** script with no arguments. This processes the installation's default **automount** configuration file.

automount requires superuser authority.

Option

–s Checks the syntax of the configuration file. No automount is performed.

Examples

1. The following example shows how automatic unmount can be avoided for a directory:

   ```
   name       wjs
   duration   nolimit
   ```

   Keywords that are not specified on a specific entry are inherited from the generic entry, if present. If the generic entry is not present, or if keys are not specified, the defaults are used. If the *filesystem* key cannot be resolved, the entry is considered invalid.

2. The following is an example of a **/etc/auto.master** file that is used to specify **/u** as automount-managed and the specifications for that directory in **/etc/u.map**:

   ```
   /u       /etc/u.map
   ```

Files

automount uses these files:

/etc/auto.master
: Specifies a list of directories to be configured, along with their **MapName** files.

  Each line in this file contains two pathnames separated by at least one space: the directory name to be managed and the pathname of the **MapName** file. Both of these pathnames should be absolute.

The pathname of the managed directory is used as a filesystem name prefixed with ***AMD**. This restricts the length of the pathname of a managed directory to 40 characters. If pathnames need to be longer, you can use symbolic links to resolve all or part of the pathname.

MapName

The **MapName** file contains the mapping between a subdirectory of a directory managed by **automount** and the mount parameters.

The file is organized as a set of specifications. Each specification contains one or more lines. Each line is of the form *keyword argument*. Each specification must begin with the keyword *name*.

Blank lines and lines beginning with the characters **/*** are considered comments and are ignored.

A generic entry can be specified as the first specification by using the name of *. The generic specification provides defaults for subsequent specific specifications. When the automounter tries to resolve a lookup request, it attempts to find a specific entry. If a specific entry does not exist for the name being looked up, it attempts to use the generic entry.

The following is an example of a generic entry:

```
name       *
type       HFS
filesystem OMVS.HFS.USER.<uc_name>
mode       rdwr
duration   30
delay      10
```

Three special symbols are supported to provide name substitution:

<**asis_name**> used to represent the name exactly, as is.
<**uc_name**> used to represent the name in uppercase characters.
<**sysname**> used to substitute the system name.

You can use these when specifying a file system name or file system parameter that has a specific form with the name inserted as a qualifier.

Following is a list of supported keywords. You can enter keywords using mixed case letters. Some arguments require mixed case.

name The name of the directory to be mounted. This key is required and must be the first key specified for the entry. If the first entry specifies *name **, it is treated as the generic entry for the automount-managed directory.

type The file system type name. The default is HFS.

filesystem
The name of the file system to mount. This argument is case-sensitive. For the HFS file system, this argument must be specified in uppercase.

mode The mount mode for the file system (rdwr or read). The default is rdwr.

parm The file system-specific parameter. This argument is case-sensitive.

duration The minimum amount of time in minutes to leave the file system mounted. The default is *nolimit*.

delay The minimum amount of time in minutes to leave the file system mounted after the duration has expired and the file system is no longer in use. The default is `0`.

setuid Specifies whether the setuid/setgid mode bits are to be respected for executables run from this file system. You can specify these values:

Yes The setuid/setgid modes or respected. This is the default.

No The setuid/setgid modes are ignored.

security Specifies security checking which should be done for files in the filesystem. You can specify these values:

Yes Normal security checking will be done. This is the default.

No Specifies that security checks will not be enforced for files in this filesystem. Any user may access or change any file or directory in any way.

Security auditing will still be performed if the installation is auditing successes.

The SETUID, SETGID, APF, and Program Control mode bits may be turned on in files from this filesystem, but will not be honored while it is mounted with NOSECURITY. In other words, setting security to **No** also sets setuid to **No**.

Note: The installation should normally take the default (**Yes**).

For more information on mounting with no security and on the MOUNT statement in BPXPRMxx, see *OS/390 UNIX System Services Planning*. Security keywords on the TSO MOUNT command are also discussed in "MOUNT — Logically mount a file system" on page 743.

---

awk — Process programs written in the awk language

Format

awk [**–F** *ere*] [**–f** *prog*] [**–v** *var*=*value* ...] [*program*] [*var*=*value* ...] [*file* ...]

Description

awk is a file-processing language that is well suited to data manipulation and retrieval of information from text files. If you are unfamiliar with the language, you may find it helpful to read the **awk** information in *OS/390 UNIX System Services User's Guide* before reading the following material.

An **awk** program consists of any number of user-defined functions and *rules* of the form:

pattern {*action*}

There are two ways to specify the **awk** program:

- Directly on the command line. In this case, *program* is a single command-line argument, usually enclosed in single quotes (') to prevent the shell from attempting to expand it.
- By using the **–f** *prog* option.

You can specify *program* directly on the command line only if you do not use any **–f** *prog* arguments.

Options

awk recognizes the following options:

–F *ere*
Is an extended regular expression to use as the field separator.

–f *prog*
Runs the **awk** program contained in the file *prog*. When more than one **–f** option appears on the command line, the resulting program is a concatenation of all programs you specify.

–v *var=value*
Assigns *value* to *var* before running the program.

Files that you specify on the command line with the *file* argument provide the input data for **awk** to manipulate. If you specify no files or you specify a dash (**–**) as a file, **awk** reads data from standard input (**stdin**).

You can initialize variables on the command line using:

var=value

You can intersperse such initializations with the names of input files on the command line. **awk** processes initializations and input files in the order they appear on the command line. For example, the command:

```
awk -f progfile a=1 f1 f2 a=2 f3
```

sets *a* to 1 before reading input from *f1* and sets *a* to 2 before reading input from *f3*.

Variable initializations that appear before the first *file* on the command line are performed immediately after the **BEGIN** action. Initializations appearing after the last *file* are performed immediately before the **END** action. For more information on **BEGIN** and **END,** see “Patterns” on page 35.

The **–v** option lets you assign a value to a variable before the **awk** program begins execution (that is, before the **BEGIN** action). For example, in:

```
awk -v v1=10 -f prog datafile
```

awk assigns the variable *v1* its value before the **BEGIN** action of the program (but after default assignments made to such built-in variables as **FS** and **OFMT**; these built-in variables have special meaning to **awk**, as described later).

awk divides input into *records*. By default, newline characters separate records; however, you can specify a different record separator if you want.

One at a time, and in order, **awk** compares each input record with the pattern of every rule in the program. When a pattern matches, **awk** performs the action part of the rule on that input record. Patterns and actions often refer to separate *fields* within a record. By default, white space (usually blanks, newlines, or horizontal tab characters) separates fields; however, you can specify a different field separator string using the **–F** *ere* option).

You can omit the *pattern* or *action* part of an **awk** rule (but not both). If you omit *pattern*, **awk** performs the *action* on every input record (that is, every record matches). If you omit *action*, **awk** writes every record matching the *pattern* to the standard output (**stdout**).

awk considers everything after a `#` in a program line to be a comment. For example:

```
# This is a comment
```

To continue program lines on the next line, add a backslash (\) to the end of the line. Statement lines ending with a comma (`,`), double or-bars (`||`), or double ampersands (`&&`) continue automatically on the next line.

Variables and Expressions

There are three types of variables in **awk**: *identifiers*, *fields*, and *array elements*.

An identifier is a sequence of letters, digits, and underscores beginning with a letter or an underscore. These characters must be from the POSIX portable character set. (Data can come from other character sets.)

For a description of fields, see “Input.” on page 31.

Arrays are associative collections of values called the *elements* of the array. Constructs of the form:

identifier[*subscript*]

where *subscript* has the form *expr* or *expr,expr,...,* refer to array elements. Each such *expr* can have any string value. For multiple *expr* subscripts, **awk** concatenates the string values of all *expr* arguments with a separate character **SUBSEP** between each. The initial value of **SUBSEP** is set to `\042` (code page 01047 field separator).

We sometimes refer to fields and identifiers as *scalar variables* to distinguish them from arrays.

You do not declare **awk** variables, and you do not need to initialize them. The value of an uninitialized variable is the empty string in a string context and the number `0` in a numeric context.

Expressions consist of constants, variables, functions, regular expressions, and *subscript-in-array* conditions combined with operators. (Subscript-in-array conditions are described in Subscript in Array) Each variable and expression has a string value and a corresponding numeric value; **awk** uses the value appropriate to the context.

When converting a numeric value to its corresponding string value, **awk** performs the equivalent of a call to the **sprintf()** function where the one and only *expr* argument is the numeric value and the *fmt* argument is either `%d` (if the numeric value is an integer) or the value of the variable **CONVFMT** (if the numeric value is not an integer). The default value of **CONVFMT** is `%.6g`. If you use a string in a numeric context, and **awk** cannot interpret the contents of the string as a number, it treats the value of the string as zero.

Numeric constants are sequences of decimal digits.

String constants are quoted, as in `"a literal string"`. Literal strings can contain the following escape sequences:

Escape Character Sequence	
`\a`	Audible bell
`\b`	Backspace
`\f`	Form feed
`\n`	Newline
`\r`	Carriage return
`\t`	Horizontal tab
`\v`	Vertical tab
`\`*ooo*	Octal value *ooo*
`\x`*dd*	Hexadecimal value *dd*
`\/`	Slash
`\"`	Quote
`\`*c*	Any other character *c*

awk supports full regular expressions. (See Appendix C, “Regular Expressions (regexp)” on page 797 for more information.) When **awk** reads a program, it compiles characters enclosed in slash characters (/) as regular expressions. In addition, when literal strings and variables appear on the right side of a ~ or ~ operator, or as certain arguments to built-in matching and substitution functions, **awk** interprets them as dynamic regular expressions.

Note: When you use literal strings as regular expressions, you need extra backslashes to escape regular expression metacharacters, because the backslash is also the literal string escape character. For example the regular expression:

```
/e .g\7./
```

when written as a string is:

```
"e  .g  .."
```

Subscript in Array

awk defines the *subscript-in-array* condition as:

```
index in array
```

where *index* looks like *expr* or *(expr,...,expr)*. This condition evaluates to `1` if the string value of *index* is a subscript of *array*, and to `0` otherwise. This is a way to determine if an array element exists. When the element does not exist, the subscript-in-array condition does not create it.

Symbol Table

You can access the symbol table through the built-in array **SYMTAB.** **SYMTAB[***expr***]** is equivalent to the variable named by the evaluation of *expr*.

For example, **SYMTAB[***"var"***]** is a synonym for the variable *var*.

Environment

An **awk** program can determine its initial environment by examining the **ENVIRON** array. If the environment consists of entries of the form *name=value,* then **ENVIRON**[*name*] has string value "*value*". For example, the following program is equivalent to the default output of **env**:

```
BEGIN {
        for (i in ENVIRON)
                printf("%s=%s\n", i, ENVIRON[i])
        exit
     }
```

Operators

awk follows the usual precedence order of arithmetic operations, unless overridden with parentheses; a table giving the order of operations appears later in this section.

The unary operators are +, –, ++, and – – , where you can use the ++ and – – operators as either postfix or prefix operators, as in C. The binary arithmetic operators are +, –, *, /, %, and ^.

The conditional operator

expr ? *expr1* : *expr2*

evaluates to *expr1* if the value of *expr* is nonzero, and to *expr2* otherwise.

If two expressions are not separated by an operator, **awk** concatenates their string values.

The tilde operator ~ yields **1** (true) if the regular expression on the right side matches the string on the left side. The operator !~ yields 1 when the right side has no match on the left. To illustrate:

```
$2 ~ /[0-9]/
```

selects any line where the second field contains at least one digit. **awk** interprets any string or variable on the right side of ~ or !~ as a dynamic regular expression.

The relational operators are <, <=, >, >=, ==, and !=. When both operands in a comparison are numeric, **awk** compares their values numerically; otherwise, it compares them as strings. An operand is numeric if it is an integer or floating-point number, if it is a field or **ARGV** element that looks like a number, or if it is a variable created by a command-line assignment that looks like a number.

The Boolean operators are || (or), && (and), and ! (not). **awk** uses *short-circuit evaluation* when evaluating expressions. With an && expression, if the first operator is false, the entire expression is false and it is not necessary to evaluate the second operator. With an || expression, a similar situation exists if the first operator is true.

You can assign values to a variable with:

var = expr

If *op* is a binary arithmetic operator, *var op= expr* is equivalent to *var = var op expr*, except that *var* is evaluated only once.

See Figure 2 for the precedence rules of the operators.

Figure 2. The Order of Operations for awk

Operators	Order of Operations
(A)	Grouping
$*i* V[a]	Field, array element
V++ V-- ++V --V	Increment, decrement
A^B	Exponentiation
+A -A !A	Unary plus, unary minus, logical NOT
A*B A/B A%B	Multiplication, division, remainder
A+B A-B	Addition, subtraction
A B	String concatenation
A<B A>B A<=B A>=B A!=B A= =B	Comparisons
A ~ B A! ~ B	Regular expression matching
A in V	Array membership
A && B	Logical AND
A || B	Logical OR
A ? B : C	Conditional expression
V=B V+=B V-=B V*=B V/=B V%=B V^=B	Assignment

Notes:

1. A, B, C are any expression.
2. i is any expression yielding an integer.
3. V is any variable.

Command-Line Arguments

awk sets the built-in variable **ARGC** to the number of command-line arguments. The built-in array **ARGV** has elements subscripted with digits from zero to **ARGC**−1, giving command-line arguments in the order they appeared on the command line.

The **ARGC** count and the **ARGV** vector do not include command-line options (beginning with -) or the program file (following **–f**). They do include the name of the command itself, initialization statements of the form *var=value*, and the names of input data files.

awk actually creates **ARGC** and **ARGV** before doing anything else. It then “walks through” **ARGV,** processing the arguments. If an element of **ARGV** is an empty

string, **awk** skips it. If it contains an equals sign (=), **awk** interprets it as a variable assignment. If it is a minus sign (–), **awk** immediately reads input from **stdin** until it encounters the end of the file. Otherwise, **awk** treats the argument as a filename and reads input from that file until it reaches the end of the file.

Note: **awk** runs the program by "walking through" **ARGV** in this way; thus, if the program changes **ARGV, awk** can read different files and make different assignments.

Input.

awk divides input into records. A *record separator character* separates each record from the next. The value of the built-in variable **RS** gives the current record separator character; by default, it begins as the newline (`\n`). If you assign a different character to **RS, awk** uses that as the record separator character from that point on.

awk divides records into fields. A *field separator string*, given by the value of the built-in variable **FS**, separates each field from the next. You can set a specific separator string by assigning a value to **FS**, or by specifying the **–F** *ere* option on the command line. You can assign a regular expression to **FS**. For example:

```
FS = "[,:$]"
```

says that commas, colons, or dollar signs can separate fields. As a special case, assigning **FS** a string containing only a blank character sets the field separator to white space. In this case, **awk** considers any sequence of contiguous space or tab characters a single field separator. This is the default for **FS**. However, if you assign **FS** a string containing any other character, that character designates the start of a new field. For example, if we set `FS=\t` (the tab character),

```
texta \t textb \t  \t  \t textc
```

contains five fields, two of which contain only blanks. With the default setting, this record only contains three fields, since **awk** considers the sequence of multiple blanks and tabs a single separator.

The following list of built-in variables provides various pieces of information about input:

NF	Number of fields in the current record
NR	Number of records read so far
FILENAME	Name of file containing current record
FNR	Number of records read from current file

Field specifiers have the form $*n*, where *n* runs from `1` through **NF**. Such a field specifier refers to the *n*th field of the current input record. **$0** (zero) refers to the entire current input record.

The **getline** function can read a value for a variable or **$0** from the current input, from a file, or from a pipe. The result of **getline** is an integer indicating whether the read operation was successful. A value of `1` indicates success; `0` indicates that the end of the file was encountered; and `–1` indicates that an error occurred. Possible forms for **getline** are:

getline

Reads next input record into `$0` and splits the record into fields. **NF, NR,** and **FNR** are set appropriately.

getline *var*
Reads the next input record into the variable *var*. **awk** does not split the record into fields (which means that the current *$n* values do not change), but sets **NR** and **FNR** appropriately.

getline <*expr*
Interprets the string value of *expr* to be a filename. **awk** reads the next record from that file into **$0**, splits it into fields, and sets **NF** appropriately. If the file is not open, **awk** opens it. The file remains open until you close it with a **close** function.

getline *var* <*expr*
Interprets the string value of *expr* to be a filename, and reads the next record from that file into the variable *var*, but does not split it into fields.

expr | **getline**
Interprets the string value of *expr* as a command line to be run. **awk** pipes output from this command into **getline**, and reads it into *$0*, splits it into fields, and sets **NF** appropriately. See System Function for additional details.

expr | **getline** *var*
Runs the string value of *expr* as a command and pipes the output of the command into **getline**. The result is similar to **getline** *var* <*expr*.

You can have only a limited number of files and pipes open at one time. You can close files and pipes during execution using the **close**(*expr*) function. The *expr* argument must be one that came before | or after < in **getline**, or after > or >> in **print** or **printf**.

If the function successfully closes the pipe, it returns zero. By closing files and pipes that you no longer need, you can use any number of files and pipes in the course of running an **awk** program.

Built-in Arithmetic Functions

atan2(*expr1*, *expr2*)
Returns the arctangent of *expr1*/*expr2* in the range of $-\Pi$ through Π.

exp(*expr*), **log**(*expr*), **sqrt**(*expr*)
Returns the exponential, natural logarithm, and square root of the numeric value of *expr*. If you omit (*expr*), these functions use *$0* instead.

int(*expr*)
Returns the integer part of the numeric value of *expr*. If you omit (*expr*), the function returns the integer part of *$0*.

rand()
Returns a random floating-point number in the range `0` through `1`.

sin(*expr*), **cos**(*expr*)
Returns the sine and cosine of the numeric value of *expr* (interpreted as an angle in radians).

srand(*expr*)
Sets the seed of the **rand** function to the integer value of *expr*. If you omit (*expr*), **awk** uses the time of day as a default seed.

Built-in String Functions

len = **length (***expr***)**
Returns the number of characters in the string value of *expr*. If you omit (*expr*), the function uses *$0* instead. The parentheses around *expr* are optional.

n = **split(***string***,** *array***,** *regexp***)**
Splits the *string* into fields. *regexp* is a regular expression giving the field separator string for the purposes of this operation. This function assigns the separate fields, in order, to the elements of *array*; subscripts for `array` begin at `1`. **awk** discards all other elements of *array*. **split** returns the number of fields into which it divided *string* (which is also the maximum subscript for *array*).

regexp divides the record in the same way that the **FS** field separator string does. If you omit *regexp* in the call to **split**, it uses the current value of **FS**.

str = **substr(***string***,** *offset***,** *len***)**
Returns the substring of *string* that begins in position *offset* and is at most *len* characters long. The first character of the string has an *offset* of `1`. If you omit *len*, **substr** returns the rest of *string*.

pos = **index(***string***,** *str***)**
Returns the position of the first occurrence of *str* in *string*. The count is in characters. If **index** does not find *str* in *string*, it returns `0`.

pos = **match(***string***,** *regexp***)**
Searches *string* for the first substring matching the regular expression *regexp*, and returns an integer giving the position of this substring counting from `1`. If it finds no such substring, **match** returns zero. This function also sets the built-in variable **RSTART** to *pos* and the built-in variable **RLENGTH** to the length of the matched string. If it does not find a match, **match** sets **RESTART** to `0`, and **RLENGTH** to –1. You can enclose *regexp* in slashes or specify it as a string.

n = **sub(***regexp***,** *repl***,** *string***)**
Searches *string* for the first substring matching the regular expression *regexp*, and replaces the substring with the string *repl*. **awk** replaces any ampersand (&) in *repl* with the substring of *string* which matches *regexp*. You can suppress this special behavior by preceding the ampersand with a backslash. If you omit *string*, **sub** uses the current record instead. **sub** returns the number of substrings replaced (which is `1` if it found a match, and `0` otherwise).

n = **gsub(***regexp***,** *repl***,** *string***)**
Works the same way as **sub**, except that **gsub** replaces all matching substrings (global substitution). The return value is the number of substitutions performed.

str = **sprintf(***fmt***,** *expr***,** *expr...***)**
Formats the expression list *expr, expr, ...* using specifications from the string *fmt*, and then returns the formatted string. The *fmt* string consists of conversion specifications that convert and add the next *expr* to the string, and ordinary characters that **sprintf** simply adds to the string. These conversion specifications are similar to those used by the ANSI (see *OS/390 C/C++ IBM Open Class Library Reference*).

Conversion specifications have the form

`%[-][0][x][.y]c`

where

- `-` Left-justifies the field; default is right justification.
- `0` (Leading zero) prints numbers with leading zero.
- *x* Is the minimum field width.
- *y* Is the precision.
- *c* Is the conversion character.

In a string, the precision is the maximum number of characters to be printed from the string; in a number, the precision is the number of digits to be printed to the right of the decimal point in a floating-point value. If *x* or *y* is * (asterisk), the minimum field width or precision is the value of the next *expr* in the call to **sprintf**.

The conversion character *c* is one of following:

- **d** Decimal integer
- **i** Decimal integer
- **o** Unsigned octal integer
- **x,X** Unsigned hexadecimal integer
- **u** Unsigned decimal integer
- **f,F** Floating point
- **e,E** Floating point (scientific notation)
- **g,G** The shorter of **e** and **f** (suppresses nonsignificant zeros)
- **c** Single character of an integer value; first character of string
- **s** String

The lowercase **x** specifies alphabetic hex digits in lowercase, whereas the uppercase **X** specifies alphabetic hex digits in uppercase. The other uppercase-lowercase pairs work similarly.

n = **ord(***expr***)**
: Returns the integer value of first character in the string value of *expr*. This is useful in conjunction with *%c* in **sprintf**.

str = **tolower(***expr***)**
: Converts all letters in the string value of *expr* into lowercase, and returns the result. If you omit *expr*, **tolower** uses *$0* instead. This function uses the value of the locale or the **LC_CTYPE** environment variable.

str = **toupper(***expr***)**
: Converts all letters in the string value of *expr* into uppercase, and returns the result. If you omit *expr*, **toupper** uses *$0* instead. This function uses the value of the locale or the **LC_CTYPE** environment variable.

System Function

status = **system(***expr***)**
: Runs the string value of *expr* as a command. For example, **system***("tail " $1)* calls the **tail** command, using the string value of *$1* as the file that **tail** examines. The standard command interpreter runs the command, as discussed in the Portability Section, and the exit status returned depends on that command interpreter.

User-Defined Functions

You can define your own functions using the form:

```
function name(parameter-list) {
        statements
}
```

A function definition can appear in the place of a *pattern* {*action*} rule. The *parameter-list* argument contains any number of normal (scalar) and array variables separated by commas. When you call a function, **awk** passes scalar arguments by value, and array arguments by reference. The names specified in *parameter-list* are local to the function; all other names used in the function are global. You can define local variables by adding them to the end of the parameter list as long as no call to the function uses these extra parameters.

A function returns to its caller either when it runs the final statement in the function, or when it reaches an explicit **return** statement. The return value, if any, is specified in the **return** statement (see "Actions").

Patterns

A *pattern* is a regular expression, a special pattern, a pattern range, or any arithmetic expression.

BEGIN is a special pattern used to label actions that **awk** performs before reading any input records. **END** is a special pattern used to label actions that **awk** performs after reading all input records.

You can give a pattern range as:

```
pattern1,pattern2
```

This matches all lines from one that matches *pattern1* to one that matches *pattern2*, inclusive.

If you omit a pattern, or if the numeric value of the pattern is nonzero (true), **awk** runs the resulting action for the line.

Actions

An *action* is a series of statements ended by semicolons, newlines, or closing braces. A *condition* is any expression; **awk** considers a nonzero value true, and a zero value false. A *statement* is one of the following or any series of statements enclosed in braces:

```
# expression statement, e.g. assignment
expression

# if statement
if (condition)
        statement
[else
        statement]

# while loop
while (condition)
        statement
```

```
# do-while loop
do
        statement
while (condition)

# for loop
for (expression1; condition; expression2)
        statement
```

The **for** statement is equivalent to:

```
expression1
while (condition) {
        statement
        expression2
}
```

The **for** statement can also have the form:

```
for (i in array)
        statement
```

awk runs the statement (specified with the *statement* argument) once for each element in *array*; on each repetition, the variable *i* contains the name of a subscript of *array*, running through all the subscripts in an *arbitrary* order. If *array* is multidimensional (has multiple subscripts), *i* is expressed as a single string with the **SUBSEP** character separating the subscripts.

The statement **break** exits a **for** or a **while** loop immediately. **continue** stops the current iteration of a **for** or **while** loop and begins the next iteration (if there is one). **next** ends any processing for the current input record and immediately starts processing the next input record. Processing for the next record begins with the first appropriate rule. **exit**[(*expr*)] immediately goes to the **END** action if it exists; if there is no **END** action, or if **awk** is already running the **END** action, the **awk** program ends. **awk** sets the exit status of the program to the numeric value of *expr*. If you omit (*expr*), the exit status is `0`. **return** [*expr*] returns from the execution of a function.

If you specify an *expr*, the function returns the value of the expression as its result; otherwise, the function result is undefined. **delete** *array*[*i*] deletes element *i* from the given *array*. **print** *expr*, *expr*, *...* is described in Output. **printf** *fmt*, *expr*, *expr*, *...* is also described in Output.

Output

The **print** statement prints its arguments with only simple formatting. If it has no arguments, it prints the entire current input record. **awk** adds the output record separator **ORS** to the end of the output that each **print** statement produces; when commas separate arguments in the **print** statement, the output field separator **OFS** separates the corresponding output values. **ORS** and **OFS** are built-in variables, whose values you can change by assigning them strings. The default output record separator is a newline, and the default output field separator is a space.

The variable **OFMT** gives the format of floating-point numbers output by **print**. By default, the value is `%.6g`; you can change this by assigning **OFMT** a different string value. **OFMT** applies only to floating-point numbers (ones with fractional parts).

The **printf** statement formats its arguments using the *fmt* argument. Formatting is the same as for the built-in function **sprintf**. Unlike **print**, **printf** does not add output separators automatically. This gives the program more precise control of the output.

The **print** and **printf** statements write to **stdout**. You can redirect output to a file or pipe.

If you add >*expr* to a **print** or **printf** statement, **awk** treats the string value of *expr* as a filename, and writes output to that file. Similarly, if you add >>*expr*, **awk** sends output to the current contents of the file. The distinction between > and >> is important only for the first **print** to the file *expr.* Subsequent outputs to an already open file append to what is there already.

You cannot use such ambiguous statements as:

```
print a > b c
```

Use parentheses to resolve the ambiguity.

If you add |*expr* to a **print** or **printf** statement, **awk** treats the string value of *expr* as an executable command and runs it with the output from the statement piped as input into the command.

As mentioned earlier, you can have only a limited number of files and pipes open at any time. To avoid going over the limit, use the **close** function to close files and pipes when you no longer need them.

print and **printf** are also available as functions with the same calling sequence, but no redirection.

Examples

1. The following example:

   ```
   awk '{print NR ":" $0}' input1
   ```

   outputs the contents of the file **input1** with line numbers prepended to each line.

2. The following is an example using *var=value* on the command line:

   ```
   awk '{print NR SEP $0}' SEP=":" input1
   ```

   awk can also read the program script from a file as in the command line:

   ```
   awk –f addline.awk input1
   ```

   which produces the same output when the file **addline.awk** contains:

   ```
   {print NR ":" $0}
   ```

3. The following program appends all input lines starting with `January` to the file **jan** (which may or may not exist already), and all lines starting with `February` or `March` to the file **febmar**:

   ```
   /^January/ {print >> "jan"}
   /^February|^March/ {print >> "febmar"}
   ```

4. This program prints the total and average for the last column of each input line:

   ```
           {s += $NF}
   END     {print "sum is", s, "average is", s/NR}
   ```

5. The next program interchanges the first and second fields of input lines:

```
{
        tmp = $1
        $1 = $2
        $2 = tmp
        print
}
```

6. The following inserts line numbers so that output lines are left-aligned:

```
{printf "%-6d: %s\n", NR, $0}
```

7. The following prints input records in reverse order (assuming sufficient memory):

```
{
        a[NR] = $0 # index using record number
}
END {
        for (i = NR; i>0; --i)
                print a[i]
}
```

8. The following program determines the number of lines starting with the same first field:

```
{
        ++a[$1] # array indexed using the first field
}
END {   # note output will be in undefined order
        for (i in a)
                print a[i], "lines start with", i
}
```

   You can use the following program to determine the number of lines in each input file:

```
{
        ++a[FILENAME]
}
END {
        for (file in a)
                if (a[file] == 1)
                        print file, "has 1 line"
                else
                        print file, "has", a[file], "lines"
}
```

9. The following program illustrates how you can use a two-dimensional array in **awk**. Assume the first field of each input record contains a product number, the second field contains a month number, and the third field contains a quantity (bought, sold, or whatever). The program generates a table of products versus month.

```
BEGIN   {NUMPROD = 5}
{
        array[$1,$2] += $3
}
END     {
        print "\t Jan\t Feb\tMarch\tApril\t May\t" \
            "June\tJuly\t Aug\tSept\t Oct\t Nov\t Dec"
        for (prod = 1; prod <= NUMPROD; prod++) {
                printf "%-7s", "prod#" prod
                for (month = 1; month <= 12; month++){
                        printf "\t%5d", array[prod,month]
                }
                printf "\n"
        }
}
```

10. As the following program reads in each line of input, it reports whether the line matches a predetermined value:

```
function randint() {
        return (int((rand()+1)*10))
}
BEGIN   {
        prize[randint(),randint()] = "$100";
        prize[randint(),randint()] = "$10";
        prize[1,1] = "the booby prize"
        }
{
        if (($1,$2) in prize)
                printf "You have won %s!\n", prize[$1,$2]
}
```

11. The following example prints lines, the first and last fields of which are the same, reversing the order of the fields:

```
$1==$NF {
        for (i = NF; i > 0; --i)
                printf "%s", $i (i>1 ? OFS : ORS)
}
```

12. The following program prints the input files from the command line. The **infiles** function first empties the passed array, and then fills the array. The extra parameter *i* of **infiles** is a local variable.

```
function infiles(f,i) {
        for (i in f)
               delete f[i]
        for (i = 1; i < ARGC; i++)
                if (index(ARGV[i],"=") == 0)
                        f[i] = ARGV[i]
}
BEGIN   {
        infiles(a)
        for (i in a)
                print a[i]
        exit
     }
```

13. Here is the standard recursive factorial function:

```
function fact(num) {
        if (num <= 1)
                return 1
        else
                return num * fact(num - 1)
}
{ print $0 " factorial is " fact($0) }
```

14. The following program illustrates the use of **getline** with a pipe. Here, **getline** sets the current record from the output of the **wc** command. The program prints the number of words in each input file.

```
function words(file,   string) {
        string = "wc " fn
        string | getline
        close(string)
        return ($2)
}
BEGIN   {
        for (i=1; i<ARGC; i++) {
                fn = ARGV[i]
                printf "There are %d words in %s.",
                    words(fn), fn
        }
}
```

Environment Variables

awk uses the following environment variable:

PATH Contains a list of directories that **awk** searches when looking for commands run by **system**(*expr*), or input and output pipes.

Any other environment variable can be accessed by the **awk** program itself.

Localization

awk uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_NUMERIC**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F, “Localization” on page 817 for more information.

Exit Values

0 Successful completion

1 Any of the following errors:

- Parser internal stack overflow
- Syntax error
- Function redefined
- Internal execution tree error

- Insufficient memory for string storage
- Unbalanced parenthesis or brace
- Missing script file
- Missing field separator
- Missing variable assignment
- Unknown option
- Incorrect character in input
- Newline in regular expression
- Newline in string
- EOF in regular expression
- EOF in string
- Cannot open script file
- Inadmissible use of reserved keyword
- Attempt to redefine built-in function
- Cannot open input file
- Error on **print**
- Error on **printf**
- Getline in **END** action was not redirected
- Too many open I/O streams
- Error on I/O stream
- Insufficient arguments to **printf** or **sprintf()**
- Array cannot be used as a scalar
- Variable cannot be used as a function
- Too many fields
- Record too long
- Division (/ or %) by zero
- Syntax error
- Cannot assign to a function
- Value required in assignment
- Return outside of a function
- Can delete only array element or array
- Scalar cannot be used as array
- **SYMTAB** must have exactly one index
- Impossible function call
- Function call nesting level exceeded
- Wrong number of arguments to function
- Regular expression error
- Second parameter to “split” must be an array
- **sprintf** string longer than allowed number of characters
- No open filename
- Function requires an array
- Is not a function
- Failed to match
- Incorrect collation element
- Trailing \ in pattern
- Newline found before end of pattern
- More than 9 \(\) pairs
- Number in [0–9] incorrect
- [ ] imbalance or syntax error
- ( ) or \(\) imbalance
- { } or \{ \} imbalance
- Incorrect endpoint in range
- Out of memory
- Incorrect repetition

- Incorrect character class type
- Internal error
- Unknown *regex* error

When an **awk** program ends because of a call to **exit()**, the exit status is the value passed to **exit()**.

Limits

Most constructions in this implementation of **awk** are dynamic, limited only by memory restrictions of the system.

The maximum record size is guaranteed to be at least LINE_MAX as returned by **getconf**. The maximum field size is guaranteed to be LINE_MAX, also.

The parser stack depth is limited to 150 levels. Attempting to process extremely complicated programs may result in an overflow of this stack, causing an error.

Input must be text files.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The **ord** function is an extension to traditional implementations of **awk**. The **toupper** and **tolower** functions and the **ENVIRON** array are in POSIX and the UNIX System V Release 4 version of **awk**. This version is a superset of New awk, as described in *The AWK Programming Language* by Aho, Weinberger, and Kernighan.

The *standard command interpreter* that the *system* function uses and that **awk** uses to run pipelines for **getline**, **print**, and **printf** is system-dependent. On OS/390 UNIX, this interpreter is always **/bin/sh**.

Related Information

ed, **egrep**, **sed**, **vi**

For more information about **regexp**, see Appendix C.

---

basename — Return the nondirectory components of a pathname

Format

basename *name* [*suffix*]

Description

basename strips off the leading part of a pathname, leaving only the final component of the name, which is assumed to be the filename. To accomplish this, **basename** first checks to see if *name* consists of nothing but slash (/) characters. If so, **basename** replaces *name* with a single slash and the process is complete. If not, **basename** removes trailing slashes. If slashes still remain, **basename** strips off all leading characters up to and including the final slash. Finally, if you specify

suffix and the remaining portion of *name* contains a suffix that matches *suffix*, **basename** removes that suffix.

Examples

The command:

```
basename src/dos/printf.c
```

produces:

```
printf
```

Localization

basename uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion
1 Failure due to any of the following:
- Unknown command-line option
- Incorrect number of arguments

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Information

dirname

---

batch — Run commands when the system is not busy

Format

batch

Description

batch lets you run commands in batch mode. It reads the commands from the standard input (**stdin**). The system records the commands and runs them at a time when the system load is relatively low (that is, when the system is not busy).

The **batch** command is equivalent to

```
at –q b –m now
```

For more details, see **at**.

at, **batch**, and **crontab** submit jobs to **cron**; the data in those jobs may contain doublebyte characters. When the jobs are run, the data in the jobs are interpreted in the locale that **cron** is using. Since it is strongly recommended that **cron** be started in the POSIX locale, doublebyte characters in the job may not be interpreted correctly. You may be able to get around this by calling **setlocale()** in the job itself.

Environment Variables

batch uses the following environment variable:

SHELL Contains the name of the shell command interpreter used to invoke the **batch** job.

Localization

batch uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

`0` Successful completion

`>0`
Returned if the command fails for any reason

If an error occurs, **batch** does not schedule the job.

Portability

POSIX.2 User Portability Extension.

Related Information

at, **bg**, **crontab**

---

bc — Use the arbitrary-precision arithmetic calculation language

Format

bc [**–i**] [**–l**] [*file*]

Description

bc is a programming language that can perform arithmetic calculations to arbitrary precision. You can use it interactively, by entering instructions from the terminal. It can also run programs taken from files.

The *file* arguments you specify on the command line should be text files containing **bc** instructions. **bc** runs the instructions from those files, in the order that they appear on the command line, and then runs instructions from the standard input

(**stdin**). **bc** ends when it runs a **quit** instruction or reaches the end of the file on **stdin**.

bc is a simple but complete programming language with a syntax reminiscent of the C programming language. This version of **bc** is a superset of the standard language available on most systems. It has a number of additional features intended to make the language more flexible and useful. Features unique to this implementation are noted.

Input consists of a series of instructions that assign values to variables or make calculations. It is also possible to define subprograms called *functions,* which perform a sequence of instructions to calculate a single value.

bc displays the result of any line that calculates a value, but does not assign it to a variable. For example, the instruction:

```
2+2
```

displays:

```
4
```

By default, **bc** displays the result of any evaluated instruction followed by a newline. **bc** also saves the last value displayed in a special variable **.** (dot), so that you can use it in subsequent calculations.

Options

bc supports the following options.

–i Puts **bc** into interactive mode with a displayed prompt. In this mode, **bc** displays a prompt, which is ":"—waiting for input. In addition, it handles errors somewhat differently. Normally, when **bc** encounters an error while processing a file, the interpreter displays the error message and exits. In interactive mode, the interpreter displays the message and returns to the prompt mode to allow debugging.

–l Loads a library of standard mathematical functions before processing any other input. This library also sets the *scale* to 20. For a description of the functions in the **–l** library, see "Built-in Functions" on page 56.

Numbers

Numbers consist of an optional minus (–) sign or an optional plus (+) sign followed by a sequence of zero or more digits, followed by an optional decimal point (**.**), followed by a sequence of zero or more digits. Valid digits are `0` through `9`, and the hexadecimal digits `A` through `F`. The uppercase letters represent the values from `10` through `15`. There must be at least one digit, either before or after the decimal point. If not, **bc** interprets the decimal point as the special variable **..**

A number can be arbitrarily long and can contain spaces. Here are some valid numbers with an input base of 10:

```
0    0.   .0   -3.14159   +09.    -12    1 000 000
```

Here are some valid numbers with an input base of 16 (*ibase*=16):

```
0    FF    FF.3    -10.444   A1
```

See “Bases” on page 47 for more information.

A final point is that you cannot break up numbers with commas; you can write `1000000` or `1 000 000`, but `1,000,000` results in an error message.

Identifiers

Identifiers can include sequences containing any number of letters, digits, or the underscore (_) character but must start with a lowercase letter. Spaces are not allowed in identifiers.

In the POSIX locale, valid identifiers can include sequences containing any number of letters, digits, or the underscore (_) character but must start with a lowercase letter, as defined by the current locale.

For other locales, the character map for that locale determines which characters are valid in an identifier. If you want identifiers to be portable between locales, use characters from the POSIX character set. The use of identifiers longer than one character is an extension of this implementation. Identifiers are used as names for variables, functions, or arrays:

- A *variable* holds a single numeric value. You can declare variables that are local to a function using the **auto** statement. (See “Functions” on page 54). All other variables are global and you can use them inside any function or outside all functions. You do not need to declare global variables. **bc** creates variables as it requires them, with an initial value of zero. (Remember that there is also the special variable **.** [dot], which contains the result of the last calculation.)
- A *function* is a sequence of instructions that calculates a single value. A list of zero or more values enclosed in parentheses always follow a function name, as in **my_func(3.14159)**. See “Functions” on page 54.
- An *array* is a list of values. Values in the list are called *elements* of the array. These elements are numbered, beginning at zero. We call such a number a *subscript,* or *index,* of the array. Subscripts always appear in square brackets after the array. For example, **a[0]** refers to element zero in the array **a**. The first element of the array always has the subscript `0`. If a subscript value is a floating-point number, the fractional part is discarded to make the subscript into an integer. For example, the following expressions all refer to the same element:

  ```
  a[3]   a[3.2]   a[3.999]
  ```

  The maximum number of elements in a **bc** array is in the range from `0` to {*BC_DIM_MAX*}–1 inclusive. Unlike with many languages, you don't need to declare the size of an array. Elements are created dynamically as required, with an initial value of zero.

Since parentheses always follow function names and square brackets always follow array names, **bc** can distinguish between all three types of names—variable names, function names, and array names. Therefore, you can have variables, functions, and arrays with the same name. For example, *foo* may be a variable whereas **foo()** is a function and **foo[ ]** is an array.

Built-in Variables

bc has a number of built-in variables that are used to control various aspects of the interpreter. These are described in the following sections.

Scale

The *scale value* is the number of digits to be retained after the decimal point in arithmetic operations. For example, if the scale is 3, each calculation retains at least three digits after the decimal point. This means that:

```
5 / 3
```

has the value:

```
1.666
```

If **–l** is specified, the scale is set to 20; otherwise, the default scale is zero.

The variable *scale* holds the current scale value. To change scales, assign a new value to *scale*, as in:

```
scale = 5
```

Since *scale* is just a regular **bc** variable, it can be used in the full range of **bc** expressions.

The number of decimal places in the result of a calculation is affected not only by the scale, but also by the number of decimal places in the operands of the calculation. discusses this. “Arithmetic Operations” on page 48 discusses this.

There is also a function **scale**, which can determine the scale of any expression. For example, **scale(1.1234)** returns the result 4, which is the scale of the number 1.1234. The result of the **scale** function is always an integer (that is, it has the scale of `0`).

The maximum value for **scale** is given by the configuration variable {*BC_SCALE_MAX*} and the minimum value is `0`.

Bases

bc lets you specify numbers in different bases—for example, octal (base 8) or hexadecimal (base 16). You can input numbers in one base and output them in a different base, simplifying the job of converting from one base to another. **bc** does this using the built-in variables *ibase* and *obase*.

ibase is the base for input numbers. It has an initial value of 10 (normal decimal numbers). To use a different base for inputting numbers, assign an integer to *ibase*, as in:

```
ibase = 8
```

This means that all future input numbers are to be in base 8 (octal). The largest valid input base is 16, and the smallest valid input base is 2. There is no mechanism provided to represent digits larger than 15, so bases larger than 16 are essentially useless. When the base is greater than 10, use the uppercase letters as digits. For example, base 16 uses the digits `0` through `9`, and `A` through `F`. The digits are allowed in any number, regardless of the setting of *ibase* but are largely meaningless if the base is smaller than the digit. The one case where this is useful is in

resetting the input base to 10. The constant `A` always has the value 10 no matter what *ibase* is set to, so to reset the input base to 10, type:

```
ibase = A
```

obase is the base in which numbers are output. It has an initial value of 10 (normal decimal numbers). To change output bases, assign an appropriate integer to *obase*.

If the output base is 16 or less, **bc** displays numbers with normal digits and hexadecimal digits (if needed). The output base can also be greater than 16, in which case each *digit* is printed as a decimal value and digits are separated by a single space. For example, if *obase* is 1000, the decimal number 123 456 789 is printed as:

```
123 456 789
```

Here, the digits are decimal values from `0` through `999`. As a result, all output values are broken up into one or more *chunks* with three digits per chunk. Using output bases that are large powers of 10, you can arrange your output in columns; for example, many users find that 100 000 makes a good output base, because numbers are grouped into chunks of five digits each.

Long numbers are output with a maximum of 70 characters per line. If a number is longer than this, **bc** puts a backslash (\) at the end of the line, indicating that the number is continued on the next line.

Internal calculations are performed in decimal, regardless of the input and output bases. Therefore the number of places after the decimal point are dictated by the scale when numbers are expressed in decimal form.

The maximum value for **obase** is given by the configuration variable {*BC_BASE_MAX*}.

Arithmetic Operations

bc provides a large number of arithmetic operations. Following standard arithmetic conventions, some operations are calculated before others. For example, multiplications take place before additions unless you use parentheses to group operations. Operations that take place first are said to have a higher *precedence* than operations that take place later.

Operations also have an *associativity*. The associativity dictates the order of evaluation when you have a sequence of operations with equal precedence. Some operations are evaluated left to right, whereas others are evaluated right to left. The following list shows the operators of **bc** from highest precedence to lowest.

bc Operator	**Associativity**
( )	Left to right
Unary ++ ––	Not applicable
Unary – !	Not applicable
	Right to left, or Left to right
+ –	Left to right
= = *= /= %= +=	Right to left
== <= >= != < >	None
&&	Left to right
||	Left to right

bc's order of precedence is not the same as C's. In C, the assignment operators have the lowest precedence.

The following list describes what each operation does. In the descriptions, A and B can be numbers, variables, array elements, or other expressions. V must be either a variable or an array element.

(A) Indicates that this expression—A—should be evaluated before any other operations are performed on it.

-A Is the negation of the expression.

!A Is the logical complement of the expression. If A evaluates to zero, !A evaluates to 1. If A is not zero, !A evaluates to zero. This operator is unique to this version of **bc**.

++V Adds 1 to the value of V. The result of the expression is the new value of V.

––V Subtracts 1 from the value of V. The result of the expression is the new value of V.

V++ Adds 1 to the value of V, but the result of the expression is the old value of V.

V–– Subtracts 1 from the value of V, but the result of the expression is the old value of V.

A ^ B Calculates A to the power B. B must be an integer. The scale of the result of A^B is:

```
min(scale(A) * abs(B), max(scale, scale(A)))
```

where **min** calculates the minimum of a set of numbers and **max** calculates the maximum.

A * B Calculates A multiplied by B. The scale of the result is:

```
min(scale(A) + scale(B), max(scale, scale(A), scale(B)))
```

A / B Calculates A divided by B. The scale of the result is the value of *scale*.

A % B Calculates the remainder from the division of A by B. This is calculated in two steps. First, **bc** calculates A/B to the current scale. It then obtains the remainder through the formula:

```
A - (A / B) * B
```

calculated to the scale:

```
max(scale + scale(B), scale(A))
```

A + B Adds A plus B. The scale of the result is the maximum of the two scales of the operands.

A–B Calculates A minus B. The scale of the result is the maximum of the two scales of the operands.

The next group of operators are all *assignment* operators. They assign values to objects. An assignment operation has a value: the value that is being assigned. Therefore, you can write such operations as a=1+(b=2). In this operation, the value of the assignment in parentheses is 2 because that is the value assigned to b. Therefore, the value 3 is assigned to a. The possible assignment operators are:

V = B Assigns the value of B to V.

V ⅛= B Is equivalent to V=V⅛B.

V *= B Is equivalent to V=V*B.

V /= B Is equivalent to V=V/B.

V %= B Is equivalent to V=V%B.

V += B Is equivalent to V=V+B.

V –= B Is equivalent to V=V-B.

The following expressions are called *relations*, and their values can be either true (1) or false (0). This version of **bc** lets you use the relational operators in any expression, not just in the conditional parts of **if**, **while**, or **for** statements. These operators work exactly like their equivalents in the C language. The result of a relation is 0 if the relation is false and 1 if the relation is true.

A == B Is true if and only if A equals B.

A <= B Is true if and only if A is less than or equal to B.

A >= B Is true if and only if A is greater than or equal to B.

A != B Is true if and only if A is not equal to B.

A < B Is true if and only if A is less than B.

A > B Is true if and only if A is greater than B.

A && B Is true if and only if A is true (nonzero) and B is true. If A is not true, the expression B is never evaluated.

A ll B Is true if A is true or B is true. If A is true, the expression B is never evaluated.

Comments and White Space

A *comment* has the form:

```
/* Any string */
```

Comments can extend over more than one line of text. When **bc** sees /* at the start of a The only effect a comment has is to indicate the end of a token. As an extension, this version of **bc** also provides an additional comment convention using the # character. All text from the # to the end of the line is treated as a single blank, as in:

```
2+2 # this is a comment
```

bc is free format. You can freely insert blanks or horizontal tab characters to improve the readability of the code. Instructions are assumed to end at the end of the line. If you have an instruction that is so long you need to continue it on a new line, put a backslash (\) as the very last character of the first line and continue on the second, as in:

```
a = 2\
 + 3
```

The \ indicates that the instruction continues on the next line, so this is equivalent to:

```
a = 2 + 3
```

Instructions

A **bc** instruction can be an expression that performs a calculation, an assignment, a function definition, or a statement. If an instruction is not an assignment, **bc** displays the result of the instruction when it has completed the calculation. For example, if you enter:

```
3.14 * 23
```

bc displays the result of the calculation. However, with:

```
a = 3.14 * 23
```

bc does not display anything, because the expression is an assignment. If you do want to display the value of an assignment expression, simply place the expression in parentheses.

The following list shows the instruction forms recognized by **bc**:

expression Calculates the value of the *expression*.

"string" Is a string constant. When **bc** sees a statement of this form, it displays the contents of the string. For example:

```
"Hello world!"
```

tells **bc** to display `Hello world!` A newline character is *not* output after the string. This makes it possible to do things like:

```
foo = 15
"The value of foo is "; foo
```

With these instructions, **bc** displays

```
The value of foo is 15
```

statement **;** *statement ...*
Is a sequence of statements on the same line. In **bc**, a semicolon (**;**) and a newline are equivalent. They both indicate the end of a statement. **bc** runs these statements in order from left to right.

{*statement*} Is a brace-bracketed statement. Brace brackets are used to group sequences of statements together, as in:

```
{
  statement
  statement
     ...
}
```

Brace brackets can group a series of statements that are split over several lines. Braces are usually used with control statements like **if** and **while**.

break Can be used only inside a **while** or **for** loop. `break` ends the loop.

for (*initexp* **;** *relation* **;** *endexp*) *statement*
Is equivalent to:

```
initexp
while (relation) {
    statement
    endexp
}
```

where *initexp* and *endexp* are expressions and *relation* is a relation. For example:

```
a = 0
for (i = 1; i <= 10; ++i) a += i
```

is equivalent to the **while** example given earlier.

Note: All three items inside the parentheses must be specified. Unlike C, **bc** does not let you omit any of these expressions.

if (*relation***)** *statement*

Tests whether the given *relation* is true. If so, **bc** runs the *statement*; otherwise, **bc** skips over the *statement* and goes to the next instruction. For example:

```
if ((a%2) == 0) "a is even"
```

displays `a is even` if `a` has an even value.

if (*relation***)** *statement1* **else***statement2*

Is similar to the simple **if** statement. It runs *statement1* if *relation* is true and otherwise runs *statement2*. It may be used as follows:

```
if ((a%2) == 0) "a is even" else "a is odd"
```

Note: There is no statement separator between “a is even” and the **else** keyword. This differs from the C language.

Here is another example:

```
if (a<10) {
        "a "
        "is "; "less than 10 "
        a
} else {
        "a is"
        " greater than 10 "
        a
}
```

Note: The braces must be on the same line as the **if** and the **else** keywords. This is because a new line or a semicolon right after (*relation*) indicates that the body of the statement is null. One common source of errors in **bc** programs is typing the statement body portion of an **if** statement on a separate line. If **–i** is used, the interpreter displays a warning when **if** statements with null bodies are encountered.

while (*relation***)** *statement*

Repeatedly runs the given *statement* while *relation* is true. For example:

```
i = 1
a = 0
while (i <= 10) {
    a += i
    ++i
}
```

adds the integers from 1 through 10 and stores the result in `a`.

If *relation* is not true when **bc** encounters the **while** loop, **bc** does not run *statement* at all.

print *expression* **,** *expression ...*

Displays the results of the argument expressions. Normally, **bc** displays the value of each expression or string it encounters. This makes it difficult to format your output in programs. For this reason, the OS/390 shell version of **bc** has a **print** statement to give you more control over how things are displayed. **print** lets you display several numbers on the same line with strings. This statement displays all its arguments on a single line. A single space is displayed between adjacent numbers (but not between numbers and strings). A **print** statement with no arguments displays a newline. If the last argument is null, subsequent output continues on the same line. Here are some examples of how to use **print**:

```
/* basic print statement */
print "The square of ", 2, "is ", 2*2
The square of 2 is 4

/* inserts a space between adjacent numbers */
print 1,2,3
1 2 3

/* note - no spaces */
print 1,"",2,"",3
123

/* just print a blank line */
print

/* two statements with output on same line */
print 1,2,3, ; print 4, 5, 6
1 2 3 4 5 6
```

quit Ends **bc**. In other implementations of **bc**, the interpreter exits as soon as it reads this token. This version of **bc** treats **quit** as a real statement, so you can use it in loops, functions, and so on.

sh ... Lets you send a line to the system command interpreter for execution, as in:

```
sh more <foo
```

This command passes everything from the first nonblank character until the end of the line to the command interpreter for execution.

void *expression*

Throws away, or “voids,” the result of the evaluation of *expression* instead of displaying it. This instruction is useful when using ++ and -- operators, or when you want to use a function but don't want to use the return value for anything. For example:

```
void foo++
```

increments `foo` but does not display the result. The **void** statement is unique to this version of **bc**.

Several other types of statements are relevant only in function definitions. These are described in the next section.

Functions

A function is a *subprogram* to calculate a result based on *argument* values. For example, the following function converts a temperature given in Fahrenheit into the equivalent temperature in Celsius:

```
define f_to_c(f) {
    return ((f-32) * 5 / 9)
}
```

This defines a function named **f_to_c()** that takes a single argument called `f`. The *body* of the function is enclosed in brace brackets. The opening brace must be on the same line as the **define** keyword. The function body consists of a sequence of statements to calculate the *result* of the function. An expression of the form:

return (*expression*)

returns the value of *expression* as the result of the function. The parentheses around the expression are optional.

To activate the subprogram you use a *function call*. This has the form:

name(*expression*,*expression*,...)

where **name** is the name of the function, and the *expression*s are argument values for the function. You can use function call anywhere you might use any other expression. The value of the function call is the value that the function returns. For example, with the function **f_to_c()**, described earlier, **f_to_c(41)** has the value `5` (since 41 Fahrenheit is equivalent to 5 Celsius).

The general form of a function definition is:

```
define name(parameter,parameter,...) {
    auto local, local, ...
    statement
    statement
          ...
}
```

Each *parameter* on the first line can be a variable name or an array name. Array names are indicated by putting square brackets after them. For example, if **cmpvec** is a function that compares two vectors, the function definition might start with:

```
define cmpvec(a[],b[]) {
```

Parameters do not conflict with arrays or variables of the same name. For example, you can have a parameter named `a` inside a function, and a variable named `a` outside, and the two are considered entirely separate entities. Assigning a value to the variable does not change the parameter and vice versa. All parameters are *passed by value*. This means that a copy is made of the argument value and is assigned to the formal parameter. This also applies to arrays. If you pass an array to a function, a copy is made of the whole array, so any changes made to the array parameter do not affect the original array.

A function may not need any arguments. In this case, the **define** line does not have any parameters inside the parentheses, as in:

```
define f() {
```

The **auto** statement declares a sequence of *local* variables. When a variable or array name appears in an **auto** statement, the current values of those items are saved and the items are initialized to zero. For the duration of the function, the items have their new values. When the function ends, the old values of the items are restored.

However, **bc** uses dynamic scoping rules, unlike C which uses lexical scoping rules.

See "Usage Notes" on page 59 for more details.

For example:

```
define addarr(a[],l) {
    auto i, s
    for (i=0; i < l; ++i) s += a[i]
    return (s)
}
```

is a function that adds the elements in an array. The argument `l` stands for the number of elements in the array. The function uses two local names: a variable named *i* and a variable named *s*. These variables are "local" to the function **addarr** and are unrelated to objects of the same name outside the function (or in other functions). Objects that are named in an **auto** statement are called *autos*. Autos are initialized to `0` each time the function is called. Thus, the sum *s* is set to zero each time this function is called. You can also have local arrays, which are specified by placing square brackets after the array name in the **auto** statement.

```
define func_with_local_array() {
        auto local_array[];
        for(i=0; i<100; i++) local_array[i] = i*2
}
```

This example defines a local array called **local_array**. Local arrays start out with no elements in them.

If a function refers to an object that is not a parameter and not declared **auto**, the object is assumed to be *external*. External objects may be referred to by other functions or by statements that are outside of functions. For example:

```
define sum_c(a[],b[ ],l) {
    auto i
    for (i=0; i < l; ++i) c[i] = a[i] + b[i]
}
```

refers to an external array named **c**, which is the element-by-element sum of two other arrays. If **c** did not exist prior to calling **sum_c**, it is created dynamically. After the program has called **sum_c**, statements in the program or in functions can refer to array **c**.

Functions usually require a `return` statement. This has the form:

```
return (expression)
```

The argument *expression* is evaluated and used as the result of the function. The expression must have a single numeric value; it cannot be an array.

A **return** statement ends a function, even if there are more statements left in the function. For example:

```
define abs(i) {
    if (i < 0) return (-i)
    return (i)
}
```

is a function that returns the absolute value of its argument. If *i* is less than zero, the function takes the first **return**; otherwise, it takes the second.

A function can also end by running the last statement in the function. If so, the result of the function is zero. The function **sum_c** is an example of a function that does not have a **return** statement. The function does not need a **return** statement, because its work is to calculate the external array **c**, not to calculate a single value. Finally, if you want to return from a function, but not return a value you can use **return()** or simply **return**. If there are no parameters to the **return** statement, a default value of zero is returned.

Built-in Functions

bc has a number of built-in functions that perform various operations. These functions are similar to user-defined functions. You do not have to define them yourself, however; they are already set up for you. These functions are:

length(*expression***)**
: Calculates the total number of decimal digits in *expression*. This includes digits both before and after the decimal point. The result of **length()** is an integer. For example, **length(123.456)** returns `6`.

scale(*expression***)**
: Returns the scale of *expression*. For example, **scale(123.456)** returns `3`. The result of **scale()** is always an integer. Subtracting the scale of a number from the length of a number lets you determine the number of digits before the decimal point.

sqrt(*expression***)**
: Calculates the square root of the value of *expression*. The result is truncated in the least significant decimal place (not rounded). The scale of the result is the scale of *expression*, or the value of **scale()**, whichever is larger.

You can use the following functions if **–l** is specified on the command line. If it is not, the function names are not recognized. There are two names for each function: a full name, and a single character name for compatibility with POSIX.2. The full names are the same as the equivalent functions in the standard C math library.

arctan(*expression*) or **a**(*expression*)
: Calculates the arctangent of *expression*, returning an angle in radians. This function can also be called as **atan**(*expression*).

bessel(*integer*,*expression*) or **j**(*integer*,*expression*)
: Calculates the Bessel function of *expression*, with order *integer*. This function can also be called as **jn**(*integer*,*expression*).

cos(*expression*) or **c**(*expression*)
: Calculates the cosine of *expression*, where *expression* is an angle in radians.

exp(*expression*) or **e**(*expression*)
Calculates the exponential of *expression* (that is, the value **e** to the power of *expression*).

ln(*expression*) or **l**(*expression*)
Calculates the natural logarithm of *expression*. This function can also be called as **log**(*expression*).

sin(*expression*) or **s**(*expression*)
Calculates the sine of *expression*, where *expression* is an angle in radians.

Examples

1. Here is a simple function to calculate the sales tax on a purchase. The amount of the purchase is given by *purchase*, and the amount of the sales tax (in per cent) is given by *tax*.

```
define sales_tax(purchase,tax) {
    auto old_scale
    scale = 2
    tax = purchase*(tax/100)
    scale = old_scale
    return (tax)
}
```

   For example:

```
sales_tax(23.99,6)
```

   calculates 6% tax on a purchase of $23.99. The function temporarily sets the scale value to 2 so that the monetary figures have two figures after the decimal point. Remember that **bc** truncates calculations instead of rounding, so some accuracy may be lost. It is better to use one more digit than needed and perform the rounding at the end. The **round2** function, shown later in this section, rounds a number to two decimal places.

2. Division resets the scale of a number to the value of *scale*. You can use this to extract the integer portion of a number, as follows:

```
define integer_part(x) {
        # a local to save the value of scale
        auto old_scale
        # save the old scale, and set scale to 0
        old_scale = scale; scale=0
        # divide by 1 to truncate the number
        x /= 1
        # restore the old scale
        scale=old_scale
        return (x)
}
```

3. Here is a function you can define to return the fractional part of a number:

```
define fractional_part(x) {return (x - integer_part(x))}
```

4. The following function lets you set the scale of number to a given number of decimal places:

```
define set_scale(x, s)
     {  auto os
        os = scale
        scale = s
        x /= 1
        scale = os
        return (x)  }
```

You can now use **set_scale()** in a function that rounds a number to two decimal places:

```
define round2(num) {
        auto temp;
        if(scale(num) < 2) return (set_scale(num, 2))
        temp = (num - set_scale(num, 2)) * 1000
        if(temp > 5) num += 0.01
        return (set_scale(num,2))
}
```

This is a very useful function if you want to work with monetary values. For example, you can now rewrite **sales_tax()** to use **round2()**:

```
define sales_tax(purchase,tax) {
    auto old_scale
    scale = 2
    tax = round2(purchase*(tax/100))
    scale = old_scale
    return (tax)
}
```

5. Here is a function that recursively calculates the factorial of its argument:

```
define fact (x) {
        if(x < 1) return 1
        return (x*fact(x-1))
}
```

You can also write the factorial function iteratively:

```
define fact (x) {
        auto result
        result = 1
        while(x>1) result *= x--
        return (result)
}
```

With either version, **fact(6)** returns `720`.

6. Here is another recursive function, that calculates the *n*th element of the Fibonacci sequence:

```
define fib(n) {
        if(n < 3) {
                return (1)
        } else {
                return (fib(n-1)+fib(n-2))
        }
}
```

Usage Notes

1. Unlike the C language, which uses lexical scoping rules, **bc** uses dynamic scoping, which is most easily explained with an example:

```
a=10
define f1() {
        auto a;
        a = 13;
        return (f2())
}
define f2() {
        return (a)
}
f1()
13
f2()
10
```

   If **f1()** is called, **bc** prints the number 13, instead of the number 10. This is because **f1()** hides away the old (global) value of `a` and then sets it to 13. When **f2()** refers to `a`, it sees the variable dynamically created by **f1()** and so prints 13. When **f1()** returns, it restores the old value of `a`. When **f2()** is called directly, instead of through **f1()**, it sees the global value for `a` and prints 10. The corresponding C code prints 10 in both cases.

2. Numbers are stored as strings in the program and converted into numbers each time they are used. This is important because the value of a "constant" number may change depending on the setting of the *ibase* variable. For example, suppose the following instructions are given to **bc**:

```
define ten() {
        return (10)
}
ten()
10
ibase=16
ten()
16
```

   In this example, when the base is set to 10, **ten()** returns the decimal value 10. However, when the input base is changed to 16, the function returns the decimal value 16. This can be a source of confusing errors in **bc** programs.

3. The library of functions loaded using the **–l** option is stored in the file **/usr/lib/lib.b** under your root directory. This is a simple text file that you can examine and change to add new functions as desired.

Files

bc uses the following file:

/usr/lib/lib.b File containing the library of functions loaded with **–l**

Localization

bc uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Values

`0` Successful completion

`1` Failure due to any of the following errors:

- Break statement found outside loop
- Parser stack overflow
- Syntax error
- End of file in comment
- End of file in string
- Numerical constant is too long
- String is too long
- Empty evaluation stack
- Cannot pass scalar to array
- Cannot pass array to scalar
- Incorrect array index
- Built-in variable cannot be used as a parameter or auto variable
- *name* is not a function
- Incorrect value for built-in variable
- Shell command failed to run
- Division by `0`
- Incorrect value for exponentiation operator
- Attempt to take square root of negative number
- Out of memory

`2` Unknown command-line option

Limits

The parser stack depth is limited to 150 levels. Attempting to process extremely complicated programs may result in an overflow of this stack, causing an error.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The following are extensions to the POSIX standard:

- The **–i** option
- The &&and || operators
- The **if ... else ...** statement
- Identifiers of more than one character or containing characters outside the POSIX character set

- The **print** statement
- The **sh** statement
- The optional parentheses in the **return** statement

In a doublebyte environment, remember that only numbers and operators from the POSIX character set can be used. Identifiers can use characters from the current locale; if you want scripts to be portable, use only characters from the POSIX character set.

---

bg — Move a job to the background

Format

bg [*job...*]

tcsh shell: **bg** [*%job ...*]

Description

bg runs one or more jobs in the background. The *job* IDs given on the command line identify these jobs, which should all be ones that are currently stopped. If you do not specify any *job* IDs, **bg** uses the most recently stopped job.

bg works only if job control is enabled; see the **–m** option of **set** for more information. Job control is enabled by default in the OS/390 shell.

bg in the tcsh shell

In the tcsh shell, **bg** puts the specified jobs (or, without arguments, the current job) into the background, continuing each if it is stopped. *job* may be a number, a string, '', %, + or - .

In the tcsh shell, **%job &** is a synonym of the **bg** command.

Localization

bg uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Usage Note

bg is a built-in shell command.

Exit Values

0 Successful completion

>0
Failure because a *job* argument is incorrect or there is no current job

If an error occurs, **bg** exits and does not place the job in the background.

Portability

POSIX.2 User Portability Extension, UNIX systems.

Related Information

at, **batch**, **fg**, **jobs**, **set**, **tcsh**

---

bpxmtext — Display reason code text

Format

bpxmtext *reason_code*

Description

bpxmtext displays the description and action text for a reason code returned from the kernel. Non-kernel reason codes such as those returned by HFS or NFS are not supported by this command. This command is intended as an aid for problem determination.

reason_code is specified as 8 hexadecimal characters. Leading zeros may be omitted.

Usage Notes

If no text is available for the reason code, a blank line is displayed.

An argument that is not 1–8 hex digits will result in a usage message. This message will not be translated.

Examples

The command:

```
bpxmtext 058800B0
```

produces data displayed in the following format:

```
BPXFSUMT 08/18/98
JRUserNotPrivileged:  The requester of the service is not privileged

Action:  The service requested required a privileged user.  Check the
documentation for the service to understand what privilege is required.
```

Exit Values

0 Successful completion

2 Failure due to an argument that is not 1–8 hex digits

---

break — Exit from a loop in a shell script

Format

break [*number*]

tcsh shell: **break**

Description

break exits from a **for**, **select**, **while**, or **until** loop in a shell script. If *number* is given, **break** exits from the given number of enclosing loops. The default value of *number* is 1.

break in the tcsh shell

In the tcsh shell, **break** causes execution to resume after the end of the nearest enclosing **foreach** or **while**. The remaining commands on the current line are executed. Multi-level breaks are thus possible by writing them all on one line.

Localization

break uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Usage Note

break is a special built-in shell command.

Exit Value

break always exits with an exit status of 0.

Portability

POSIX.2, X/Open Portability Guide.

Related Information

continue, **sh**, **tcsh**

c++ — Compile C++/MVS and C/MVS and create an executable file

See **c89/cc/c++**.

Note: When working in the shell, to view man page information about **c++**, type:
`man c89`

c89 — Compile ANSI C/MVS source code and create an executable file

Format

c89 | **cc** | **c++** | **cxx** [**–+CcEFfgOpqrsVv2**]
[**–D** *name*[=*value*]]... [**–U** *name*]...
[**–e** *function*] [**–u** *function*]...
[**–W** *phase,option*[,*option*]...]...
[**–o** *outfile*]
[**–I** *directory*]... [**–L** *directory*]...
[*file.C*]... [*file.i*]... [*file.c*]... [*file.s*]...
[*file.a*]... [*file.o*]... [*file.x*]... [**–l** *libname*]...

Note: The **I** option signifies an uppercase **i**, not a lowercase **L**.

Description

c89, **cc**, and **c++**compile, assemble, and link-edit OS/390 C and OS/390 C++ programs.

- **c89** should be used when compiling C programs that are written according to *Standard C*.
- **cc** should be used when compiling C programs that are written according to *Common Usage C*.
- **c++** must be used when compiling C++ programs, which are written according to *Draft Proposal International Standard for Information Systems — Programming Language C++ (X3J16)*. **c++** can compile both C++ and C programs, and can also be invoked by the name **cxx** (all references to **c++** throughout this document apply to both names).

First, **c89**, **cc**, and **c++** perform the compilation phase (including preprocessing) by compiling all source file operands (*file.C*, *file.i*, and *file.c*, as appropriate). For **c++**, if automatic template generation is being used (which is the default), then OS/390 C++ source files may be created or updated in the **tempinc** subdirectory of the working directory during the compilation phase (the **tempinc** subdirectory will be created if it does not already exist). Then, **c89**, **cc**, and **c++** perform the assemble phase by assembling all operands of the *file.s* form. The result of each compile step and each assemble step is a *file.o* file. If all compilations and assemblies are successful, or if only *file.o* and/or *file.a* files are specified, **c89**, **cc**, and **c++** proceed to the link-editing phase. For **c++**, the link-editing phase begins with an automatic template generation step when applicable. For IPA (Interprocedural Analysis) optimization an additional IPA link step comes next. The link-edit step is last. See the environment variable **{_STEPS}** under “Environment Variables” on page 74 for more information about the link-editing phase steps.

In the link-editing phase, **c89**, **cc**, and **c++** combine all *file.o* files from the compilation phase along with any *file.o* files that were specified on the command line. For

c++, this is preceded by compiling all OS/390 C++ source files in the **tempinc** subdirectory of the working directory (possibly creating and updating additional OS/390 C++ source files during the automatic template generation step). After compiling all the OS/390 C++ source files, the resulting object files are combined along with the *file.o* files from the compilation phase and the command line. Any *file.a* files, *file.x* files and **–l** *libname* operands that were specified are also used.

The usual output of the link-editing phase is an executable file. For **c89**, **cc**, and **c++** to produce an executable file, you must specify at least one operand which is of other than **–l** *libname* form. If **–r** is used, the output file is not executable.

For more information about automatic template generation, see *OS/390 C/C++ User's Guide* and *OS/390 C/C++ Programming Guide*. Note that the **c++** command only supports using the **tempinc** subdirectory of the working directory for automatic template generation.

IPA is further described under the **–W** option on page 71.

Options

–+ Specifies that all source files are to be recognized as C++ source files. All *file.s*, *file.o*, and *file.a* files will continue to be recognized as assembler source, object, and archive files respectively. However, any C *file.c* or *file.i* files will be processed as corresponding C++ *file.C* or *file.i* files, and any other file suffix which would otherwise be unrecognized will be processed as a *file.C* file.

This option effectively overrides the environment variable **{_EXTRA_ARGS}**. This option is only supported by the **c++** command.

–C Specifies that C and C++ source comments should be retained by the preprocessor. By default all comments are removed by the preprocessor. This option is ignored except when used with the **–E** option.

–c Specifies that only compilations and assemblies be done. Link-edit is not done.

–D *name*[=*value*]
Defines a C or C++ macro for use in compilation. If only *name* is provided, a value of 1 is used for the macro it specifies. For information about macros that **c89/cc/c++** automatically define, see Usage Note 5 on page 88. Also see Usage Note 13 on page 90.

–E Specifies that output of the compiler preprocessor phase be copied to **stdout**. Compilation into object and link-edit are not done.

–e *function*
Specifies the name of the function to be used as the entry point of the program. This can be useful when creating a fetchable program, or a non–C or non–C++ main, such as a COBOL program. Non–C++ linkage symbols of up to 1024 characters in length may be specified. You can specify an S-name by preceding the function name with double slash (//). (For more information about S-names, see Usage Note 23 on page 92.)

The function **//ceestart** is the default. When the default function entry point is used, a link editor control statement is generated by **c89/cc/c++** to cause the control section (CSECT) of the same name to be ordered to the beginning of

the executable output file. Specify the name with a trailing blank to disable this behavior, as in **"//ceestart "**.

–F Ignored by **cc**, provided for compatibility with historical implementations of **cc**. Flagged as an error by **c89** and **c++**.

–f Ignored by **cc**, provided for compatibility with historical implementations of **cc**. Flagged as an error by **c89** and **c++**.

–g Specifies that the output file (executable) is to contain symbolic information and is to be loaded into read/write storage, which is required for source-level debugging with **dbx**, the debugger.

When specified for the compilation phase, the object file contains symbolic information for source-level debugging.

When specified for the link-editing phase, the executable file is marked as being serially reusable and will always be loaded into read/write storage.

dbx requires that all the executables comprising the process be loaded into read/write storage so that it can set break points in these executables. When **dbx** is attached to a running process, this cannot be guaranteed because the process was already running and some executables were already loaded. There are two techniques that will guarantee that all the executables comprising the process is loaded into read-write storage:

1. Specify the **–g** option for the link-editing phase of each executable. After this is done, the executable is always loaded into read/write storage.

   Because the executable is marked as being serially reusable, this technique works except in cases where the executable must be marked as being reentrant. For example:

   - If the executable is to be used by multiple processes in the same user space.
   - If the executable is a DLL that is used on more than one thread in a multithreaded program.

   In these cases, use the following technique instead:

2. Do not specify the **–g** option during the link-editing phase so that the executable will be marked as being reentrant. Before invoking the program, export the environment variable **_BPX_PTRACE_ATTACH** with a value of `YES`. After you do this, then executables will be loaded into read/write storage regardless of their reusability attribute.

If you compile an MVS data set source using the **–g** option, you can use **dbx** to perform source-level debugging for the executable file. You must first issue the **dbx use** subcommand to specify a path of double slash (//), causing dbx to recognize that the symbolic name of the primary source file is an MVS data set. For information on the dbx command and its use subcommand, see “ use subcommand for dbx: Set the list of directories to be searched” on page 204.

For more information on using dbx, see *OS/390 UNIX System Services Programming Tools*.

The OS/390 UNIX System Services web page also has more information about **dbx**. Go to

`http://www.s390.ibm.com/products/oe/index.html`

For more information on the **_BPX_PTRACE_ATTACH** environment variable, see *OS/390 UNIX System Services Programming: Assembler Callable Services Reference*.

–I *directory*

Note: The **I** option signifies an uppercase **i**, not a lowercase **L**.

–I specifies the directories to be used during compilation in searching for *include files* (also called *header files*).

Absolute pathnames specified on **#include** directives are searched exactly as specified. The directories specified using the **–I** option or from the usual places are not searched.

If absolute pathnames are not specified on **#include** directives, then the search order is as follows:

1. Include files enclosed in double quotes (") are first searched for in the directory of the file containing the **#include** directive. Include files enclosed in angle-brackets (< >) skip this initial search.
2. The include files are then searched for in all directories specified by the **–I** option, in the order specified.
3. Finally, the include files are searched for in the usual places. (See Usage Note 4 on page 88 for a description of the usual places.)

You can specify an MVS data set name as an include file search directory. Also, MVS data set names can explicitly be specified on **#include** directives. You can indicate both by specifying a leading double slash (//). For example, to include the include file DEF that is a member of the MVS PDS ABC.HDRS, code your C or C++ source as follows:

```
#include <//'abc.hdrs(def)'>
```

MVS data set include files are handled according to OS/390 C/C++ compiler conversion rules (see Usage Note 4 on page 88). When specifying an **#include** directive with a leading double slash (in a format other than**#include**<**//'***dsname***'**> and **#include**<**//dd:***ddname*>), the specified name is paired only with MVS data set names specified on the **–I** option. That is, when you explicitly specify an MVS data set name, any hierarchical file system (HFS) directory names specified on the **–I** option are ignored.

–L *directory*

Specifies the directories to be used to search for archive libraries specified by the **–l** operand. The directories are searched in the order specified, followed by the usual places. You cannot specify an MVS data set as an archive library directory.

For information on specifying C370LIB libraries, see the description of the **–l** *libname* operand. Also see Usage Note 7 on page 89 for a description of the usual places.

–0, –O (–1), –2

Specifies the level of compiler optimization (including inlining) to be used. The level **–1** (number one) is equivalent to **–O** (letter capital O). The level **–2** gives the highest level of optimization. The default is **–0** (level 0), no optimization and no inlining, when not using IPA (Interprocedural Analysis).

When using IPA, the default is **–0** (level 1) optimization and inlining. IPA optimization is independent from and can be specified in addition to this optimization level. IPA is further described under the **–W** option on page 71.

If you compile your program to take advantage of **dbx** source-level debugging (see the **–g** option on page 66), you will always get **–0** (level zero) optimization regardless of which of these compiler optimization levels you specify.

In addition to using optimization techniques, you may want to control writable strings by using the **#pragma strings(readonly)** directive or the ROSTRING compiler option. For more information on this topic, refer to the chapter on reentrancy in OS/390 C/C++ in *OS/390 C/C++ Programming Guide* or the description of the ROSTRING option in the *OS/390 C/C++ User's Guide*.

–o *outfile*
Specifies the name of the **c89/cc/c++** output file.

If the **–o** option is specified in addition to the **–c** option, and only one source file is specified, then this option specifies the name of the output file associated with the one source file. See *file.o* under "Operands" on page 72 for information on the default name of the output file.

Otherwise the **–o** option specifies the name of the executable file produced during the link-editing phase. The default output file is **a.out**. For related information, see Usage Note 3 on page 87.

–p Ignored by **cc**, provided for compatibility with historical implementations of **cc**. Flagged as an error by **c89** and **c++**.

–q Ignored by **cc**, provided for compatibility with historical implementations of **cc**. Flagged as an error by **c89** and **c++**.

–r Specifies that **c89/cc/c++** is to save relocation information about the object files which are processed. When the output file (as specified on **–o**) is created, it is not made an executable file. Instead, this output file can later be used as input to **c89/cc/c++**. This can be used as an alternative to an archive library.

–s Specifies that the compilation phase is to produce a *file.o* file that does *not* include symbolic information, and that the link-editing phase produce an executable that is marked reentrant.. This is the default behavior for **c89/cc/c++**.

–U *name*
Undefines a C or C++ macro specified with *name*. This option affects only macros defined by the **–D** option, including those automatically specified by **c89/cc/c++**. For information about macros that **c89/cc/c++** automatically define, see Usage Note 5 on page 88. Also see Usage Note 13 on page 90.

–u *function*
Specifies the name of the function to added to the list of symbols which are not yet defined. This can be useful if the only input to **c89/cc/c++** is archive libraries. Non–C++ linkage symbols of up to 255 characters in length may be specified. You can specify an S-name by preceding the function name with double slash (//). (For more information about S-names, see Usage Note 23 on page 92.)The function **//ceemain** is the default. However if this option is used, then the default function is not added to the list.

–V This verbose option produces and directs output to **stdout** as compiler, assembler, IPA linker, prelinker, and link editor listings. If the **–O** or **–2** options are specified and cause **c89/cc/c++** to use the compiler **INLINE** option, then

the inline report is also produced with the compiler listing. Error output continues to be directed to **stderr**. Because this option causes **c89/cc/c++** to change the options passed to the steps producing these listings so that they produce more information, it may also result in additional messages being directed to **stderr**. In the case of the compile step, it may also result in the return code of the compiler changing from 0 to 4.

–v This verbose option causes pseudo-JCL to be written to **stdout** before the compiler, assembler, IPA linker, prelinker, and link editor programs are run. It provides information about exactly which compiler, prelinker, and link editor options are being passed, and also which data sets are being used. If you want to obtain this information without actually invoking the underlying programs, specify the **–v** option more than once on the **c89/cc/c++** command string. For more information about the programs which are executed, see Usage Note 14 on page 90.

–W *phase, option[,option]...*

Specifies options to be passed to the steps associated with the compile, assemble, or link-editing phases of **c89/cc/c++**. The valid phase codes are:

0 Specifies the compile phase (used for both non-IPA and IPA compilation).

a Specifies the assemble phase.

c Same as phase code **0**.

I Enables IPA (Interprocedural Analysis) optimization.

l Specifies the link-editing phase.

- To pass options to the prelinker, the first link-editing phase option must be **p** or **P**. All the following options are then prelink options. For example, to write the prelink map to **stdout**, specify:

  ```
  c89 –W l,p,map file.c
  ```

  Note: The prelinker is no longer used in the link-editing phase in most circumstances. If it is not used, any options passed are accepted but ignored. See the environment variable **{_STEPS}** under “Environment Variables” on page 74 for more information about the link-editing phase prelink step.

- To pass options to the IPA linker, the first link-editing phase option must be **i** or **I**. All the following options are then IPA link options. For example, to write the IPA link map to **stdout**, specify:

  ```
  c89 –W l,I,map file.c
  ```

- To link-edit a DLL (Dynamic Link Library) the link-editing phase option DLL must be specified. For example:

  ```
  c89 –o outdll –W l,dll file.o
  ```

Most OS/390 C/C++ extensions can be enabled by using this option. Those which do not directly pass options through to the underlying steps, or involve files which are extensions to the compile and link-edit model, are described here:

DLL (Dynamic Link Library)

A DLL is a part of a program that is not statically bound to the program. Instead, linkage to symbols (variables and functions) is completed dynamically at execution time. DLLs can improve storage utilization, because the program can be broken into smaller parts, and some parts

may not always need to be loaded. DLLs can improve maintainability, because the individual parts can be managed and serviced separately.

In order to create a DLL, some symbols must be identified as being exported for use by other parts of the program. This can be done with the OS/390 C/C++ **#pragma export** compiler directive, or by using the OS/390 C/C++ **EXPORTALL** compiler option. If during the link-editing phase some of the parts have exported symbols, the executable which is created is a DLL. In addition to the DLL, a definition side-deck is created, containing link-editing phase **IMPORT** control statements which name those symbols which were exported by the DLL. In order for the definition side-deck to be created, the **DLL** link editor option must be specified. This definition side-deck is subsequently used during the link-editing phase of a program which is to use the DLL. See the *file.x* operand under Operands on page 74 for information on where the definition side-deck is written. In order for the program to refer to symbols exported by the DLL, it must be compiled with the **DLL** compiler option. For example, to compile and link a program into a DLL, you could specify:

```
c89 –o outdll –W c,dll,expo –W l,dll file.c
```

To subsequently use *file.x* definition side-decks, specify them along with any other *file.o* object files specified for **c89/cc/c++** link-editing phase. For example:

```
c89 –o myappl –W c,dll myappl.c outdll.x
```

In order to run an application which is link-edited with a definition side-deck, the DLL must be made available (the definition side-deck created along with the DLL is not needed at execution time). When the DLL resides in the HFS, it must be in either the working directory or in a directory named on the **LIBPATH** environment variable. Otherwise it must be a member of a data set in the search order used for MVS programs.

SOM (System Object Model)

System Object Model (SOM) provides a common programming interface with which you can build and use objects. SOM improves your C++ programming productivity.

To compile a C++ source file using implicit **SOM** mode, either the OS/390 C++ compiler **SOM** option, or the OS/390 C++ compiler directive **#pragma SOMAsDefault** must be used. Also, the SOM header libraries must be specified for the **c++** command invocation. You can pass the **SOM** option to the OS/390 C/C++ compiler by specifying the **c++ –W** option, and also specify the SOM header libraries, as in this example:

```
c++ –I "//'sommvs.sgoshh.+'" \
–I "//'sommvs.sgoshh.+'" –W c,som file.C
```

When creating a SOM application, during the **c++** link-editing phase you will need to specify either SOM C370LIB object libraries (for static linking) or SOM definition side-deck library members (for dynamic linking). If when using dynamic linking, the SOM definition side-deck library does not use the suffix *EXP*, it is recommended that you export the **{_EXTRA_ARGS}** environment variable. For example, to use dynamic linking, you might specify:

```
_CXX_EXTRA_ARGS=1 c++ \
"//'sommvs.sgosimp(gossomk,gossomu)'" file.o
```

IPA (interprocedural analysis)

IPA optimization is independent from and can be used in addition to the **c89/cc/c++** optimization level options (such as **–O**). IPA optimization can also improve the execution time of your application. IPA is a mechanism for performing optimizations across function boundaries, even across compilation units. It also performs optimizations not otherwise available with the C/C++ compiler.

When phase code **I** is specified for the compilation phase, then IPA compilation steps are performed. When phase code **I** is specified for the link-editing phase, or when the first link-editing phase (code **l**) option is **i** or **I**, then an additional IPA link step is performed prior to the prelink and link-edit steps.

With conventional compilation and link-editing, the object code generation takes place during the compilation phase. With IPA compilation and link-editing, the object code generation takes place during the link-editing phase. Therefore, you might need to request listing information about the program (such as with the **–V** option) during the link-editing phase.

Unlike the other phase codes, phase code **I** does not require that any additional options be specified. If they are, they should be specified for both the compilation and link-editing phases.

No additional preparation needs to be done in order to use IPA. So for example to create the executable **myIPApgm** using **c89** with some existing source program **mypgm.c**, you could specify:

```
c89 –W I –o myIPApgm mypgm.c
```

You cannot use **–W** to override the compiler options that correspond to **c89/cc/c++** options, with the following exceptions:

- Listing options (corresponding to **–V**)
- Inlining options (corresponding to **–O** and **–2**)
- Symbolic options (corresponding to **–s** and **–g**); symbolic options can be overridden only when neither **–s** nor **–g** is specified.

Notes:

1. Most compiler, prelinker, and IPA linker options have a positive and negative form. The negative form is the positive with a prepended NO (as in XREF and NOXREF).
2. The compiler **#pragma options** directives as well as any other **#pragma** directives which are overridden by compiler options, will have no effect in source code compiled by **c89/cc/c++**.
3. Link editor options must be specified in the *name=value* format. Both the option *name* and *value* must be spelled out in full. If you do not specify a value, a default value of YES is used, except for the following options, which if specified without a value, have the default values shown here:

   LIST LIST=NOIMPORT

   ALIASES ALIASES=ALL

COMPAT COMPAT=CURRENT

DYNAM DYNAM=DLL

Note: References thoughout this document to the link editor are generic references. **c89/cc/c++** specifically uses the Program Management binder for this function.

4. The OS/390 C/C++ compiler is described in *OS/390 C/C++ User's Guide*. Related information about the OS/390 C/C++ runtime library, including information about DLL, SOM, and IPA support, is described in *OS/390 C/C++ Programming Guide*. Related information about the OS/390 C and OS/390 C++ languages, including information about compiler directives, is described in *OS/390 C/C++ Language Reference*.
5. Since some compiler options are OS/390 C–only and some compiler options are OS/390 C++–only, you may get warning messages and a compiler return code of 4, if you use this option and compile both C and C++ source programs in the same **c++** command invocation.
6. For more information about SOM, see *OS/390 SOMobjects Programmer's Guide*.
7. The prelinker is described in *OS/390 C/C++ User's Guide*.
8. The link editor (the Program Management binder) is described in *DFSMS/MVS Program Management*. The Program Management binder messages are described in *OS/390 MVS System Messages, Vol 4 (IEC-IFD)*.

Operands

c89/cc/c++ generally recognize their file operand types by file suffixes. The suffixes shown here represent the default values used by **c89/cc/c++**. See “Environment Variables” on page 74 for information on changing the suffixes to be used.

Unlike **c89** and **c++**, which report an error if given an operand with an unrecognized suffix, **cc** determines that it is either an object file or a library based on the file itself. This behavior is in accordance with the environment variable **{_EXTRA_ARGS}**. Also see Usage Note 3 on page 87 for related information.

file.a
Specifies the name of an archive file, as produced by the **ar** command, to be used during the link-editing phase. You can specify an MVS data set name, by preceding the file name with double slash (//), in which case the last qualifier of the data set name must be *LIB*. The data set specified must be a C370LIB object library or a load library. See the description of the **–l** *libname* operand for more information about using data sets as libraries.

file.C
Specifies the name of a C++ source file to be compiled. You can specify an MVS data set name by preceding the file name with double slash (//), in which case the last qualifier of the data set name must be *CXX*. This operand is only supported by the **c++** command.

file.c
Specifies the name of a C source file to be compiled. You can specify an MVS data set name by preceding the file name with double slash (//), in which case the last qualifier of the data set name must be *C*. (The conventions formerly used by **c89** for specifying data set names are still sup-

ported. See the environment variables **{_OSUFFIX_HOSTRULE}** and **{_OSUFFIX_HOSTQUAL}** for more information.)

file.I Specifies the name of a IPA linker output file produced during the **c89/cc/c++** link-editing phase, when the **–W** option is specified with phase code **I**. IPA is further described under the **–W** option on page 71. By default the IPA linker output file is written to a temporary file. To have the IPA linker output file written to a permanent file, see the environment variable **{_TMPS}** under Environment Variables.

When an IPA linker output file is produced by **c89/cc/c++**, the default name is based upon the output file name. See the **–o** option under Options on page 68, for information on the name of the output file.

If the output file is named *a.out*, then the IPA linker output file is named *a.I*, and is always in the working directory. If the output file is named *//a.load*, then the IPA linker output file is named *//a.IPA*. If the output file specified already has a suffix, that suffix is replaced. Otherwise the suffix is appended.

file.i Specifies the name of a preprocessed C or C++ source file to be compiled. You can specify an MVS data set name, by preceding the file name with double slash (//), in which case the last qualifier of the data set name must be *CEX*.

When using the **c++** command, this source file is recognized as a C++ source file, otherwise it is recognized as a C source file. **c++** can be made to distinguish between the two. For more information see the environment variables **{_IXXSUFFIX}** and **{_IXXSUFFIX_HOST}**.

file.o

Specifies the name of a C, C++, or assembler object file, produced by **c89/cc/c++**, to be link-edited.

When an object file is produced by **c89/cc/c++**, the default name is based upon the source file. If the source file is named *file.c*, then the object file is named *file.o*, and is always in the working directory. If the source file were a data set named *//file.C*, then the object file is named *//file.OBJ*.

If the data set specified as an object file has undefined (U) record format, then it is assumed to be a load module. Load modules are not processed by the prelinker.

You can specify an MVS data set name to be link-edited, by preceding the file name with double slash (//), in which case the last qualifier of the data set name must be *OBJ*. If a partitioned data set is specified, more than one member name may be specified by separating each with a comma (,), for example:

```
c89 //file.OBJ(mem1,mem2,mem3)
```

file.p

Specifies the name of a prelinker composite object file produced during the **c89/cc/c++** link-editing phase. By default the composite object file is written to a temporary file. To have the composite object file written to a permanent file, see the environment variable **{_TMPS}** under Environment Variables.

When a composite object file is produced by **c89/cc/c++**, the default name is based upon the output file name. See the **–o** option under Options on page 68, for information on the name of the output file.

If the output file is named *a.out*, then the composite object file is named *a.p*, and is always in the working directory. If the output file is named *//a.load*, then the composite object file is named *//a.CPOBJ*. If the output file specified already has a suffix, that suffix is replaced. Otherwise the suffix is appended.

file.s

Specifies the name of an assembler source file to be assembled. You can specify an MVS data set name, by preceding the file name with double slash (//), in which case the last qualifier of the data set name must be *ASM*.

file.x

Specifies the name of a definition side-deck produced during the **c89/cc/c++** link-editing phase when creating a DLL (Dynamic Link Library), and used during the link-editing phase of an application using the DLL. DLLs are further described under the **–W** option.

When a definition side-deck is produced by **c89/cc/c++**, the default name is based upon the output file name. See the **–o** option under Options on page 68, for information on the name of the output file.

If the output file is named *a.dll*, then the definition side-deck is named *a.x*, and is always in the working directory. If the output file is named *//a.DLL*, then the definition side-deck is named *//a.EXP*. If the output file specified already has a suffix, that suffix is replaced. Otherwise the suffix is appended.

You can specify an MVS data set name to be link-edited, by preceding the file name with double slash (//), in which case the last qualifier of the data set name must be *EXP*. If a partitioned data set is specified, more than one member name may be specified by separating each with a comma (,), for example:

```
c89 //file.EXP(mem1,mem2,mem3)
```

–l *libname*

Specifies the name of an archive library. **c89/cc/c++** searches for the file **lib***libname***.a** in the directories specified on the **–L** option and then in the usual places. The first occurrence of the archive library is used. For a description of the usual places, see Usage Note 7 on page 89.

You can also specify an MVS data set; you must specify the full data set name, because there are no rules for searching library directories.

The data set specified must be a C370LIB object library or a load library. If a data set specified as a library has undefined (U) record format, then it is assumed to be a load library. For more information about the OS/390 C/C++ Object Library Utility, *OS/390 C/C++ Programming Guide*. For more information about how load libraries are searched, see Usage Note 7 on page 89.

Environment Variables

You can use environment variables to specify necessary system and operational information to **c89/cc/c++**. When a particular environment variable is not set, **c89/cc/c++** uses the default shown. For information about the JCL parameters used in these environment variables, see *OS/390 MVS JCL User's Guide*.

Each environment variable name is shown with the prefix omitted. To specify **cc** environment variables, the name shown must be prefixed with **_CC** (for example, **_CC_ACCEPTABLE_RC**). To specify **c89** environment variables, the name shown must be prefixed with **_C89** (for example, **_C89_ACCEPTABLE_RC**). To specify

c++ environment variables, the name shown must be prefixed with **_CXX** (for example, **_CXX_ACCEPTABLE_RC**).

Note: **c89/cc/c++** can accept parameters only in the syntax indicated here. A null values indicate that **c89/cc/c++** is to omit the corresponding parameters during dynamic allocation. Numbers in parentheses following the environment variable name correspond to usage notes, which begin on Page 87, and indicate specific usage information for the environment variable.

{_ACCEPTABLE_RC}

The maximum allowed return code (result) of any step (compile, assemble, IPA link, prelink, or link-edit). If the result is between zero and this value (inclusive), then it is treated internally by **c89/cc/c++** exactly as if it were a zero result, except that message FSUM3065 is also issued. The default value is:

`"4"`

When used under **c89/cc/c++**, the prelinker by default returns at least a `4` when there are duplicate symbols or unresolved writable static symbols (but not for other unresolved references). The link editor returns at least a `4` when there are duplicate symbols, and at least an `8` when there are unresolved references and automatic library call was used.

{_ASUFFIX} (15)

The suffix by which **c89/cc/c++** recognizes an archive file. This environment variable does not affect the treatment of archive libraries specified as **–l** operands, which are always prefixed with *lib* and suffixed with *.a*. The default value is:

`"a"`

{_ASUFFIX_HOST} (15)

The suffix by which **c89/cc/c++**recognizes a library data set. This environment variable does not affect the treatment of data set libraries specified as **–l** operands, which are always used exactly as specified. The default value is:

`"LIB"`

{_CCMODE}

Controls how **c89/cc/c++** does parsing. The default behavior of **c89/cc/c++** is to expect all options to precede all operands. Setting this variable allows compatibility with historical implementations (other **cc** commands). When set to 1, **c89/cc/c++** operates as follows:

- Options and operands can be interspersed.
- The double dash (—) is ignored.

Setting this variable to `0` results in the default behavior. The default value is:

`"0"`

{_CLIB_PREFIX} (14 ,17)

The prefix for the following named data sets used during the compilation phase and execution of your application.

To be used, the following data sets must be cataloged:

- The data sets {_CLIB_PREFIX}.SCLBH.+ contain the OS/390 C/C++ Class Library include (header) files.
- The data set {_CLIB_PREFIX}.SCLBSID contains the OS/390 C/C++ Class Library definition side-decks.

The following data sets are also used:

- The data set {_CLIB_PREFIX}.SCBCCMP contains the compiler programs called by **c89/cc/c++**.
- The data set {_CLIB_PREFIX}.SCLBDLL contains the OS/390 C/C++ Class Library DLLs and messages.

The preceding data sets contain MVS programs that are invoked during the execution of **c89/cc/c++** and during the execution of a C/C++ application built by **c89/cc/c++**. To be executed correctly, these data sets must be made part of the MVS search order. Regardless of the setting of this or any other **c89/cc/c++** environment variable, **c89/cc/c++** does not affect the MVS search order. These data sets are listed here for information only, to assist in identifying the correct data sets to be added to the MVS program search order.

The default value is:

```
"CBC"
```

{_CMEMORY}
A suggestion as to the use of compiler C/C++ Runtime Library memory files. When set to 0, **c89/cc/c++** will prefer to use the compiler **NOMEMORY** option. When set to 1, **c89/cc/c++** will prefer to use the compiler **MEMORY** option. When set to 1, and if the compiler **MEMORY** option can be used, **c89/cc/c++** need not allocate data sets for the corresponding work files. In this case it is the responsibility of the user to not override the compiler options (using the **–W** option) with the **NOMEMORY** option or any other compiler option which implies the **NOMEMORY** option.

The default value is:

```
"1"
```

{_CMSGS} (14)
The Language Environment national language name used by the compiler program. A null value will cause the default Language Environment **NATLANG** runtime name to be used, and a non-null value must be a valid Language Environment **NATLANG** runtime option name (Language Environment runtime options are described in *OS/390 Language Environment Programming Guide* . The default value is:

```
"" (null)
```

{_CNAME} (14)
The name of the compiler program called by **c89/cc/c++**. It must be a member of a data set in the search order used for MVS programs. The default value is:

```
"CBCDRVR"
```

{_CSUFFIX} (15)
The suffix by which **c89/cc/c++** recognizes a C source file. The default value is:

```
"c"
```

{_CSUFFIX_HOST} (15)
The suffix by which **c89/cc/c++** recognizes a C source data set. The default value is:

```
"C"
```

{_CXXSUFFIX} (15)
The suffix by which **c++** recognizes a C++ source file. The default value is:

```
"C"
```

This environment variable is only supported by the **c++** command.

{_CXXSUFFIX_HOST} (15)
The suffix by which **c++** recognizes a C++ source data set. The default value is:

```
"CXX"
```

This environment variable is only supported by the **c++** command.

{_CSYSLIB} (4 , 16)
The system library data set concatenation to be used to resolve *#include* directives during compilation.

Normally *#include* directives are resolved using all the information specified including the directory name. When **c89/cc/c++** can determine that the directory information can be used, such as when the include (header) files provided by Language Environment are installed in the default location (in accordance with **{_INCDIRS}**), then the default concatenation is:

```
"" (null)
```

When **c89/cc/c++** cannot determine that the directory information can be used, then the default concatenation is:

```
"{_PLIB_PREFIX}.SCEEH.H"
"{_PLIB_PREFIX}.SCEEH.SYS.H"
"{_PLIB_PREFIX}.SCEEH.ARPA.H"
"{_PLIB_PREFIX}.SCEEH.NET.H"
"{_PLIB_PREFIX}.SCEEH.NETINET.H"
```

When this variable is a null value, then no allocation is done for compiler system library data sets. In this case, the use of //DD:SYSLIB on the **–I** option and the *#include* directive will be unsuccessful. Unless there is a dependency on the use of //DD:SYSLIB, it is recommended that for improved performance this variable be allowed to default to a null value.

{_DAMPLEVEL}
The minimum severity level of dynamic allocation messages returned by dynamic allocation message processing. Messages with severity greater than or equal to this number are written to **stderr.** However, if the number is out of the range shown here (that is, less than 0 or greater than 8), then **c89/cc/c++** dynamic allocation message processing is disabled. The default value is:

```
"4"
```

Following are the values:

0 Informational
1–4 Warning
5–8 Severe

{_DAMPNAME} (14)
The name of the dynamic allocation message processing program called by **c89/cc/c++**. It must be a member of a data set in the search order used for MVS programs. The default dynamic allocation message processing program is described in *OS/390 MVS Programming: Authorized Assembler Services Guide*. The default value is:

"IEFDB476"

{_DCBF2008} (21)
The DCB parameters used by **c89/cc/c++** for data sets with the attributes of record format fixed unblocked and minimum block size of 2008. The block size must be in multiples of 8, and the maximum depends on the phase in which it is used but can be at least 5100. The default value is:

"(RECFM=F,LRECL=4088,BLKSIZE=4088)"

{_DCBU} (21)
The DCB parameters used by **c89/cc/c++** for data sets with the attributes of record format undefined and data set organization partitioned. This DCB is used by **c89/cc/c++** for the output file when it is to be written to a data set. The default value is:

"(RECFM=U,LRECL=0,BLKSIZE=6144,DSORG=PO)"

{_DCB121M} (21)
The DCB parameters used by **c89/cc/c++** for data sets with the attributes of record format fixed blocked and logical record length 121, for data sets whose records may contain machine control characters. The default value is:

"(RECFM=FBM,LRECL=121,BLKSIZE=3630)"

{_DCB133M} (21)
The DCB parameters used by **c89/cc/c++** for data sets with the attributes of record format fixed blocked and logical record length 133, for data sets whose records may contain machine control characters. The default value is:

"(RECFM=FBM,LRECL=133,BLKSIZE=3990)"

{_DCB137} (21)
The DCB parameters used by **c89/cc/c++** for data sets with the attributes of record format variable blocked and logical record length 137. The default value is:

"(RECFM=VB,LRECL=137,BLKSIZE=882)"

{_DCB137A} (21)
The DCB parameters used by **c89/cc/c++** for data sets with the attributes of record format variable blocked and logical record length 137, for data sets whose records may contain ISO/ANSI control characters. The default value is:

"(RECFM=VB,LRECL=137,BLKSIZE=882)"

{_DCB3200} (21)
The DCB parameters used by **c89/cc/c++** for data sets with the attributes of record format fixed blocked and logical record length 3200. The default value is:

"(RECFM=FB,LRECL=3200,BLKSIZE=12800)"

{_DCB80} (21)
The DCB parameters used by **c89/cc/c++** for data sets with the attributes of record format fixed blocked and logical record length 80. This value is also used when **c89/cc/c++** allocates a new data set for an object file. The default value is:

"(RECFM=FB,LRECL=80,BLKSIZE=3200)"

{_ELINES}
This variable controls whether the output of the **-E** option will include **#line** directives. **#line** directives provide information about the source file names

and line numbers from which the preprocessed source came. The preprocessor only inserts **#line** directives where it is necessary. When set to 1, the output of the **c89/cc/c++ -E** option will include **#line** directives where necessary. When set to `0`, the output will not include any **#line** directives. The default value is:

`"0"`

{_EXTRA_ARGS}
The setting of this variable controls whether **c89/cc/c++** treats a file operand with an unrecognized suffix as an error, or attempts to process it. When the **c++** command **–+** option is specified, all suffixes which otherwise would be unrecognized are instead recognized as C++ source, effectively disabling this environment variable. See page 65 for information about the **–+** option.

When set to 0, **c89/cc/c++** treats such a file as an error and the command will be unsuccessful, because the suffix will not be recognized.

When set to 1, **c89/cc/c++** treats such a file as either an object file or a library, depending on the file itself. If it is neither an object file nor a library then the command will be unsuccessful, because the link-editing phase will be unable to process it. The default value for **c89** and **c++** is:

`"0"`

The default value for **cc** is:

`"1"`

{_ILCTL} (14)
The name of the control file used by the IPA linker program. By default the control file is not used, so the **–W** option must be specified to enable its use, as in:

```
c89 -WI,control ...
```

The default value is:

`"ipa.ctl"`

{_ILMSGS} (14)
The name of the message data set member, or the Language Environment national language name, used by the IPA linker program. The default value is whatever **{_CMSGS}** is. So if **{_CMSGS}** is set or defaults to `""` (null), the default value is:

`"" (null)`

{_ILNAME} (14)
The name of the IPA linker program called by **c89/cc**. It must be a member of a data set in the search order used for MVS programs. The default value is whatever **{_CNAME}** is. So if **{_CNAME}** is set or defaults to "CBCDRVR" the default value is:

`"CBCDRVR"`

{_ILSUFFIX} (15)
The suffix **c89/cc** uses when creating an IPA linker output file. The default value is:

`"I"`

{_ILSUFFIX_HOST} (15)
The suffix **c89/cc** uses when creating an IPA linker output data set. The default value is:

```
"IPA"
```

{_INCDIRS} (22)
The directories used by **c89/cc/c++** as a default place to search for include files during compilation (before searching **{_INCLIBS}** and **{_CSYSLIB}**). The default value is:

```
"/usr/include"
```

{_INCLIBS} (22)
The directories used by **c89/cc/c++** as a default place to search for include files during compilation (after searching **{_INCDIRS}** and before searching **{_CSYSLIB}**). The default value depends on whether or not **c++** is being used. If **c++** is not being used the default value is:

```
"//'{_PLIB_PREFIX}.SCEEH.+'"
```

If **c++** is being used, the default value is:

```
"//'{_PLIB_PREFIX}.SCEEH.+' //'{_CLIB_PREFIX}.SCLBH.+'"
```

{_ISUFFIX} (15)
The suffix by which **c89/cc/c++** recognizes a preprocessed C source file. The default value is:

```
"i"
```

{_ISUFFIX_HOST} (15)
The suffix by which **c89/cc/c++** recognizes a preprocessed (expanded) C source data set. The default value is:

```
"CEX"
```

{_IXXSUFFIX} (15)
The suffix by which **c++** recognizes a preprocessed C++ source file. The default value is:

```
"i"
```

This environment variable is only supported by the **c++** command.

{_IXXSUFFIX_HOST} (15)
The suffix by which **c++** recognizes a preprocessed (expanded) C++ source data set. The default value is:

```
"CEX"
```

This environment variable is only supported by the **c++** command.

{_LIBDIRS} (22)
The directories used by **c89/cc/c++** as the default place to search for archive libraries which are specified using the **–l** operand. The default value is:

```
"/lib /usr/lib"
```

{_LSYSLIB} (16)
The system library data set concatenation to be used to resolve symbols during the IPA link step and the link-edit step of the link-editing phase. The **{_PSYSLIB}** libraries always precede the **{_LSYSLIB}** libraries when resolving symbols in the link-editing phase. The default value is the concatenation:

```
"{_PLIB_PREFIX}.SCEELKEX"
"{_PLIB_PREFIX}.SCEELKED"
"{_SLIB_PREFIX}.CSSLIB"
```

{_MEMORY}
A suggestion as to the use of C/C++ Runtime Library memory files by **c89/cc/c++**. When set to 0, **c89/cc/c++** uses temporary data sets for all work files. When set to 1, **c89/cc/c++** uses memory files for all work files that it can. The default value is:

```
"1"
```

{_NEW_DATACLAS} (18)
The DATACLAS parameter used by **c89/cc/c++** for any new datasets it creates. The default value is:

```
"" (null)
```

{_NEW_DSNTYPE} (18 , 20)
The DSNTYPE parameter used by **c89/cc/c++** for any new data sets it creates. The default value is:

```
"" (null)
```

{_NEW_MGMTCLAS} (18)
The MGMTCLAS parameter used by **c89/cc/c++** for any new datasets it creates. The default value is:

```
"" (null)
```

{_NEW_SPACE} (18 , 19)
The SPACE parameters used by **c89/cc/c++** for any new data sets it creates. A value for the number of directory blocks should always be specified. When allocating a sequential data set, **c89/cc/c++** automatically ignores the specification. The default value is:

```
"(,(10,10,10))"
```

{_NEW_STORCLAS} (18)
The STORCLAS parameter used by **c89/cc/c++**for any new data sets it creates. The default value is:

```
"" (null)
```

{_NEW_UNIT} (18)
The UNIT parameter used by **c89/cc/c++** for any new data sets it creates. The default value is:

```
"" (null)
```

{_OPERANDS} (22)
These operands are parsed as if they were specified after all other operands on the **c89/cc/c++** command line. The default value is:

```
"" (null)
```

{_OPTIONS} (22)
These options are parsed as if they were specified before all other options on the **c89/cc/c++** command line. The default value is:

```
"" (null)
```

{_OSUFFIX} (15)
The suffix by which **c89/cc/c++** recognizes an object file. The default value is:

```
"o"
```

{_OSUFFIX_HOST} (15)
The suffix by which **c89/cc/c++** recognizes an object data set. The default value is:

```
"OBJ"
```

{_OSUFFIX_HOSTQUAL}
The data set name of an object data set is determined by the setting of this option. If it is set to 0, then the suffix **{_OSUFFIX_HOST}** is appended to the source data set name to produce the object data set name. If it is set to 1, then the suffix **{_OSUFFIX_HOST}** replaces the last qualifier of the source data set name to produce the object data set name (unless there is only a single qualifier, in which case the suffix is appended). The default value is:

```
"1"
```

Note: Earlier versions of **c89** always appended the suffix, which was inconsistent with the treatment of files in the hierarchical file system. It is recommended that any existing data sets be converted to use the new convention.

{_OSUFFIX_HOSTRULE}
The way in which suffixes are used for host data sets is determined by the setting of this option. If it is set to 0, then data set types are determined by the rule described in the note which follows. If it is set to 1, then the data set types are determined by last qualifier of the data set (just as a suffix is used to determine the type of hierarchical file system file). Each host file type has an environment variable by which the default suffix can be modified. The default value is:

```
"1"
```

Notes:

1. Earlier versions of **c89** scanned the data set name to determine if it was an object data set. It searched for the string **OBJ** in the data set name, exclusive of the first qualifier and the member name. If it was found, the data set was determined to be an object data set, and otherwise it was determined to be a C source data set. It is recommended that any existing data sets be converted to use the new convention. Also, because the earlier convention only provided for recognition of C source files, assembler source cannot be processed if it is used.
2. The **c++** command does not support this environment variable, as the earlier convention would not provide for recognition of both C++ and C source files. Therefore regardless of its setting, **c++** always behaves as if it is set to "1".

{_PLIB_PREFIX} (14 ,17)
The prefix for the following named data sets used during the compilation, assemble, and link-editing phases, and during the execution of your application.

To be used, the following data sets must be cataloged:

- The data sets {_PLIB_PREFIX}.SCEEH.+ contain the include (header) files for use with the runtime library functions (where + can be any of H, SYS.H, ARPA.H, NET.H, and NETINET.H).
- The data set {_PLIB_PREFIX}.SCEEMAC contains COPY and MACRO files to be used during assembly.

- The data sets {_PLIB_PREFIX}.SCEEOBJ and {_PLIB_PREFIX}.SCEECPP contain runtime library bindings which exploit constructed reentrancy, used during link-editing.
- The data set {_PLIB_PREFIX}.SCEELKEX contains C runtime library bindings which exploit L-names used during link-editing. For more information about L-names, see the usage note 23 on page 92.
- The data set {_PLIB_PREFIX}.SCEELKED contains all other Language Environment runtime library bindings, used during link-editing.

The following data set is also used:

- The data set {_PLIB_PREFIX}.SCEERUN contains the runtime library programs.

The above data sets contain MVS programs that are invoked during the execution of **c89/cc/c++** and during the execution of a C/C++ application built by **c89/cc/c++**. To be executed correctly, these data sets must be made part of the MVS search order. Regardless of the setting of this or any other **c89/cc/c++** environment variable, **c89/cc/c++** does not affect the MVS program search order. These data sets are listed here for information only, to assist in identifying the correct data sets to be added to the MVS program search order. The default value is:

```
"CEE"
```

{_PMEMORY}
A suggestion as to the use of prelinker C/C++ Runtime Library memory files. When set to 0, **c89/cc/c++** uses the prelinker **NOMEMORY** option. When set to 1, **c89/cc/c++** uses the prelinker **MEMORY** option. The default value is:

```
"1"
```

{_PMSGS} (14)
The name of the message data set used by the prelinker program. It must be a member of the cataloged data set {_PLIB_PREFIX}.SCEEMSGP. The default value is:

```
 "EDCPMSGE"
```

{_PNAME} (14)
The name of the prelinker program called by **c89/cc/c++**. It must be a member of a data set in the search order used for MVS programs. The prelinker program is shipped as a member of the {_PLIB_PREFIX}.SCEERUN data set. The default value is:

```
"EDCPRLK"
```

{_PSUFFIX} (15)
The suffix **c89/cc/c++** uses when creating a prelinker (composite object) output file. The default value is:

```
"p"
```

{_PSUFFIX_HOST} (15)
The suffix **c89/cc/c++** uses when creating a prelinker (composite object) output data set. The default value is:

```
"CPOBJ"
```

{_PSYSIX} (16)
The system definition side-deck list to be used to resolve symbols during the link-editing phase. A definition side-deck contains link-editing phase **IMPORT**

control statements naming symbols which are exported by a DLL. The default value when **c++** is being used is the list containing the following single entry. Otherwise, the default value is null:

```
"{_CLIB_PREFIX}.SCLBSID(ASCCOLL,COMPLEX,IOSTREAM)"
```

{_PSYSLIB} (16)
The system library data set list to be used to resolve symbols during the link-editing phase. The **{_PSYSLIB}** libraries always precede the **{_LSYSLIB}** libraries when resolving symbols in the link-editing phase. The default value depends on whether or not **c++** is being used. If **c++** is not being used, the default value is the list containing the single entry:

```
"{_PLIB_PREFIX}.SCEEOBJ"
```

If **c++** is being used, the default value is the list:

```
"{_PLIB_PREFIX}.SCEEOBJ"
"{_PLIB_PREFIX}.SCEECPP"
```

{_SLIB_PREFIX} (17)
The prefix for the named data sets used by the link editor (CSSLIB) and the assembler system library data sets (MACLIB and MODGEN). The data set {_SLIB_PREFIX}.CSSLIB contains the OS/390 UNIX assembler callable services bindings. The data sets {_SLIB_PREFIX}.MACLIB and {_SLIB_PREFIX}.MODGEN contain COPY and MACRO files to be used during assembly. These data sets must be cataloged to be used. The default value is:

```
"SYS1"
```

{_SNAME} (14)
The name of the assembler program called by **c89/cc/c++**. It must be a member of a data set in the search order used for MVS programs. The default value is:

```
"ASMA90"
```

{_SSUFFIX} (15)
The suffix by which **c89/cc/c++** recognizes an assembler source file. The default value is:

```
"s"
```

{_SSUFFIX_HOST} (15)
The suffix by which **c89/cc/c++** recognizes an assembler source data set. The default value is:

```
"ASM"
```

{_SSYSLIB} (16)
The system library data set concatenation to be used to find COPY and MACRO files during assembly. The default concatenation is:

```
"{_PLIB_PREFIX}.SCEEMAC"
"{_SLIB_PREFIX}.MACLIB"
"{_SLIB_PREFIX}.MODGEN"
```

{_STEPS}
The steps that are executed for the link-editing phase can be controlled with this variable. For example, the prelinker step can be enabled, so that the inputs normally destined for the link editor instead go into the prelinker, and then the output of the prelinker becomes the input to the link editor.

This variable allows the prelinker to be used in order to produce output which is compatible with previous releases of **c89/cc/c++**. The prelinker is normally used by **c89/cc/c++** when the output file is a data set which is not a PDSE (partitioned data set extended).

The format of this variable is a set of binary switches which either enable (when turned on) or disable (when turned off) the corresponding step. Turning a switch on will not cause a step to be enabled if it was not already determined by **c89/cc/c++** that any other conditions necessary for its use are satisfied. For example, the IPA link step will not be executed unless the **–W** option is specified to enable the IPA linker. Enabling the IPA linker is described under the **–W** option on page 71.

Considering this variable to be a set of 32 switches, numbered left-to-right from 0 to 31, the steps corresponding to each of the switches are as follows:

0-27	Reserved
28	TEMPINC
29	IPALINK
30	PRELINK
31	LINKEDIT

For example, to override the default behavior of **c89/cc/c++** and cause the prelinker step to be run (this is also the default when the output file is a data set which is not a PDSE), set this variable to:

```
"0xffffffff"
```

The default value when the output file is an HFS file or a PDSE data set is:

```
"0xfffffffD"
```

{_SUSRLIB} (16)
The user library data set concatenation to be used to find COPY and MACRO files during assembly (before searching **{_SSYSLIB}**). The default value is:

```
"" (null)
```

{_TMPS}
The use of temporary files by **c89/cc/c++** can be controlledwith this variable.

The format of this variable is a set of binary switches which either cause a temporary file to be used (when turned on) or a permanent file to be used (when turned off) in the corresponding step.

The correspondence of these switches to steps is the same as for the variable **{_STEPS}**. Only the prelinker and IPA linker output can be captured using this variable.

For example, to capture the prelinker output, set this variable to:

```
"0xfffffffD"
```

The default value is :

```
"0xffffffff"
```

{_WORK_DATACLAS} (18)
The DATACLAS parameter used by **c89/cc/c++** for unnamed temporary (work) data sets. The default value is:

```
"" (null)
```

{_WORK_DSNTYPE} (18 , 20)
The DSNTYPE parameter used by **c89/cc/c++** for unnamed temporary (work) data sets. The default value is:

```
"" (null)
```

{_WORK_MGMTCLAS} (18)
The MGMTCLAS parameter used by **c89/cc/c++** for unnamed temporary (work) data sets. The default value is:

```
"" (null)
```

{_WORK_SPACE} (18 , 19)
The SPACE parameters used by **c89/cc/c++** for unnamed temporary (work) data sets. The default value is:

```
"(32000,(30,30))"
```

{_WORK_STORCLAS} (18)
The STORCLAS parameter used by **c89/cc/c++** for unnamed temporary (work) data sets. The default value is:

```
"" (null)
```

{_WORK_UNIT} (18)
The UNIT parameter used by **c89/cc/c++** for unnamed temporary (work) data sets. The default value is:

```
"SYSDA"
```

{_XSOMDIRS} (22)
The directories used by **c++** as a default place for include files which are to be e*x*cluded from implicit SOM mode during compilation (that is, when the OS/390 C/C++ compiler **SOM** option is specified). The directories specified on **{_XSOMDIRS}** precede those specified on **{_XSOMLIBS}**. The default value is whatever **{_INCDIRS}** is. So if **{_INCDIRS}** is set or defaults to "/usr/include", the default value is:

```
"/usr/include"
```

This environment variable is only supported by the **c++** command.

{_XSOMLIBS} (22)
The directories used by **c++** as a default place for includefiles which are to be excluded from implicit SOM mode during compilation (that is, when the OS/390 C/C++ compiler **SOM** option is specified). The directories specified on **{_XSOMLIBS}** follow those specified on **{_XSOMDIRS}**. The default value is whatever **{_INCLIBS}** is. So if **{_INCLIBS}** is set or defaults to

```
" //'{_PLIB_PREFIX}.SCEEH.+'
//'{_CLIB_PREFIX}.SCLBH.+'"
```

the default value is:

```
"//'{_PLIB_PREFIX}.SCEEH.+'
//'{_CLIB_PREFIX}.SCLBH.+'"
```

This environment variable is only supported by the **c++** command.

{_XSUFFIX} (15)
The suffix by which **c89/cc/c++** recognizes a definition side-deck file of e*x*ported symbols. The default value is:

```
"x"
```

{_XSUFFIX_HOST} (15)
The suffix by which **c89/cc/c++** recognizes a definition side-deck data set of e*x*ported symbols. The default value is:

```
"EXP"
```

Files

libc.a
C/C++ Runtime Library function library (see Usage Note 7 on page 89).

libm.a
C/C++ Runtime Library math function library (see Usage Note 7 on page 89).

libl.a
lex function library.

liby.a
yacc function library.

/dev/fd0,**/dev/fd1**, **...**
Character special files required by **c89/cc/c++**. For installation information, see *OS/390 UNIX System Services Planning*.

/usr/include
The usual place to search for include files (see Usage Note 4 on page 88).

/lib The usual place to search for runtime library bindings (see Usage Note 7 on page 89).

/usr/lib
The usual place to search for runtime library bindings (see Usage Note 7 on page 89).

Usage Notes

1. To be able to specify an operand that begins with a dash (–), before specifying any other operands that do not, you must use the double dash (—) end-of-options delimiter. This also applies to the specification of the **–l** operand. (See the description of environment variable {_CCMODE} for an alternate style of argument parsing.)
2. When invoking **c89/cc/c++** from the shell, any option-arguments or operands specified that contain characters with special meaning to the shell must be escaped. For example, some **–W** option-arguments contain parentheses. Source files specified as PDS member names contain parentheses; if they are specified as fully qualified names, they contain single quotes.

   To escape these special characters, either enclose the option-argument or operand in double quotes, or precede each character with a backslash.
3. Some **c89/cc/c++** behavior applies only to files (and not to data sets).
   - The **–o** option does not allow a source file (*file.C*, *file.c*, *file.i*, *file.s*) to be specified.
   - If the compile or assemble is not successful, the corresponding object file (*file.o*) is always removed.
   - If the DLL option is passed to the link-editing phase, and afterwards the *file.x* file exists but has a size of zero, then that file is removed.

4. MVS data sets may be used as the usual place to resolve C and C++ **#include** directives during compilation.

   Such data sets are installed with Language Environment. When it is allocated, searching for these include files can be specified on the **–I** option as *//DD:SYSLIB*. (See the description of environment variable **{_CSYSLIB}** for information.

   When include files are MVS PDS members, OS/390 C/C++ uses conversion rules to transform the include (header) file name on a **#include** preprocessor directive into a member name. If the "//'*dataset_prefix*.+'" syntax is not used for the MVS data set which is being searched for the include file, then this transformation strips any directory name on the **#include** directive, and then takes the first 8 or fewer characters up to the first dot (.).

   If the "//'*dataset_prefix*.+'" syntax is used for the MVS data set which is being searched for the include file, then this transformation uses any directory name on the **#include** directive, and the characters follwing the first dot (.), and substitutes the "+" of the dataset being searched with these qualifiers.

   In both cases the data set name and member name are converted to uppercase and underscores (_) are changed to at signs (@).

   If the include (header) files provided by Language Environment are installed into the hierarchical file system in the default location (in accordance with the **{_INCDIRS}** environment variable), then the compiler will use those files to resolve **#include** directives during compilation. **c89/cc/c++** by default searches the directory **/usr/include** as the usual place, just before searching the data sets just described. See the description of environment variables **{_CSYSLIB}**, **{_INCDIRS}**, and **{_INCLIBS}** for information on customizing the default directories to search.

5. Feature test macros control which symbols are made visible in a source file (typically a header file). **c89/cc/c++** automatically defines the following feature test macros along with the errno macro, according to whether or not **cc** was invoked.

   - Other than **cc**

     –D "errno=(*__errno())"
     –D _OPEN_DEFAULT=1

   - **cc**

     –D "errno=(*__errno())"
     –D _OPEN_DEFAULT=0
     –D _NO_PROTO=1

   c89/cc/c++add these macro definitions only after processing the command string. Therefore, you can override these macros by specifying **–D** or **–U** options for them on the command string.

6. The default LANGLVL and UPCONV compiler options are set according to whether or not **cc** was invoked. The LANGLVL compiler option affects OS/390 C/C++ predefined macros, which are used like feature test macros to control which symbols are made visible in a source file (typically a header file), but are not defined or undefined except by this compiler option. Together the LANGLVL and UPCONV options affect the language rules used by the compiler. For more information about these compiler options, see *OS/390 C/C++ User's Guide* . For more information about OS/390 C/C++ predefined macros, see *OS/390*

C/C++ Language Reference. The options are shown here in a syntax that the user can specify on the **c89/cc/c++** command line to override them:

- Other than **cc**
  –W "0,langlvl(ansi),noupconv"
- **cc**
  –W "0,langlvl(commonc),upconv"

7. By default the usual place for the **–L** option search is the **/lib** directory followed by the **/usr/lib** directory. See the description of environment variable **{_LIBDIRS}** for information on customizing the default directories to search.

   The archive libraries **libc.a** and **libm.a** exist as files in the usual place for consistency with other implementations. However, the runtime library bindings are not contained in them.

   Instead, MVS data sets installed with the Language Environment runtime library are used as the usual place to resolve runtime library bindings. In the final step of the link-editing phase, any MVS load libraries specified on the **–l** operand are searched in the order specified, followed by searching these data sets. See the description of environment variables **{_PLIB_PREFIX}** , **{_PSYSIX_PREFIX}**, **{_PSYSLIB_PREFIX}**, and **{_LSYSLIB_PREFIX}**,for information on customizing the default data sets to search.

8. Because archive library files are searched when their names are encountered, the placement of **–l** operands and *file.a* operands is significant. You may have to specify a library multiple times on the command string, if subsequent specification of *file.o* files requires that additional symbols be resolved from that library.

9. When the prelinker is used during the link-editing phase, you cannot use as input to **c89/cc/c++** an executable file produced as output from a previous use of **c89/cc/c++**. The output of **c89/cc/c++** when the **–r** option is specified (which is not an executable file) may be used as input.

10. All MVS data sets used by **c89/cc/c++** must be cataloged (including the system data sets installed with the OS/390 C/C++ compiler and the Language Environment runtime library).

11. **c89/cc/c++** operation depends on the correct setting of their installation and configuration environment variables (see “Environment Variables” on page 74). Also, they require that certain character special files are in the **/dev** directory. For additional installation and configuration information, see *OS/390 UNIX System Services Planning*.

12. Normally, options and operands are processed in the order read (from left to right). Where there are conflicts, the last specification is used (such as with **–g** and **–s**). However, some **c89/cc/c++** options will override others, regardless of the order in which they are specified. The option priorities, in order of highest to lowest, are as follows:

    –v specified twice
    : The pseudo-JCL is printed only, but the effect of all the other options and operands as specified is reflected in the pseudo-JCL.

    –E Overrides **–0**, **–O**, **–1**, **–2**, **–V**, **–c**, **–g** and **–s** (also ignores any *file.s* files).

    –g Overrides **–0**, **–O**, **–1**, **–2**, and **–s**.

    –s Overrides **–g** (the last one specified is honored).

–0, –O, –1, –2, –V, –c
All are honored if not overridden. **–0**, **–O**, **–1**, and **–2**, override each other (the last one specified is honored).

13. For options that have option-arguments, the meaning of multiple specifications of the options is as follows:

    –D All specifications are used. If the same name is specified on more than one **–D** option, only the first definition is used.

    –e The entry function used will be the one specified on the last **–e** option.

    –I All specifications are used. If the same directory is specified on more than one **–I** option, the directory is searched only the first time.

    –L All specifications are used. If the same directory is specified on more than one **–L** option, the directory is searched only the first time.

    –o The output file used will be the one specified on the last **–o** option.

    –U All specifications are used. The name is *not* defined, regardless of the position of this option relative to any **–D** option specifying the same name.

    –u All specifications are used. If a definition cannot be found for any of the functions specified, the link-editing phase will be unsuccessful.

    –W All specifications are used. All options specified for a phase are passed to it, as if they were concatenated together in the order specified.

14. The following environment variables can be at most eight characters in length. For those whose values specify the names of MVS programs to be executed, you can dynamically alter the search order used to find those programs by using the **STEPLIB** environment variable.

    c89/cc/c++ environment variables do not affect the MVS program search order. Also, for **c89/cc/c++** to work correctly, the setting of the **STEPLIB** environment variable should reflect the Language Environment library in use at the time that **c89/cc/c++** is invoked.

    For more information on the **STEPLIB** environment variable, see *OS/390 UNIX System Services Planning*. It is also described under the **sh** command. Note that the STEPLIB allocation in the pseudo-JCL produced by the **–v** verbose option is shown as a comment, and has no effect on the MVS program search order. Its appearance in the pseudo-JCL is strictly informational.

    {_CMSGS}
    {_CNAME}
    {_DAMPNAME}
    {_ILNAME}
    {_ILMSGS}
    {_PMSGS}
    {_PNAME}
    {_SNAME}

15. The following environment variables can be at most 15 characters in length. You should not specify any dots (`.`) when setting these environment variables since they would then never match their corresponding operands:

    {_ASUFFIX}
    {_ASUFFIX_HOST}
    {_CSUFFIX}
    {_CSUFFIX_HOST}

{_CXXSUFFIX}
{_CXXSUFFIX_HOST}
{_ISUFFIX}
{_ISUFFIX_HOST}
{_ILSUFFIX}
{_ILSUFFIX_HOST}
{_IXXSUFFIX}
{_IXXSUFFIX_HOST}
{_OSUFFIX}
{_OSUFFIX_HOST}
{_PSUFFIX}
{_PSUFFIX_HOST}
{_SSUFFIX}
{_SSUFFIX_HOST}
{_XSUFFIX}
{_XSUFFIX_HOST}

16. The following environment variables are parsed as colon-delimited data set names, and represent a data set concatenation or a data set list. The maximum length of each specification is 1024 characters:

    {_CSYSLIB}
    {_LSYSLIB}
    {_PSYSIX}
    {_PSYSLIB}
    {_SSYSLIB}
    {_SUSRLIB}

17. The following environment variables can be at most 44 characters in length:

    {_CLIB_PREFIX}
    {_PLIB_PREFIX}
    {_SLIB_PREFIX}

18. The following environment variables can be at most 63 characters in length:

    {_NEW_DATACLAS}
    {_NEW_DSNTYPE}
    {_NEW_MGMTCLAS}
    {_NEW_SPACE}
    {_NEW_STORCLAS}
    {_NEW_UNIT}
    {_WORK_DATACLAS}
    {_WORK_DSNTYPE}
    {_WORK_MGMTCLAS}
    {_WORK_SPACE}
    {_WORK_STORCLAS}
    {_WORK_UNIT}

19. The following environment variables are for specification of the SPACE parameter, and support only the syntax as shown with their default values (including all commas and parentheses). Also as shown with their default values, individual subparameters can be omitted, in which case the system defaults are used.

    {_NEW_SPACE}
    {_WORK_SPACE}

20. The following environment variables are for specification of the DSNTYPE parameter, and support only the subparameters LIBRARY or PDS (or null for no DSNTYPE):

    {_NEW_DSNTYPE}
    {_WORK_DSNTYPE}

21. The following environment variables can be at most 127 characters in length:

    {_DCBF2008}
    {_DCBU}
    {_DCB121M}
    {_DCB133M}
    {_DCB137}
    {_DCB137A}
    {_DCB3200}
    {_DCB80}

    These environment variables are for specification of DCB information, and support only the following DCB subparameters, with the noted restrictions:

    RECFM Incorrect values are ignored.
    LRECL None
    BLKSIZE None
    DSORG Incorrect values are treated as if no value had been specified.

22. The following environment variables are parsed as blank-delimited words, and therefore no embedded blanks or other white-space is allowed in the value specified. The maximum length of each word is 1024 characters:

    {_INCDIRS}
    {_INCLIBS}
    {_LIBDIRS}
    {_OPTIONS}
    {_OPERANDS}
    {_XSOMDIRS}
    {_XSOMLIBS}

23. An S-name is a *short* external symbol name, such as produced by the OS/390 C/C++ compiler when compiling OS/390 C programs with the `NOLONGNAME` option. An L-name is a *long* external symbol name, such as produced by the OS/390 C/C++ compiler when compiling OS/390 C programs with the `LONGNAME` option.

24. The C/C++ Runtime Library supports a file naming convention of // (the filename can begin with exactly two slashes). **c89/cc/c++** indicate that the file naming convention of // can be used.

    However, the Shell and Utilities feature *does not* support this convention. Do not use this convention (//) unless it is specifically indicated (as here in **c89/cc/c++**). The OS/390 Shell and Utilities feature does support the POSIX file naming convention where the filename can be selected from the set of character values excluding the slash and the null character.

Localization

c89/cc/c++ use the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

Exit Values

0 Successful completion.

1 Failure due to incorrect specification of the arguments.

2 Failure processing archive libraries:

- Archive library was not in any of the library directories specified.
- Archive library was incorrectly specified, or was not specified, following the **–l** operand.

3 Step of compilation, assemble, or link-editing phase was unsuccessful.

4 Dynamic allocation error, when preparing to call the compiler, assembler, IPA linker, prelinker, or link editor, for one of the following reasons:

- The file or data set name specified is incorrect.
- The file or data set name cannot be opened.

5 Dynamic allocation error, when preparing to call the compiler, assembler, prelinker, IPA linker, or link editor, due to an error being detected in the allocation information.

6 Error copying the file between a temporary data set and a hierarchical file system file (applies to the **–2** option, when processing assembler source files, and **–r** option processing).

7 Error creating a temporary control input data set for the link-editing phase.

8 Error creating a temporary system input data set for the compile or link-editing phase.

Portability

For **c89**, X/Open Portability Guide, POSIX.2 C-Language Development Utilities Option.

For **cc**, POSIX.2 C-Language Development Utilities Option, UNIX systems.

The following are extensions to the POSIX standard:

- The **—v**, **—V**, **—0**, **—1**, and **—2** options
- DLL support
- IPA optimization support
- The behavior of the **—o** option in combination with the **—c** option and a single source file.

Related Information

ar, **dbx**, **file**, **lex**, **make**, **nm**, **strings**, **strip**, **yacc**

---

cal — Display a calendar for a month or year

Format

cal [*month*] [*year*]

Description

cal displays a calendar on standard output (**stdout**).

- With no arguments, **cal** displays a calendar for the current month of the current year.
- If one argument is given and it is numeric, **cal** interprets it as a year (for example, 1991); if a single argument is not numeric, **cal** interprets it as the name of a month, possibly abbreviated (for example, apr).
- If two arguments are given, **cal** assumes that the first argument is the *month* (either a number from 1 to 12 or a month name) and the second is the *year*.

Localization

cal uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Usage Note

Year numbers less than 100 refer to the early Christian era, not the current century. This command prints the Gregorian calendar, handling September 1752 correctly. Many cultures observe other calendars.

Exit Values

`0` Successful completion.

`1` Failure due to any of the following:

- An incorrect command-line argument
- An incorrect date
- A year outside the range 1 to 9999 A.D.

Portability

X/Open Portability Guide, UNIX systems.

calendar — Display all current appointments

Format

calendar [–]

Description

If you do not specify any options, **calendar** displays all current appointments on standard output (**stdout**). It searches the file **calendar** in the current directory, looking for lines that match either today's date or tomorrow's date. On Friday, Saturday, or Sunday, *tomorrow* extends through to Monday. Each appointment must fit on a single line, with the date formatted as one of:

```
January 27
1/27
jan 27
```

Note: The name of the month can be abbreviated to three letters. Also, the case is not significant and the month can be given numerically.

Options

– Searches the RACF data base to find user IDs. **calendar** uses the **mailx** command (or, alternatively, the command named in the **MAILER** environment variable) to send mail to the corresponding user for any appointments that are found to be current. Because **calendar** cannot determine each user's locale, it runs in the POSIX locale when this option is used; otherwise it runs in the user's locale, processing data in single-byte mode.

Examples

If today is Friday April 7th and the following **calendar** file is found in the current directory:

```
tue mar 7 1:00 pm dentist
Sat April 8 Trip to the zoo
mon april 10 3:30 pm job interview
4/11  vacation starts
```

calendar prints the following:

```
Sat April 8 Trip to the zoo
mon april 10 3:30 pm job interview
```

Environment Variable

calendar uses the following environment variable:

MAILER Contains the name of the command that **calendar** uses to send mail. If this variable is not set, **calendar** uses **/bin/mail** as the default mail command.

Files

calendar uses the following file:

calendar File used in the current directory, or user's home directory.

Localization

calendar uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion

1 Failure due to any of the following:

- An incorrect command-line argument
- An inability to open the calendar file

Portability

X/Open Portability Guide, UNIX systems

The **MAILER** environment variable is an extension to traditional implementations of **calendar**.

Related Information

mailx

---

cancel — Cancel print queue requests (stub command)

Format

cancel [*print_ID ...*] *printer ...*
cancel *print_ID ...* [*printer ...*]

Description

cancel cancels print queue requests. *print_ID* specifies the particular job (or jobs) to be canceled; the *print_ID* number is reported by **lp** when the job is submitted, or by **lpstat**.

cancel is recognized, but its functions are not supported.

If you are using the OS/390 Print Server feature, your system automatically uses that version of the **cancel** command. For more information about the OS/390 Print Server commands, see *OS/390 Infoprint Server User's Guide*.

captoinfo — Print the terminal entries in the terminfo database

Format

captoinfo [**–1vV**] [**–w width**] [**file ...**]

Description

captoinfo prints all of the terminal entries in the terminfo database to standard output (**stdout**) in terminfo format. You can either look at the output or send it to a file that can be processed by **tic**.

The Curses application uses the terminfo database, which contains a list of terminal descriptions. This enables you to manipulate a terminal's display regardless of the terminal type. To create the terminfo database, use **tic**. For information on defining the terminfo database, see *OS/390 UNIX System Services Planning*.

For more information about curses, see *OS/390 C Curses*.

Options

1 Single-column output
V Print the program version
v Print debugging information (verbose) to standard error (**stderr**)
w Specifies the width of the output
filename
Specifies the termcap entries to be processed

Examples

1. This example shows how to print all the terminal entries in the file **/etc/termcap.src** in terminfo format. The entry for a vt52 is shown. Issue:

   ```
   captoinfo  /etc/termcap.src
   ```

   You get the following display:

   ```
   captoinfo: obsolete 2 character name 'dv' removed.
         synonyms are: 'vt52|dec vt52'
   #
   vt52|dec vt52,
         xon,
         cols#80, lines#24,
         bel=^¬, clear=-E-310-E-321, cub1=^½, cud1=^-,
         cuf1=-E-303,
         cup=-E-350%-227-361%'-s'%+%-203%-227-362%'-s'%+%-203,
         cuu1=-E-301, ed=-E-321, el=-E-322, ind=^-,
         kbs=^½, kcub1=-E-304, kcud1=-E-302, kcuf1=-E-303,
         kcuu1=-E-301, ri=-E-311,
   # END OF TERMCAP
   #  ------------------------
   ```

2. To print all the terminal entries in the file **/etc/termcap.src** in terminfo format with each entry on a separate line, issue:

   ```
   captoinfo  –1 /etc/termcap.src
   ```

   You get the following display:

```
captoinfo: obsolete 2 character name 'dv' removed.
      synonyms are: 'vt52|dec vt52'

#
vt52|dec vt52,
     xon,
     cols#80,
     lines#24,
     clear=-E-310-E-321,
     cub1=^½,
     cud1=^-,
     cuf1=-E-303,
     cup=-E-350%-227-361%'-s'%+%-203%-227-362%'-s'%+%-203,
     cuu1=-E-301,
     ed=-E-321,
     el=-E-322,
     ind=^-,
     kbs=^½,
     kcub1=-E-304,
     kcud1=-E-302,
     kcuf1=-E-303,
     kcuu1=-E-301,
     ri=-E-311,
# END OF TERMCAP
#  ------------------------
```

3. This example shows how to write all the terminal entries in the file **/etc/termcap.src** to the file **/test/terminfo.ti**. The resulting file can be processed by **tic**. Notice that the error messages are written to **stderr**.

   ```
   captoinfo  /etc/termcap.src  1> /test/terminfo.ti
   ```

   You get the following:

   ```
   captoinfo: obsolete 2 character name 'dv' removed.
         synonyms are: 'vt52|dec vt52'
   #
   ```

Related Information

infocmp, **tic**

---

cat — Concatenate or display text files

Format

cat [**–su**] [**–v** [**et**]] [*file ...*]

Description

cat displays and concatenates files. It copies each *file* argument to the standard output (**stdout**). If you specify no files or specify a dash (**–**) as a filename, **cat** reads the standard input (**stdin**).

Note: You can use **cat** in conjunction with the scrolling facility of the OMVS TSO/E command to browse data files.

Options

–e Displays a $ character at the end of each line. This option works only if you also specify **–v**.

–s Does not produce an error message if **cat** cannot find or read a specified file.

–t Displays tabs as ⅛I. This option works only if you also specify **–v**.

–u Does not buffer output.

–v Displays all characters including those that are unprintable characters. With a doublebyte character set, an unprintable wide character is converted back to its doublebyte representation. Each byte is then checked as if it were a singlebyte character. If the character is unprintable, one of the following three representations is used:

- M–X is used for character X if the significant bit is set.
- ⅛X is used for the control character X (for example, ⅛A for CTRL-A).
- \xxx represents a character with the octal value xxx.

The \xxx form is used if neither of the other representations can be used.

Localization

cat uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion

1 Failure due to any of the following:

- An incorrect command-line argument
- Inability to open the input file
- End of the file detected on **stdout**
- The input file is the same as the output file

2 An incorrect command-line argument

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The **–e**, **–s**, **–t**, and **–v** options are extensions of the POSIX standard.

Related Information

cp, **more**, **mv**

---

cc — Compile common usage source code and create an executable file

See **c89/cc/c++**.

---

cd — Change the working directory

Format

cd [*directory*]
cd *old new*
cd –

tcsh shell: **cd** [**-p**] [**-l**] [**-nl-v**] [*name*]

Description

The command **cd** *directory* changes the working directory of the current shell execution environment (see **sh**) to *directory*. If you specify *directory* as an absolute pathname, beginning with /, this is the target directory. **cd** assumes the target directory to be the name just as you specified it. If you specify *directory* as a relative pathname, **cd** assumes it to be relative to the current working directory.

If the variable **CDPATH** is defined in the shell, the built-in **cd** command searches for a relative pathname in each of the directories defined in **CDPATH**. If **cd** finds the directory outside the working directory, it displays the new working directory.

Use colons to separate directories in **CDPATH**. In **CDPATH**, a null string represents the working directory. For example, if the value of **CDPATH** begins with a separator character, **cd** searches the working directory first; if it ends with a separator character, **cd** searches the working directory last.

In the shell, the command **cd -** is a special case that changes the current working directory to the previous working directory by exchanging the values of the variables **PWD** and **OLDPWD**.

Note: Repeating this command toggles the current working directory between the current and the previous working directory.

Calling **cd** without arguments sets the working directory to the value of the **HOME** environment variable, if the variable exists. If there is no **HOME** variable, **cd** does not change the working directory.

The form **cd** *old new* is an extension to the POSIX standard and optionally to the Korn shell. The shell keeps the name of the working directory in the variable **PWD**. The **cd** command scans the current value of **PWD** and replaces the first occurrence of the string *old* with the string *new*. The shell displays the resulting value of **PWD**, and it becomes the new working directory.

If either directory is a symbolic link to another directory, the behavior depends on the setting of the shell's **–o** logical option. See the **set** command for more information.

cd in the tcsh shell

If a directory name is given, **cd** changes the tcsh shell's working directory to *name*. If not, it changes the directory to home. If *name* is '-' it is interpreted as the previous working directory. If *name* is not a subdirectory of the current directory (and does not begin with /, ./ or ../), each component of the tcsh variable **cdpath** is checked to see if it has a subdirectory name. Finally, if all else fails but *name* is a tcsh shell variable whose value begins with /, then this is tried to see if it is a directory (see also the **implicitcd** tcsh shell variable).

Options for the **cd** tcsh built-in command are:

-l Output is expanded explicitly to home or the pathname of the home directory for the user.

-n Entries are wrapped before they reach the edge of the screen.

-p Prints the final directory stack.

-v Entries are printed one per line, preceded by their stack postions.

If more than one of **-n** or **-v** is given, **-v** takes precedence. **-p** is accepted but does nothing.

Environment Variables

cd uses the following environment variables:

CDPATH
Contains a list of directories for **cd** to search in when *directory* is a relative pathname.

HOME Contains the name of your home directory. This is used when you do not specify *directory* on the command line.

OLDPWD
Contains the pathname of the previous working directory. This is used by **cd –**.

PWD Contains the pathname of the current working directory. This is set by **cd** after changing to that directory.

Localization

cd uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Usage Note

cd is a built-in shell command.

Exit Values

0 Successful completion

1 Failure due to any of the following:

- No **HOME** directory
- No previous directory
- A search for *directory* failed
- An *old*-to-*new* substitution failed

2 An incorrect command-line option

Messages

Possible error messages include:

dir **bad directory**
cd could not locate the target directory. This does not change the working directory.

Restricted
You are using the restricted version of the shell (for example, by specifying the **–r** option for **sh**). The restricted shell does not allow the **cd** command.

No HOME directory
You have not assigned a value to the **HOME** environment variable. Thus, when you run **cd** in order to return to your home directory, **cd** cannot determine what your home directory is.

No previous directory
You tried the command **cd –** to return to your previous directory; but there is no record of your previous directory.

Pattern *old* **not found in** *dir*
You tried a command of the form **cd** *old new*. However, the name of the working directory *dir* does not contain any string matching the regular expression *old*.

Portability

POSIX.2, X/Open Portability Guide.

All UNIX systems feature the first form of the command.

The **cd** *old new* form of the command is an extension of the POSIX standard.

Related Information

dirs, **popd**, **pushd**, **set**, **sh**, **tcsh**

chaudit — Change audit flags for a file

Format

chaudit [**–Fdai**] *attr pathname ...*

Description

chaudit changes the audit attributes of the specified files or directories. Audit attributes determine whether or not accesses to a file are audited by the system authorization facility (SAF) interface.

Note: **chaudit** can be used only by the file owner or a superuser for non-auditor-requested audit attributes. It takes a user with auditor authority to change the auditor-requested audit attributes.

Options

–F If you specify a directory as a pathname on the command, **chaudit** changes the audit characteristics of all files in that directory. Subdirectory audit characteristics are not changed.

–d If you specify a directory as a pathname on the command, **chaudit** changes the audit characteristics of all the subdirectories in that directory. File audit characteristics are not changed.

–a Auditor-requested audit attributes are to be changed for the files or directories specified. If **–a** is not specified, user-requested audit attributes are changed.

–i Does not issue error messages concerning file access authority, even if **chaudit** encounters such errors.

The symbolic form of the *attr* argument has the form:

```
[operation]
op auditcondition[op auditcondition ...]
```

The *operation* value is any combination of the following:

r Sets the file to audit read attempts.
w Sets the file to audit write attempts.
x Sets the file to audit execute attempts.

The default is all of the above (**rwx**).

The *op* part of a symbolic mode is an operator telling whether **chaudit** should turn file auditing on or off. The possible values are:

+ Turns on specified audit conditions.
- Turns off specified audit conditions.
= Turns on the specified audit conditions and turns off all others.

The *auditcondition* part of a symbolic mode is any combination of the following:

s Audit on successful access if the audit attribute is on.
f Audit on failed access if the audit attribute is on.

You can specify multiple symbolic *attr* values if you separate them with commas.

Examples

1. The command:

   ```
   chaudit –s file
   ```

   changes the file **file** so that successful file accesses are not audited.

2. The command:

   ```
   chaudit rwx=sf file1
   ```

   changes the file **file1** so that all successful and unsuccessful file accesses are audited.

3. The command:

   ```
   chaudit r=f file2
   ```

   changes the file **file2** so that unsuccessful file read accesses are audited.

4. The command:

   ```
   chaudit r-f,w+s file3
   ```

   changes the file **file3** to not audit unsuccessful file read accesses and to audit successful write accesses.

Localization

chaudit uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion

1 Failure due to any of the following:
- Inability to access a specified file
- Inability to change the audit attributes for a specified file
- Inability to not read the directory containing item to change
- Irrecoverable error when using the **–F** or **–d** option

2 Failure due to any of the following:
- Missing or incorrect *attr* argument
- Too few arguments

Messages

Possible error messages include:

fatal error during -F or -d option
: You specified the **–F** or **–d** option, but some file or directory in the directory structure was inaccessible. This may happen because of permissions or because you have removed a removable unit.

read directory *name*
: You do not have read permissions on the specified directory.

Portability

None. This is a security extension that comes with OpenMVS services.

Related Information

chmod, **chown**, **ls**

---

chcp — Set or query ASCII/EBCDIC code pages for the terminal

Format

chcp [**–r** | **–q**]
chcp [**–s**] [**–a** *ASCII_cp*] [**–e** *EBCDIC_cp*]

Description

chcp sets, resets, or queries the current ASCII/EBCDIC code conversion in effect for the controlling terminal. Use it when the terminal requires ASCII data and the shell application uses EBCDIC. Do not use **chcp** if you are logged on through the TSO/E OMVS command. The **_BPX_TERMPATH** environment variable enables shell scripts to tell if the user logged on from TSO, rather from rlogin or telnet.

Options

–a *ASCII_cp*
The name of the ASCII code page used by the terminal. EBCDIC data from the shell application is converted to this ASCII code page before it is sent out to the terminal. Data from the terminal is converted from this ASCII code page to EBCDIC before the application receives it.

The name of the ASCII code page is case-sensitive.

For a list of code pages supported by the shell, see *OS/390 C/C++ Programming Guide*.

–e *EBCDIC_cp*
The name of the EBCDIC code page used for this session. EBCDIC data from the shell application is converted from this EBCDIC code page to ASCII before it is sent out to the terminal. ASCII data from the terminal is converted to this EBCDIC code page before the application receives it.

The name of the EBCDIC code page is case-sensitive.

For a list of code pages supported by the OS/390 shell, see *OS/390 C/C++ Programming Guide*.

–q Queries the current ASCII and EBCDIC code pages for this terminal. The results are written to **stdout**. You cannot use any other options if you use the **–q** option.

–r Resets the ASCII/EBCDIC conversion for the terminal to the default code pages. The default ASCII code page is ISO8859-1, and the default EBCDIC code page is IBM-1047.

You cannot use **–r** with any other options.

–s Specifies that the ASCII/EBCDIC conversion for the terminal is to use the code pages specified by the **–a** and **–e** options. You cannot use **–s** with any other options other than **–a** or **–e**. Either **–a** or **e** (or both) must also be specified if **–s** is used.

The **chcp** query output is written to **stdout**. For example, if you enter

```
chcp –q
```

You get the following output:

```
Current ASCII  code page = ISO8859-1
Current EBCDIC code page = IBM-1047
```

Examples

1. To set the ASCII and EBCDIC code pages to IBM-eucJP and IBM-939, enter:

   ```
   chcp –a IBM-eucJP –e IBM-939
   ```

2. To change just the EBCDIC code page to IBM-277, enter:

   ```
   chcp –seIBM-277
   ```

3. To change just the ASCII code page to IBM-850, enter:

   ```
   chcp –a IBM-850
   ```

4. To reset ASCII/EBCDIC code page conversion to the default code pages for this terminal, enter:

   ```
   chcp –r
   ```

5. To query the current ASCII and EBCDIC code pages for this terminal, enter:

   ```
   chcp –q
   ```

Usage Notes

1. Do not use **chcp** when you are logged on from the TSO/E OMVS command because the OMVS command does not do any ASCII/EBCDIC code page conversion.

   Shell scripts can test **_BPX_TERMPATH** environment variable and bypass **chcp** when the user is logged on through OMVS. (The **_BPX_TERMPATH** environment variable enables shell scripts to tell if the user logged on from TSO/E rather than from rlogin or telnet.)

   Before starting the session, the TSO/E OMVS command sets **_BPX_TERMPATH** to “OMVS”.

   Sample shell script code:

   ```
   #  ---------------------------------------------
   #  Issue chcp only if not using TSO/E OMVS command
   #  ---------------------------------------------

   if
   test "$_BPX_TERMPATH" != "OMVS"
   then
   chcp –a IBM-850 –e IBM-1047
   fi
   ```

2. After running **chcp –s** to change the EBCDIC code page for the session, you may also need to alter or set the following environment variables to match the new code page:
   - **LANG**
   - **LC_ALL**
   - **LC_COLLATE**
   - **LC_CTYPE**
   - **LC_MESSAGES**
   - **LC_SYNTAX**
   - **NLSPATH**
3. The code page names supplied with the **–a** and **–e** options are passed to **iconv_open()** without any uppercase or lowercase conversion. Code page converters that convert between the specified ASCII and EBCDIC code pages must be available for **iconv()**.
4. If ASCII/EBCDIC conversion is not active for this terminal, both the ASCII and EBCDIC code pages must be specified on the **chcp –s** command. At other times, omit **–a** when just the EBCDIC code page needs to be changed. Omit **–e** when just the ASCII code page needs to be changed.
5. All code pages with names not known to **chcp** are considered to be singlebyte (SBCS) user-defined code pages. User-defined multibyte code pages are not supported.
6. **chcp** cannot check user-defined code page names to make sure that **–a** really specifies an ASCII code page and **–e** specifies an EBCDIC code page. In this case, specifying the wrong code pages may cause terminal input and output to be completely unreadable. It may also be impossible to enter any more shell commands.
7. **chcp** operates on the controlling terminal.
8. **chcp** should not be run as a background job.
9. The **–d** option specifies that special debugging information be printed. Specify this option only when requested by IBM.

Localization

chcp uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

`0` Successful completion

`1` Incorrect command-line arguments or options

2 Any of the following errors:

- There is no controlling terminal.
- The controlling terminal does not support ASCII/EBCDIC code page conversion (the TSO/E OMVS command, for example).
- **iconv**_() fails when passed the code page names specified on the command line.
- **chcp** cannot build SBCS conversion tables using **iconv()** when required.
- An I/O error occurred on the controlling terminal.
- Either the **–a** or **–e** was omitted and the **chcp –s** command was run while the terminal code page conversion is in binary mode.

Portability

None. **chcp** is not described in any standard.

Related Information

lm, **rlogin**

---

chgrp — Change the group owner of a file or directory

Format

chgrp [**–fhR**] *group pathname ...*

Description

chgrp sets the group ID to *group* for the files and directories named by the *pathname* arguments. *group* can be a group name from a group database, or it can be a numeric group ID (GID).

Note: **chgrp** can be used only by the file owner or a superuser. The file owner must have the new group as his or her group or one of the supplementary groups.

chgrp also turns off the set-user-ID bit and set-group -ID bit of the named files and directories.

Options

–f Does not issue an error message if **chgrp** cannot change the group ID. In this case, **chgrp** always returns a status of 0.

–h Does not attempt to follow the symbolic link (or external link), but instead makes changes to the symbolic link (or external link) itself.

–R If a *pathname* on the command line is the name of a directory, **chgrp** changes the group ID of all files and subdirectories in that directory. If **chgrp** cannot change some file or subdirectory in the directory, it continues to try to change the other files and subdirectories in the directory, but exits with a nonzero status.

Localization

chgrp uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 You specified **–f**, or **chgrp** successfully changed the group ownership of all the specified files and directories.

1 Failure due to any of the following:

- Inability to access a specified file
- Inability to change the group of a specified file
- An irrecoverable error was encountered when you specified the **–R** option

2 Failure due to any of the following:

- The command line contained an unknown option or too few arguments
- **chgrp** did not recognize the specified *group*

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The **–f** and **–h** options are an extension of the POSIX standard.

Related Information

chmod, **chown**

---

chmod — Change the mode of a file or directory

Format

chmod [**–fhR**] *mode pathname*

Description

chmod changes the access permissions, or *modes,* of the specified file or directory. (Modes determine who can read, write, or search a directory or file.)

Note: **chmod** can be used only by the file owner or a superuser.

Options

–f Does not issue error messages concerning file access permissions, even if **chmod** encounters such errors.

–h Does not make any permissions changes if the file is a symbolic link (or external link). Symbolic link (or external link) permissions cannot be changed on an OS/390 system.

–R If you specify a directory as a pathname on the command, **chmod** changes the access permissions of all files and subdirectories under that directory.

You can specify the *mode* value on the command line in either symbolic form or as an octal value.

The symbolic form of the *mode* argument has the form:

[*who*] *op permission*[*op permission* ...]

The *who* value is any combination of the following:

u Sets owner (user or individual) permissions.
g Sets group permissions.
o Sets other permissions.
a Sets all permissions; this is the default. If a *who* value is not specified, the default is **a**, modified by **umask**.

The *op* part of a symbolic mode is an operator that tells **chmod** to turn the permissions on or off. The possible values are:

+ Turns on a permission.
– Turns off a permission.
= Turns on the specified permissions and turns off all others.

The *permission* part of a symbolic mode is any combination of the following:

r Read permission. If this is off, you cannot read the file.

x Execute permission. If this is off, you cannot run the file.

X Execute or search permission for a directory; or execute permission for a file only when the current mode has at least one of the execute bits set.

w Write permission. If this is off, you cannot write to the file.

s If in owner permissions section, the *set-user-ID* bit is on; if in group permissions section, the *set-group-ID* bit is on.

t This represents the sticky bit. For a file, the sticky bit causes a search for the program in the user's STEPLIB, the link pack area, or link list concatenation. For a directory, the sticky bit allows files in a directory or subdirectories to be deleted or renamed only by the owner of the file, by the owner of the directory, or by a superuser.

You can specify multiple symbolic names if you separate them with commas.

Absolute modes are octal numbers specifying the complete list of attributes for the files; you specify attributes by ORing together these bits.

```
4000   Set-user-ID bit
2000   Set-group-ID bit
1000   Sticky bit
0400   User read
0200   User write
0100   User execute (or list directory)
0040   Group read
0020   Group write
0010   Group execute
0004   Other read
0002   Other write
0001   Other execute
```

Examples

```
chmod –w orgcht
```

removes write permission from **orgcht**.

```
chmod a=rwx aprsal
```

turns on read, write, and execute permissions, and turns off the set-user-ID bit, set-group-ID bit, and sticky-bit attributes. This is equivalent to `chmod 0777 aprsal`.

Localization

chmod uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

`0` Successful completion

`1` Failure due to any of the following:

- Inability to access a specified file
- Inability to change the modes on a specified file
- Inability to read the directory containing the item to change
- An irrecoverable error was encountered when using the **–R** option

`2` Failure due to any of the following:

- Missing or incorrect *mode* argument
- Too few arguments

Messages

Possible error messages include:

irrecoverable error during –R option
The **–R** option was specified, but some file or directory in the directory structure was inaccessible. This may happen because of permissions.

read directory *name*
Read permissions are not on the specified directory.

Portability

POSIX.2, X/Open Portability Guide.

The **–f** and **–h** options and the **t** permission are extensions of the POSIX standard.

Related Information

ls, **umask**

---

chmount — Change the mount attributes of a file system

Format

chmount [**–R** [**–D** | **–d** *destsys*] [**–a yes|no**]*pathname...*

Description

The **chmount** shell command, located in **/usr/sbin**, changes the mount attributes of a specified file system.

Note: A **chmount** user must have UID(0) or at least have READ access to the SUPERUSER.FILESYS.MOUNT resource found in the UNIXPRIV class.

Options

–a yes|no
: In a shared HFS environment, **–a yes** allows the system to automatically move logical ownership for a specified file system due to a system outage. **–a no** disables this move function.

–D
: Reassigns logical ownership of a file system to any available file system participating in shared HFS.

–d *destsys*
: To designate a specific reassignment, use **–d** *destsys*, where *destsys* becomes the logical owner of a file system in a shared HFS environment.

–R
: Changes the attributes of a specified file system and all file systems mounted below it in the file system hierarchy.

pathname... specifies the pathnames to use for locating the file systems that need attributes changed.

Example

To move ownership of the file system that contains **/u/wjs** to SY1:

```
chmount -d SY1 /u/wjs
```

Exit Values

0 Successful completion

Related Information

mount, **unmount**

---

chown — Change the owner or group of a file or directory

Format

chown [**–fhR**] *owner*[**:***group*] *pathname ...*

Description

chown sets the user ID (UID) to *owner* for the files and directories named by *pathname* arguments. *owner* can be a user name from the user data base, or it can be a numeric user ID. (If a numeric owner exists as a user name in the user data base, the user ID number associated with that user name is used.) If there is no change to the UID, then specify **–1**.

If you include a *group* name (that is, if you specify *owner* followed immediately by a colon (**:**) and then *group* with no intervening spaces, such as *owner*:*group*) **chown** also sets the group ID (GID) to *group* for the files and directories named. *group* can be a group name from the security facility group data base, or it can be a numeric group ID. If a numeric group exists as a group name in the group data base, the group ID number associated with that group is used. If there is no change to the GID, then specify **–1** (or do not specify the *:group*)..

Note: Only a superuser can change the UID. To change the GID, you must either be a superuser, or the effective user ID of the process must be equal to the user ID of the file owner, and the owner argument is also equal to the user ID of the file owner or **–1**, and the group argument is the calling process's effective group ID or one of its supplementary group IDs.

chown also turns off the set-user-ID bit and set-group -ID bit of the named files and directories.

Options

–f Does not issue an error message if **chown** cannot change the owner. In this case, **chown** always returns a status of zero. Other errors may cause a nonzero return status.

–h Does not attempt to follow the symbolic link (or external link), but instead makes the changes on the symbolic link (or external link) itself.

–R If *pathname* on the command line is the name of a directory, **chown** changes all the files and subdirectories in that directory to belong to the specified *owner* (and *group*, if *:group* is specified).

If a symbolic link is specified or encountered during the traversal of a file hierarchy, **chown** changes the directory referenced by the symbolic link and all files in the file hierarchy below it.

If **chown** cannot change some file or subdirectory in the directory, it continues to try to change the other files and subdirectories in the directory, but exits with a nonzero status.

Localization

chown uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 You specified **–f**, or **chown** successfully changed the ownership of all the specified files and directories.

1 Failure due to any of the following:

- Inability to access a specified file.
- Inability to change the owner of a specified file.
- Inability to read the directory containing the directory entry of the file.
- An irrecoverable error was encountered when using the **–R** option.

2 Failure due to any of the following:

- The command line contained an incorrect option.
- The command line had too few arguments.
- An owner was specified with a user ID that the system did not recognize.

Portability

POSIX.2, UNIX systems.

The **–f** and **–h** optio ns are an extension of the POSIX standard.

Related Information

chgrp, **chmod**

---

chroot — Change the root directory for the execution of a command

Format

chroot *directory command*

Description

If you have appropriate privileges, the **chroot** command changes the root directory to the directory specified by the directory parameter of a specific command. The new root directory will also contain its children.

In order to use **chroot**, you must either be a superuser (UID=0), or be a member of the BPX.SUPERUSER facility class.

The directory path name is always relative to the current root. If a nested **chroot** command is in effect, the directory path name is still relative to the current (new) root of the running process.

In order for your process to operate properly after the **chroot** is issued, you need to have in your new root all the files that your program depends on. For example, if your new root is **/tmp** and you issue an **ls**, you will get a not found error. To use **ls** with **/tmp** as your new root, you will need a **/tmp/bin** with **ls** in it before you issue the **chroot** command.

In addition, utilities that depend on locale-sensitive files (**/usr/lib/nis/***) may be unsuccessful if these files are not in the new root file system.

After **chroot** is issued, your current working directory is the new root (directory), **chroot** does not change environment variables.

directory Specifies the new root directory

command Specifies a command to run with the **chroot** command

Examples

1. To run the **ls** command with the **/tmp** directory as the root file system, enter:

   ```
   mkdir /tmp/bin
   cp /bin/ls /tmp/bin
   chroot /tmp ls
   ```

2. To run a child shell with another file system as the root file system (assuming that **/tmp** is the mount point of a file system), enter:

   ```
   mkdir /tmp/bin
   cp /bin/sh /tmp/bin
   chroot /tmp sh       or      chroot /tmp /bin/sh
   ```

   This makes the directory name / (slash) refer to the **/tmp** for the duration of the **/bin/sh** command. It also makes the original root file system inaccessible. The file system on the **/tmp** file must contain the standard directories of a root file system.

   Running the **/bin/sh** command creates a child shell that runs as a separate process from your original shell. Press the END OF FILE (Crtl-D) key sequence or type exit to end the child shell and go back to where you were in the original shell. This restores the environment of the original shell, including the meanings of the . (current directory) and the / (root directory).

3. To create a file relative to the original root, not the new one, enter:

   ```
   chroot Directory Command > file
   ```

   For example, `chroot /tmp ls > /bin/file` will create the file in **/bin/file**.

   Note: Redirection is handled by the current shell before **chroot** is executed.

4. To create a file relative to the new root, enter:

   ```
   chroot Directory 'Command > file'
   ```

   For example, `chroot /tmp 'ls > /bin/file'` will create the file in **/tmp/bin/file**.

5. Examples of how the current root changes:

```
Given the standard directories of the file system plus:

# echo $PATH
/bin
# ls /tmp/bin
bin file2 sh
# ls /tmp/bin/bin
file1 sh

# whence file2
#
# whence file1
#

# chroot /tmp 'whence file1'
#
# chroot /tmp 'type file2'
/bin/file2

# chroot /tmp/bin 'type file1'
/bin/file1
```

Exit Values

0 The command completed successfully

1 Failure due to any of the following:

- **chroot** seteuid failed
- User not authorized to issue **chroot**

2 Failure due to any of the following:

- Cannot chdir to directory specified
- **chroot** cannot change root
- Unable to execute the shell
- Incorrect command syntax

Note: If the SHELL environment variable is set, **chroot** uses its value to invoke the shell.

Limits

chroot may only be used by a superuser (UID=0) or a member of the BPX.SUPERUSER facility class.

---

cksum — Calculate and write checksums and byte counts

Format

cksum [**–ciprt**] [*file ...*]

Description

cksum calculates and displays a checksum for each input *file*. (A *checksum* is an error-checking technique used by many programs as a quick way to compare files that have been moved from one location to another to ensure that no data has been lost.) It also displays the number of 8-bit bytes in each *file*.

If you do not specify any files on the command line, or if you specify – as the filename, **cksum** reads the standard input (**stdin**).

The output has the form:

checksum bytecount filename

Options

cksum can calculate checksums in a variety of ways. The default is compatible with the POSIX.2 standard. You can specify other algorithms with the following options. The POSIX standard does not recognize these algorithms; they are provided for compatibility with the UNIX **sum** command.

–c Uses a standard 16-bit cyclic redundancy check (CRC-16).

–i Uses the CCITT standard cyclic redundancy check (CRC-CCITT). Data communication network protocols often use a cyclic redundancy check to ensure proper transmission. This algorithm is more likely to produce a different sum for inputs—the only difference is byte order.

–p Uses the POSIX.2 checksum algorithm. This is the default.

–r Enables the use of an alternate checksum algorithm that has the advantage of being sensitive to byte order.

–t Produces a line containing the total number of bytes of data read as well as the checksum of the concatenation of the input files.

Localization

cksum uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion

1 Failure due to any of the following:

- Inability to open input file
- An error reading the input file

2 Unknown command-line option

Portability

POSIX.2, X/Open Portability Guide.

All the listed options are extensions of the POSIX standard.

Related Information

cmp, **diff**, **ls**, **sum**, **wc**

---

cmp — Compare two files

Format

cmp [**–blsx**] *file1 file2* [*seek1*[*seek2*]]

Description

cmp compares two files. If either filename is –, **cmp** reads the standard input (**stdin**) for that file. By default, **cmp** begins the comparison with the first byte of each file. If you specify *seek1* and/or *seek2*, **cmp** uses it as a byte offset into *file1* or *file2* (respectively), and comparison begins at that offset instead of at the beginning of the files. The comparison continues (1 byte at a time) until a difference is found, at which point the comparison ends and **cmp** displays the byte and line number where the difference occurred. **cmp** numbers bytes and lines beginning with 1.

Options

–b Compares single blocks at a time. Normally, **cmp** reads large buffers of data into memory for comparison.

–l Causes the comparison and display to continue to the end; however, **cmp** attempts no resynchronization. **cmp** displays the byte number (in decimal) and the differing bytes (in octal) for each difference found.

–s Suppresses output and returns a nonzero status if the files are not identical.

–x Displays the differing bytes shown by the **–l** option in hex; normally **cmp** displays them in octal.

Localization

cmp uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 The files were identical
1 The files were not identical
2 Failure because of an error opening or reading an input file

Messages

Possible error messages include:

EOF on *filename*
cmp reached the end of the file on the specified file before reaching the end of the file on the other file.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The **–b** and **–x** options and the *seek* pointers are extensions of the POSIX standard.

Related Information

comm, **diff**, **uniq**

---

col — Remove reverse line feeds

Format

col [**–bfpx**] [*file ...*]

Description

col processes control characters for vertical line feeds and writes the processed text to the standard output. It is intended to be used as a filter between a program such as **nroff** and an output device that cannot handle reverse line feeds.

Where possible, blank characters (spaces) are converted to tabs; tab stops are assumed to be every eight characters.

col also removes all escape sequences except for those shown in the following list. ESC is the ASCII escape character, octal code 033.

Character	ASCII Control Character
Backspace	010
Carriage-return	015
Newline	012
Vertical Tab	013
SO	016
SI	017
Space	040
Tab	011
Reverse line feed	ESC-7
Reverse half-line feed	ESC-8
Forward half-line feed	ESC-9

The ASCII control characters SO and SI denote the beginning and end of text in an alternative character set. The set of each input character is remembered. **col** generates SO and SI characters as needed to output each character in the correct character set.

Options

–b Ignores backspace (**CRTL-H**) characters. If two characters are supposed to appear in the same space, the first character is ignored and the second is output.

–f Allows forward half-line motions. Normally these are changed to forward full-line motions.

–x Prevents conversion of spaces to tab characters.

Localization

col uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Usage Notes

1. **col** ignores vertical motions that back up over the first line, so you may get unexpected results if the first line contains superscripts.
2. Because **–f** allows escape sequences, it may cause unexpected results on terminals.

Exit Values

0 Successful completion

1 Failure due to any of the following:
- Incorrect command-line option
- Insufficient memory

Portability

UNIX systems.

This implementation does not handle doublebyte characters.

: (colon) — Do nothing, successfully

Format

: [*argument ...*]

tcsh shell: **:**

Description

The **:** (colon) command is used when a command is needed, as in the **then** condition of an **if** command, but nothing is to be done by the command. This command simply yields an exit status of zero (success). This can be useful, for example, when you are evaluating shell expressions for their side effects.

: (colon) in the tcsh shell

Performs as indicated above for the OS/390 version of : (colon).

Example

```
: ${VAR:="default value"}
```

sets *VAR* to a default value if and only if it is not already set.

Usage Notes

colon is a special built-in shell command.

Localization

colon uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

Because this command always succeeds, the only possible exit status is:

`0` Successful completion

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Information

sh, **tcsh**, **true**

---

comm — Show and select or reject lines common to two files

Format

comm [**–123**] *file1 file2*

Description

comm locates identical lines within files sorted in the same collating sequence, and produces three columns; the first contains lines found only in the first file, the second lines only in the second file, and the third lines that are in both files.

Options

–1 Suppresses lines that appear only in *file1*
–2 Suppresses lines that appear only in *file2*
–3 Suppresses lines that appear both in *file1* and *file2*

The options suppress individual columns. Thus, to list only the lines common to both files, use:

```
comm -12
```

To find lines unique to one file or the other, use:

```
comm -3
```

Observe that `comm -123` displays nothing.

Localization

comm uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion
1 Failure because of an error opening or reading an input file
2 Failure that generated a usage message, such as naming only one input file

Incorrect command-line options are reported but do not affect the exit status value.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Information

cmp, **diff**, **sort**, **uniq**

command — Run a simple command

Format

command [**–p**] *command-name* [*argument*...]
command [**–Vl–v**] *command-name*

Description

command causes the shell to suppress its function lookup and execute the given *command-name* and arguments as though they made up a standard command line. In most cases, if *command-name* is not the name of a function, the results are the same as omitting **command**. If, however, *command-name* is a special built-in command, (see **sh**), some unique properties of special built-in commands do not apply:

- A syntax error in the command does not cause the shell running the command to stop.
- Variable assignments specified with the special built-in command do not remain in effect after the shell has run the command.

Options

–p Searches for *command-name* using the default system variable **PATH**.

–v Writes a string indicating the pathname or command that the shell uses to invoke *command-name*.

–V Writes a string indicating how the shell interprets *command-name*. If *command-name* is a command, a regular built-in command, or an implementation-provided function found using the **PATH** variable, the string identifies it as such and includes the absolute pathname. If *command-name* is an alias, function, special built-in command, or reserved word, the string identifies it as such and includes its definition if it is an alias. If the command is a tracked alias, the string identifies it as *cached*.

Example

Typically, you use **command** when you have a command that may have the same name as a function. For example, here's a definition of a **cd** function that not only switches to a new directory but also uses **lc** to list the contents of that directory:

```
function cd {
    command cd $1
    lc
}
```

Inside the function, we use **command** to get at the real **cd**. If we didn't do this, the **cd** function would call itself in an infinite recursion.

Localization

command uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

- **NLSPATH**

See Appendix F for more information.

Usage Note

command is a built-in shell command.

Exit Values

If you specified **–v**, possible exit status values are:

0 Successful completion
1 **command** could not find *command-name*, or an error occurred
2 Failure due to incorrect command-line argument

If you did not specify **–v**, possible exit status values are:

126 **command** found *command-name*, but failed to invoke it.
127 An error occurred in the command or it could not find *command-name*.

Otherwise, the exit status of **command** is the exit status of *command-name*.

Portability

POSIX.2.

Related Information

sh

---

compress — Lempel-Ziv file compression

Format

compress [**–cDdfVv**] [**–b** *bits*] [*file ...*]

Description

compress compresses each input file using Lempel-Ziv compression techniques. If you do not specify any input files, **compress** reads data from standard input (**stdin**) and writes the compressed result to standard output (**stdout**).

The output files have the same names as the input files but with a **.Z** suffix. For example, **abc** is compressed into **abc.Z**. If the **.Z** file already exists and you did not specify the **–f** option, **compress** gives an error and asks whether it should overwrite the existing file.

compress uses the modified Lempel-Ziv algorithm described in *A Technique for High Performance Data Compression*, Terry A. Welch, *IEEE Computer,* vol. 17, no. 6 (June 1984), pp.8-19. **compress** first replaces common substrings in the file by 9-bit codes starting at 257. After it reaches code 512, **compress** begins with 10-bit codes and continues to use more bits until it reaches the limit set by the **–b** option.

After attaining the bits limit, **compress** periodically checks the compression ratio. If it is increasing, **compress** continues to use the existing code dictionary. However, if the compression ratio decreases, **compress** discards the table of substrings and

rebuilds it from scratch. This allows the algorithm to compensate for files, such as archives, where individual components have different information content profiles.

Options

–b *bits*
Limits the maximum number of bits of compression to *bits*. The value *bits* can be an integer from 9 to 16. The default is 16.

–c Writes the output to **stdout**. When you use this option, you can only specify one file on the command line.

–D Allows an extra degree of compression to be done for files such as sorted dictionaries where subsequent lines normally have many characters in common with the preceding line.

–d Decompresses argument files instead of compressing them. This works by overlaying the **compress** program with the **uncompress** program. For this to work, **uncompress** must be available somewhere in your search path (given by the **PATH** environment variable). Decompressing files this way is slower than calling **uncompress** directly.

–f Forces compression even if the resulting file is larger or the output file already exists. When you do not specify this option, files which are larger after compression are not compressed. **compress** does not print an error message if this happens.

–V Prints the version number of **compress**.

–v Prints statistics giving the amount of compression achieved. Statistics give the name of each file compressed and the compression ratio, expressed as a percentage. If the file resulting from compression is larger than the original, the compression ratio is negative.

Localization

compress uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion

1 Failure due to one of the following:

- Missing number of bits after the **–b** option
- Incorrect number of bits specified
- Failed to execute **uncompress**
- Unknown option
- Dictionary option—same count of string exceeded
- Output path or file name too long
- Cannot stat file
- Argument file not a regular file: unchanged

- Argument file has other links: unchanged
- No space for compression tables

2 One or more files were not compressed because the compressed version was larger than the original

Limits

This implementation of **compress** is limited to a maximum of 16-bit compression.

Portability

A binary-compatible version of **compress** with more options is often found on UNIX systems.

The **–D** option is an extension to traditional implementations of **compress**. The **–D**, **–d** and **–V** options are extensions of the POSIX standard.

For portability, you should restrict the number of bits in the code (**–b** option) to a value between 9 and 14.

Related Information

cpio, **pack**, **pax**, **tar**, **uncompress**, **unpack**, **zcat**

---

confighfs — Invoke vfs_pfsctl HFS functions

Format

confighfs [**–l**] [**–v n**] [**–f n**] [**–q**] [**pathname**] [**–x**[**n**] **size pathname**]

Note: The **l** option signifies a lowercase **L**, not an uppercase **i**.

Description

confighfs gives interactive shell users the ability to invoke vfs_pfsctl HFS functions. The vfs_pfsctl function is used to pass control information to the PFS (physical file system). For more information on vfs_pfsctl, see the *OS/390 UNIX System Services File System Interface Reference*.

Options

–l Query HFS limits.

Note: **l** signifies a lowercase **L**, not an uppercase **i**.

–v n Set virtual storage max to n (where n is in MB). Requires superuser authority.

–f n Set fixed storage min to n (where n is in MB). Requires superuser authority.

–q Query your global statistics.

pathname
Query file system statistics for the file system containing each of the path names specified.

–x size pathname
Extend the specified file system, where size is the amount to be extended suffixed by the extend unit of M, T, or C (for megabytes, tracks, or cylinders), and the pathname is a full or simple pathname to a file or directory in the file system to extend. Requires superuser authority.

–xn size pathname
Extend the specified file system to a new volume with the same rules as above. Requires superuser authority.

The following are internal debug options:

–dn Prints incoming and outgoing pfsctl buffers (where n is 0, 1, or 2).

–t Skips issuing the pfsctl.

Examples

Note: On systems running shared HFS, this command should only be issued on the server system (file system owner) for the file system pointed to by the pathname. Issuing it on client systems results in fields of zeros. See *OS/390 UNIX System Services Planning* for more information on OS/390 UNIX in a Sysplex.

1. To set virtual and fixed HFS buffer limits:

   ```
   > confighfs –v 128 –f 32
   ```

2. To extend the file system for your current directory 100 cylinders:

   ```
   > confighfs –x 100c .
   ```

3. If you need to get stats for the root file system and the file system mounted over **/tmp**, you would do the following:

   ```
   > confighfs / /tmp .
   ```

   Note: The **.** (period) in examples two and three above indicates the current directory.

configstk — Configure the AF_UEINT stack

Format

configstk {**–s**} *Configuration_file_name*

Description

configstk is used to configure the AF_UEINT stack. This command should initially be run from the **/etc/rc** script, which is executed as part of OS/390 UNIX System Services initialization. It should also be run each time the AF_UEINT network topology changes after OS/390 UNIX services have been initialized.

This command requires superuser authority.

Option

–s Does syntax checking only.

Files

configstk uses the following file:

Configuration_file_name
Specifies the configuration for the AF_UEINT stack. As with any system-wide configuration file, it should have the appropriate permissions set.

Syntax for Configuration Files

This file has two types of specifications, HOME and GATEWAY. Be careful when modifying the configuration file to insure that the F_UEINT environment is not corrupted due to user error.

HOME *ip_address* **BUFFERS(***number***) blocking**
This statement is required but you can only specify it once. The entire statement must be on a single line.

ip_address
Defines the single virtual IP address to be used by all RS/6K clients when accessing the OS/390 host, independent of how many RS/6K gateways are connected to a given OS/390 image. This implementation differs from the standard IP model which defines an IP address per physical adapter.

number
Defines the maximum number of 32K page-fixed buffers (in OMVS private memory) that are to be used by the protocol stack. The number specified is be distributed equally among the read and write flows. As new ESCON fibers are added to the configuration, additional IO buffers are required. Thruput decreases and overhead increases if the number specified is too restrictive. You should initially specify a value of 10 times the number of defined gateways for low-to-average use and increase it proportionally as the number of users increase). The maximum number of buffers allocated is the larger of six times the number of active gateways, or the number specified. A decrease in the number is not honored until the next IPL.

blocking
Indicates whether the internal blocking algorithm should be activated for outgoing packets. The default is BLOCKING. Specifying NOBLOCKING causes the internal optimization routines, which attempt to group multiple packets into a single blocked I/O, to be bypassed (such as single packet per block written on demand). Specifying BLOCKING minimizes the OS/390 overhead and maximizes the ESCON channel bandwidth, but can delay the packet delivery slightly.

GATEWAY *device_number checksum*
At least one of these statements is required and up to 32 can be specified. The entire statement must be on a single line. This statement maps the target RS/6K IP addresses to the gateway that will process the request. The device number to define the gateway must be the first of an even-odd pair of subchannels (both configured thru a single ESCON fiber) between the OS/390 image and the RS/6K gateway. Multiple target IP addresses can be mapped to a given gateway. A given target IP address can be mapped to at most one gateway.

device_number
Specifies the hex address of device to be configured. This number must be four hex digits and must be an even number.

checksum
Indicates whether a reliable communications path exists between the communicating applications. Specify CHECKSUM if any portion of the path between the communicating applications is unreliable (such as a LAN). Specify NOCHECKSUM if the entire path is reliable (such as a SP2 fast switch or ESCON).

A list of IP addresses immediately follows this statement, one IP address per line. At least one IP address must be specified for each gateway device. Up to 256 IP addresses can be specified in the configuration file.

Blank lines are permitted and lines beginning with /* are treated as comments.

Examples

```
/* configure AF_UEINT sockets

/* name the ip address for this node, default to blocking enabled
home 10.32.166.20 buffers(20)

/* configure device 324
gateway 0324 nochecksum
10.34.166.20
10.34.166.24
10.34.166.26

/* configure device b28
gateway 0b28 checksum
10.36.166.20
10.36.166.22
10.36.166.24
10.36.166.26
```

continue — Skip to the next iteration of a loop in a shell script

Format

continue [*n*]

Description

continue skips to the next iteration of an enclosing **for**, **select**, **until**, or **while** loop in a shell script. If a number *n* is given, execution continues at the loop control of the *n*th enclosing loop. The default value of *n* is 1.

Usage Note

continue is a special built-in shell command.

Localization

continue uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion
1 The value of *n* given was not an unsigned decimal greater than 0.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Information

break, **sh**, **tcsh**

---

cp — Copy a file

Format

cp [**–cfimMUv**] [**–pIF formatIBITIX**] [**–P params**]*file1 file2*
cp [**–ACcfimMUv**] lbrk;**–pIF formatIBITIX**] [**–S suffix**] *file ... directory*
cp –R [**–cfimp**] *source... directory*
cp –r [**–cfimp**] *source... directory*

Description

cp copies files to a target named by the last argument on its command line. If the target is an existing file, **cp** overwrites it; if it does not exist, **cp** creates it. If the target file already exists and does not have write permission, **cp** denies access and continues with the next copy.

If you specify more than two pathnames, the last pathname (that is, the target) must be a directory. If the target is a directory, **cp** copies the sources into that directory with names given by the final component of the source pathname.

You can also use **cp** to copy files to and from MVS data sets. If you specify more than one file to be copied, the target (last pathname on command line) must be either a directory or a partitioned data set. If the target is an MVS partitioned data set, the source cannot be a UNIX directory.

When copying records, the string " \n" is copied the same way as the string "\n": both are read back as "\n", where "\n" indicates that OS/390 C++ will write a record containing a single blank to the file (the default behavior of OS/390 C/C++). All other blanks in your output are read back as blanks, and any empty (zero-length) records are ignored on input. However, if the environment variable **_EDC_ZERO_RECLEN** is set before calling **cp**, an empty record is treated as a single newline and is not ignored. Also, if **_EDC_ZERO_RECLEN** is set, a single

newline is written to the file as an empty record, and a single blank will be represented by " \n".

You can copy:

- One file to another file in the working directory
- One file to a new file on another directory
- A set of directories and files to another place in your file system
- A UNIX file to an MVS data set
- An MVS data set to a filesystem
- An MVS data set to an MVS data set

Options

–A Specifies that all suffixes (from the first period till the end of the target) be truncated. **–A** has precedence over **–M** and **–C** options. **–S** will be turned off if **–A** is the last option specified.

–B Specifies that the data to be copied contains binary data. When you specify **–B**, **cp** operates without any consideration for <newline> characters or special characteristics of DBCS data (this type of behavior is typical when copying across a UNIX system). **–B** is mutually exclusive with **–F**, **–X**, and **–T**, i.e., you will get an error if you specify more than one of these options.

–C Specifies truncating the filename(s) to 8 characters to meet the restriction in the MVS data set member.

–c (UNIX to UNIX only)
Prompts you to change the diskette if there is not enough room to complete a copy operation. This option has no effect on systems without floppy drives.

Note: The parent directories must already exist on the new target diskette.

–F *format*
Specifies if a file is binary or text and for text files, specifies the end-of-line delimeter. Also sets the file format to *format* if the target is a UNIX file. For text files, when copying from UNIX to MVS, the end-of-line delimeter will be stripped. When copying from MVS to UNIX, the end-of-line delimeter will be added (Code page IBM-1047 will be used to check for end-of-line delimeters).

If setting *format* fails, a warning will be displayed. However, **cp** will continue to copy any remaining files specified to be copied.

–F is mutually exclusive with **–B**, **–X**, **–p**, and **–T**. If you specify one of these options with **–F**, you will get an error. If **–F** is specified more than once, the last **–F** specified will be used.

For *format* you can specify:

not	not specified
bin	binary data

Or the following text data delimeters:

nl	newline
cr	carriage return
lf	line feed
crlf	carriage return followed by line feed
lfcr	line feed followed by carriage return
crnl	carriage return followed by new line

–f (UNIX to UNIX only)
Attempts to replace files that do not have write permission.

–i When copying to a UNIX target, **–i** asks you if you want to overwrite an existing file, whether or not the file is read-only.

–M Specifies that some characters of the filename are translated when copying between a UNIX file and an MVS data set member. Characters are translated as follows:

- _ (underscore) in UNIX is translated to @ in MVS DS members and vice versa.
- . (period) in UNIX is translated to # in MVS DS members and vice versa.
- – (dash) in UNIX is translated to $ in MVS DS members and vice versa.

–m (UNIX to UNIX only)
Sets the modification and access time of each destination file to that of the corresponding source file. Normally, **cp** sets the modification time of the destination file to the present.

–P params
Specifies the parameters needed to create a sequential data set if one does not already exist. You can specify the RECFM, LRECL, BLKSIZE, and SPACE in the format the CRTL **fopen()** function uses. However, LRECL and BLKSIZE can be used for variable record format only.

`SPACE=(units,(primary,secondary)` where the following values are supported for units:

- any positive integer indicating BLKSIZE
- CYL (mixed case)
- TRK (mixed case)

For example:

```
SPACE=(500,(100,500)) units, primary, secondary
SPACE=(500,100) units and primary only
```

Note: CRTL **fopen()** arguments: LRECL specifies the length, in bytes, for fixed-length records and the maximum length for variable-length records. BLKSIZE specifies the maximum length, in bytes, of a physical block of records. RECFM refers to the record format of a data set and SPACE indicates the space attributes for MVS data sets.

–p Preserves the modification and access times (as the **–m** option does); in addition, it preserves file mode, owner, and group owner, if authorized. It also preserves extended attributes

–p is mutually exclusive with **–F**. If both are specified, you will get an error.

–R (UNIX to UNIX only)
"Clones" the source trees. **cp** copies all the files and subdirectories specified by *source...* into *directory*, making careful arrangements to duplicate special files (FIFO, character special). **cp** will traverse directories by following symbolic links through the file hierarchy.

–r (UNIX to UNIX only)
"Clones" the source trees, but makes no allowances for special files (FIFO, character special). Consequently, **cp** attempts to read from a device rather

than duplicate the special file. This is similar to, but less useful than, the preferred **–R**.

–S *d=suffix|a=suffix*

- *d=suffix*

  Removes the specifed suffix from a file.

- *a=suffix*

  Appends the specified suffix to a file.

–S has precedence over **–M** and **–C**. It also turns off the **–A** option (if **–S** is the last specified option).

–T Specifies that the data to be copied contains text data. See “Usage Notes” on page 137 for details on how to treat text data. This option looks for IBM-1047 end-of-line delimeters, and is mutually exclusive with **–F**, **–X**, and **–B**, i.e., you will get an error if you specify more than one of these options.

Note: **–T** is ignored when copying across UNIX file systems.

–U Keeps filenames in uppercase when copying from MVS data set members to UNIX files. The default is to make filenames lowercase.

–v Verbose

–X Specifies that the data to be copied is an executable. Cannot be used in conjunction with **–F**, **–X**, or **–B**.

Note: If neither **–F|B|T nor X** is specifed, **cp** will first check the format of the MVS data set indicated and then try to determine the type of file.

Limits & Requirements

General Requirements

1. To specify an MVS data set name, precede the name with double slashes (//). For example, to specify the fully qualified data set names 'turbo.gammalib' and 'turbo.gammalib(pgm1)', you write:

   ```
   "//'turbo.gammalib'"
   "//'turbo.gammalib(pgm1)'"
   ```

   The same goes for data set names that are not fully qualified:

   ```
   //turbo
   ```

2. For PDS (partitioned data set) or PDSE (partitioned data set extended), to avoid parsing by the shell, the name should be quoted or minimally, the parenthesis should be escaped. For example, to specify 'turbo(pgm1)', you can use quotes:

   ```
   "//turbo(pgm1)"
   ```

   or escape the parenthesis:

   ```
   //turbo\(pgm1\)
   ```

   As indicated above, a fully qualified name must be single-quoted (as is done within TSO). To prevent the single quotes from being interpreted by the shell, they must be escaped or the name must be placed within regular quotation marks. See the 'turbo.gammalib' examples above.

3. If you specify a UNIX file as source and the MVS data set (target) does not exist, a sequential data set will be created. If the partitioned data set exists, the UNIX file will be copied to the partitioned data set member.
4. If source is an MVS data set and target is a UNIX directory, the UNIX directory must exist.
5. You cannot have a UNIX directory, partitioned data set, or sequential data set as source if the target is a partitioned data set.
6. To copy all members from a partitioned data set, you may specify the partitioned data set as source and a UNIX directory as target.

MVS data set naming limitations

- Data set names may only contain uppercase alphabetic characters (A-Z). Lowercase characters will be converted to uppercase during any copies to MVS data sets.
- Data set names can contain numeric characters 0–9 and special characters @, #, and $.
- Data set names cannot begin with a numeric character.
- A data set member name cannot be more than 8 characters. If a filename is longer than 8 characters or uses characters that are not allowed in an MVS data set name, the file is not copied. You may use the **–C** option to truncate names to 8 characters.

Limitations: UNIX to MVS data set

1. If you specify a sequential MVS data set that is in undefined record format, the file is copied as binary.
2. If you specify a PDSE that is in undefined record format, the first file successfully copied determines in what format files will be copied. Note that PDSE does not allow mixture. So if the first successfully copied file is an executable, the PDSE will have program objects only and all other files will fail. On the other hand, if the first file is data, then all files are copied as binary.
3. If you specify a PDS that is in undefined record format, UNIX executables are saved as PDS load modules. All other files are copied as binary.
4. If you specify an MVS data set that is either in variable length or fixed record length and you have not set the file format, text files are copied as text, binaries as binary, and executables as binary. (IBM-1047 end-of-line delimeters are detected in the data)
5. If you set the file format, the set value is used to determine if data is binary or text.

Limitations: MVS data set to UNIX

1. If an HFS file does not exist, one is created using 666 mode value:

   ```
   666 mode value:  owner(rw-)  group(rw-)  other(rw-)
   ```

   whether data is binary or text. If the data to be copied is a shell script or executable, an HFS file is created using 777 mode value:

   ```
   777 mode value:  owner(rwx)  group(rwx)  other(rwx)
   ```

2. If an HFS exists and the file format is set, **cp** copies the file as that format. Otherwise,
   - load modules (PDS) are stored as UNIX executables and program objects (PDSE) are copied since they are the same as executables;
   - data within data sets of undefined record format are copied as binary if the data is not a program object or load module;
   - and data found within data sets of fixed length or variable record length are copied as text. (IBM-1047 end-of-line delimeters are detected in the data)

Limitations: MVS to MVS

1. Options **–A**, **–C**, **–f**, and **–S** are ignored.
2. If target and source are in undefined record format (and neither is a sequential data set), **cp** will attempt to copy the data as a load module. If that fails, then **cp** will copy the data as binary.
3. If target and source are in undefined record format and either is a sequential data set, **cp** copies the data as binary.
4. If the source has a fixed or variable record length and the target is in undefined record format, **cp** copies the data as binary.
5. If the source is in undefined record format and the target has a fixed or variable record length, **cp** copies the data as binary.
6. If both source and target are in fixed or variable record length, **cp** copies the data as text.

Limitations: Copying executables into a PDS

1. A PDS may not store load modules that incorporate program management features.
2. **c89**, by default, produces objects using the highest level of program management.
3. If you plan on copying a load module to a PDS, you may use a pre-linker which produces output compatible with linkage editor. Linkage editor generated output can always be stored in a PDS.

The following table is a quick reference for determining the correct use of options with **cp**.

Figure 3 (Page 1 of 2). **cp** *Format: File to File and File ... (multiple files) to Directory*

Source/Target	Options Allowed	Options Ignored	Options Failed
UNIX File/UNIX File	Ffip	ABCMPSTUX	
UNIX File/Sequential Data Set	BFiPT	ACfMpSU	X
UNIX File/PDS or PDSE Member	BFiTX	ACfMPpSU	
Sequential Data Set/UNIX File	BFfiTU	ACMPpS	X
Sequential Data Set/Sequential Data Set	BFiPT	ACfMpSU	X
Sequential Data Set/PDS or PDSE Member	BFiT	ACfMPpSU	X
PDS or PDSE Member/UNIX File	BFfiTUX	ACMPpS	

Figure 3 (Page 2 of 2). **cp** *Format: File to File and File ... (multiple files) to Directory*			
Source/Target	**Options Allowed**	**Options Ignored**	**Options Failed**
PDS or PDSE Member/Sequential Data Set	BFiPT	ACfMpSU	X
PDS or PDSE Member/PDS or PDSE Member	BFiTX	ACfMPpSU	
UNIX File/UNIX Directory	ACFipS	BMPTUX	
PDSE or PDSE Member/UNIX Directory	BFfiMSTUX	ACMPp	
UNIX File/Partitioned Data Set	ABCFiMSTX	fPpU	
PDS or PDSE Member/PartitionedData Set	BFiTX	ACfMPpSU	
Partitioned Data Set/UNIX Directory	ABCFfiMSTUX	Pp	

The tables that follow indicate the kind of copies allowed using **cp**.

Figure 4. **cp** *Format: File to File*		
Source	**Target**	**Allowed**
UNIX File, Sequential Data Set, or Partitioned Data Set Member	UNIX File, Sequential Data Set, or Partitioned Data Set Member	Yes
UNIX Directory	UNIX Directory	No (unless **cp** is used with **–R** or **–r**)
Partitioned Data Set	UNIX Directory (dir) NOTE: results in each member of data set are moved to dir.	Yes
UNIX Directory	Partitioned Data Set	No
Partitioned Data Set	Partitioned Data Set	No
UNIX File or Partitioned Data Set Member	UNIX Directory (must exist) or Partitioned Data Set	Yes
Partitioned Data Set Member	Partitioned Data Set	Yes

Figure 5. **cp** *Format: File... (multiple files) to Directory*		
Source	**Target**	**Allowed**
Any combination of UNIX File or Partitioned Data Set Member	UNIX Directory or Partitioned Data Set	Yes
Any combination of UNIX Directory or Sequential Data Set	Partitioned Data Set or UNIX Directory	No
Partitioned Data Set	UNIX Directory	Yes
Partitioned Data Set	Partitioned Data Set	No

Usage Notes

UNIX to MVS

1. To copy from UNIX to a partitioned data set, you must allocate the data set before doing the **cp**.
2. If an MVS data set does not exist, **cp** will allocate a new sequential data set of variable record format.
3. For text files, all <newline> characters are stripped during the copy. Each line in the file ending with a <newline> character is copied into a record of the MVS data set. If text file format is specified or already exists for the source file, that file format will be used for end-of-line delimeter instead of <newline>. Note that **cp** looks for IBM-1047 end-of-line delimeters in data.

   You cannot copy a text file to an MVS data set that has an undefined record format:

   - For an MVS data set in fixed record format, any line copied longer than the record size will cause **cp** to fail with a displayed error message and error code. If the line is shorter than the record size, the record is padded with blanks.
   - For an MVS data set in variable record format: Any line copied longer than the largest record size will cause **cp** to fail with a displayed error message and error code. Record length is set to the length of the line.
4. For binary files, all copied data is preserved:
   - For an MVS data set in fixed record format, data is cut into chunks of size equal to the record length. Each chunk is put into one record. The last record is padded with blanks.
   - For an MVS data set in variable record format, data is cut into chunks of size equal to the largest record length. Each chunk is put into one record. The length of the last record is equal to length of the data left.
   - For an MVS data set in undefined record format, data is cut into chunks of size equal to the block size. Each chunk is put into one record. The length of the last record is equal to the length of the data left.
5. For load modules, the partitioned data set specified must be in undefined record format otherwise the executable will not be copied.
6. If more than one filename is the same, the file is overwritten on each subsequent copy.
7. If a UNIX filename contains characters that are not allowed in an MVS data set, it will not be copied. If the UNIX filename has more than 8 characters, it can not be copied to an MVS data set member. (See the **–ACMS** options for converting filenames)
8. You are not allowed to copy files into data sets with spanned records.
9. PDSE cannot have a mixture of program objects and data members. PDS allows mixing, but it is not recommended.
10. Special files such as character special, external links, and fifo will not be copied to an MVS data set.
11. If a file is a symbolic link, **cp** will copy the resolved file, not the link itself.

12. UNIX file attributes are lost when copying to MVS. If you wish to preserve file attributes, you should use the **pax** utility.

MVS to UNIX

1. If the specified target HFS file exists, the new data overwrites the existing data. The mode of the file is unchanged.
2. If the specified HFS file does not exist, it will be created using 666 mode value if binary or text (this is subject to **umask**). If the data to be copied is a shell script or executable, the HFS file will be created with 777 mode value (also subject to **umask**).
3. When you copy MVS data sets to UNIX text files, a <newline> character is appended to the end of each record. If trailing blanks exist in the record, the <newline> character is appended after the trailing blanks. If the file format option is specified or the target file has the file format set, that file format is used as the end-of-line delimeter instead of <newline>.
4. When you copy MVS data sets to UNIX binary files, the <newline> character is not appended to the record.
5. You cannot use **cp** to copy data sets with spanned record lengths.

Examples

1. To specify **–P params** for a non-existing sequential target:

   ```
   cp -P "RECFM=U,space=(500,100)"file "//'turbo.gammalib'"
   ```

2. To copy file **f1** to a fully qualified sequential data set `'turbo.gammalib'` and treat it as a binary:

   ```
   cp -F bin f1 "//'turbo.gammalib'"
   ```

3. To copy all members from a fully qualified PDS `'turbo.gammalib'` to an existing UNIX directory `dir`:

   ```
   cp "//turbo.gammalib'" dir
   ```

4. To drop .c suffixes before copying all files in UNIX directory `dir` to an existing PDS `'turbo.gammalib'`:

   ```
   cp -S d=.c dir/* "//'turbo.gammalib'"
   ```

Localization

cp uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion

1 Failure due to any of the following:

- An argument had a trailing slash (/) but was not a directory
- Inability to find a file
- Inability to open an input file for reading
- Inability to create or open an output file
- A read error occurred on an input file
- A write error occurred on an output file
- The input and output files were the same file
- An irrecoverable error when using **–r** or **–R**. Possible irrecoverable **–r** or **–R** errors include:
  - Inability to access a file
  - Inability to change permissions on a target file
  - Inability to read a directory
  - Inability to create a directory
  - A target that is not a directory
  - Source and destination directories are the same

2 Failure due to any of the following:

- An incorrect command-line option
- Too few arguments on the command line
- A target that should be a directory but isn't
- No space left on target device
- Insufficient memory to hold the data to be copied
- Inability to create a directory to hold a target file

Messages

Possible error messages include:

cannot allocate target string
cp has no space to hold the name of the target file. Try to release some memory to give **cp** more space.

name **is a directory (not copied)**
You did not specify **–r** or **–R**, but one of the names you asked to copy was the name of a directory.

*target name***?**
You are attempting to copy a file with the **–i** option, but there is already a file with the target name. If you have specified **–f**, you can write over the existing file by typing `y` and pressing <Enter>. If you do not want to write over the existing file, type `n` and press <Enter>. If you did not specify **–f** and the file is read-only, you are not given the opportunity to overwrite it.

source *name* **and target** *name* **are identical**
The source and the target are actually the same file (for example, because of links). In this case, **cp** does nothing.

unreadable directory *name*
cp cannot read the specified directory—for example, because you do not have appropriate permission.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The **–f** and **–m** options are extensions of the POSIX standard.

Related Information

cat, **cpio**, **ln**, **mv**, **rm**

---

cpio — Copy in/out file archives

Format

cpio –o [**–aBcvyz**] [**–C** *blocksize*] [**–O** *file*] [**–V** *volpat*]
cpio –i [**–BbcdfmrsStuvqyz**] [**–C** *blocksize*] [**–I** *file*] [**–V** *volpat*] [*pattern ...*]
cpio –p [**–aBdlmruv**] *directory*

Description

cpio reads and writes files called **cpio** archives. A **cpio** archive is a concatenation of files and directories preceded by a header giving the filename and other file system information. With **cpio**, you can create a new archive, extract contents of an existing archive, list archive contents, and copy files from one directory to another.

Options

Every call to **cpio** must specify one and only one of the following *selector* options:

–i Reads an existing archive (created with the **–o** option) from the standard input (**stdin**). Unless you specify the **–t** option, **cpio** extracts all files matching one or more of the given *pattern* arguments from the archive. Patterns are the same as those used by filename generation (see **sh**). When you do not specify a *pattern* argument, the default pattern * is used; as a result, **cpio** extracts all files.

–o Writes a new archive to the standard output (**stdout**), using the list of files read from **stdin**. Such a list might be produced by the **ls** or **find** commands. For example:

```
ls . | cpio –o >arch
```

uses **ls** to list the files of the working directory and then pipes this list as input to **cpio**. The resulting archive contains the contents of all the files, and is written to **arch**.

–p Is shorthand for:

```
cpio –o | (cd directory; cpio –i)
```

where **cpio –i** is performed in the given directory. You can use this option to copy entire file trees.

Consult the syntax lines to determine which of the following additional options can be applied with a particular selector option:

–a Resets the access time (of each file accessed for copying to the archive) to what it was before the copy took place.

–B Uses buffers of 5120 bytes for input and output rather than the default 512-byte buffers.

–b Causes 16-bit words to be swapped within each longword and bytes to be swapped within each 16-bit word of each extracted file. This facilitates the transfer of information between different processor architectures. This is equivalent to specifying both the **–s** and **–S** options.

–C *blocksize*
Sets the buffer size to a specified blocksize, rather than the default 512-byte buffers.

–c Reads and writes header information in ASCII form. Normally, **cpio** writes the header information in a compact binary format. This option produces an archive more amenable to transfer through nonbinary streams (such as some data communication links) and is highly recommended for those moving data between different processors.

–d Forces the creation of necessary intermediate directories when they do not already exist.

–f Inverts the sense of pattern matching. More precisely, **cpio** extracts a file from the archive if and only if it does *not* match any of the *pattern* arguments.

–I *file*
Causes input to be read from the specified file, rather than from **stdin.**

–l Gives permission to create a link to a file rather than making a separate copy.

–m Resets the modification time of an output file to the modification time of the source file. Normally, when **cpio** copies data into a file, it sets the modification time of the file to the time at which the file is written. This option has no effect on directories.

–O *file*
Causes output to be written to the specified file, rather than to **stdout.**

–q Assumes all created files are text. This means that any `\r` (carriage return) characters are stripped, and only the `\n` (newlines) are retained.

Do not use the **–q** option for converting text to a system-independent format, because that would require all files to be read twice.

–r Lets you rename files as **cpio** works. When extracting, **cpio** displays the name of the component it is about to extract and gives you the chance to specify a name for the extracted file. If you enter **.** as the name, **cpio** processes the file or directory with no modification to the name. If you just press Enter, **cpio** skips the file.

–S For portability reasons, swaps pairs of 16-bit words within longwords (a 32-bit or 64-bit word) only when extracting files. This option does not affect the headers.

–s For portability reasons, swaps pairs of bytes within each 16-bit word only when extracting files. **–s** does not affect the headers.

–t Prevents files extraction, producing instead a table of filenames contained in the archive. See the description of the **–v** option.

–u Copies an archive file to a target file even if the target is newer than the archive. Normally, **cpio** does not copy the file.

–V *volpat*
Provides automatic multivolume support. **cpio** writes output to files, the names of which are formatted using *volpat*. The current volume number replaces any occurrence of # in *volpat*. When you invoke **cpio** with this option, it asks for the first number in the archive set, and waits for you to type the number and a carriage return before its precedes with the operation. **cpio** issues the same sort of message when a write error or read error occurs on the archive; the reasoning is that this kind of error means that **cpio** has reached the end of the volume and should go on to a new one.

–v Provides more verbose information than usual. **cpio** prints the names of files as it extracts them from or adds them to archives. When you specify both **–v** and **–t**, **cpio** prints a table of files in a format similar to that produced by the **ls –l** command.

–y When used with **–V**, does not ask for a volume number to begin with, but does ask if it gets a read or write error.

–z Performs Lempel-Ziv compression. Output is always a 16-bit compression. On input, any compression up to 16-bit is acceptable.

Usage Notes

1. Use the **pax** command if you need to use multibyte patterns when searching for filenames.
2. The POSIX 1003.1 standard defines formats for **cpio** archives that limit the UIDs and GIDs that can be stored to the maximum value of 262143. Values larger than this will not be properly restored.
3. The byte and word swapping done by the **–b**, **–S**, and **–s** options is effective only for the file data written. With or without the **–c** option, header information is always written in a machine-invariant format.
4. The **cpio** utility is scheduled to be withdrawn from XPG; for standards compatibility, you should use **pax**.

Localization

cpio uses the following localization environment variable:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion
1 Failure due to any of the following:
- An incorrect option
- Incorrect command-line arguments
- Out of memory
- Compression error
- Failure on extraction
- Failure on creation

Portability

X/Open Portability Guide, non-Berkeley UNIX systems after Version 7.

The **–q**, **–V**, **–y**, and **–z** options are specific to the OS/390 shell.

Related Information

compress, **cp**, **dd**, **find**, **ls**, **mv**, **pax**, **tar**, **cpio**, **uncompress**

Also see the **pax** file format description in Appendix H.

---

cron daemon — Run commands at specified dates and times

Format

cron

Description

cron is a clock daemon that runs commands at specified dates and times. You can specify regularly scheduled commands as described in **crontab**. You can also submit jobs that are to be run only once using the **at** command. **cron** runs commands with priorities and limits set by the **queuedefs** file.

cron only examines **crontab** files and **at** command files when initializing or when a file changes using **crontab** or **at**. This reduces the overhead of checking for new or changed files at regularly scheduled intervals.

The setuid bit for **cron** should never be set; however, it must be started by a user with appropriate privileges to issue the setuid call for any UID. Because **cron** never exits, it should only be run once, normally during the system initialization process. **cron** automatically forks and runs itself in the background, in a new shell session. **cron** uses the **pid** file to prevent more than one **cron** running at the same time.

When matching the date and time expressions given in **crontab** entries, **cron** uses the time zone in effect when the system started the daemon. As a result, you should ensure that the **TZ** environment variable is set at this time. For information on setting the **TZ** environment variable, see Appendix I. For **at** jobs, **cron** uses the value of **TZ** in effect when you submitted the job.

at, **batch**, and **crontab** submit jobs to **cron**; the data for those jobs can contain doublebyte characters. When the jobs are executed, the data in the jobs are interpreted in the locale that **cron** is using. Because it is strongly recommended that **cron** be started in the POSIX locale, doublebyte characters in the jobs may not be interpreted correctly. You can get around this by calling the **setlocale()** system call in the job itself.

cron handles the following externally generated signals in a special way:

SIGTERM

Causes **cron** to exit. You can cause **cron** to exit with the following command:

```
kill –TERM pid
```

where *pid* is the cron's PID number. To find the cron's PID number, you can use:

```
ps -e | grep cron
```

SIGUSR1
Is sent by either **at** or **crontab** to indicate a new **at** job or an updated **crontab** entry. **cron** does not delete **at** jobs until they finish running. If the **cron** daemon is terminated while **at** jobs are running, **cron** runs them again when the daemon is restarted.

SIGUSR2
Writes internal **cron** queue information to the log file.

cron uses a number of files in the **/usr/lib/cron** directory to determine which users may and may not use the **at** and **crontab** commands.

- The file **at.allow** contains the list of users who have permission to use **at**.
- The file **at.deny** contains the list of users who do not have permission to use **at**.

If these files do not exist, only the superuser can use the **at** command. To allow all users access to **at**, there must be a null **at.deny** file and no **at.allow** file.

cron uses the files **cron.allow** and **cron.deny** in a similar manner.

- **cron.allow** contains the list of users who have permission to use **crontab**.
- **cron.deny** contains the list of users who do not have permission to use **crontab**.

If these files do not exist, only the superuser can use **crontab**. To allow all users access to **crontab**, there must be a null **cron.deny** file and no **cron.allow** file.

Files

cron uses the following files which reside in a system-defined directory:

/usr/spool/cron
The main **cron** directory.

/usr/spool/cron/atjobs
A directory containing **at** files.

/usr/spool/cron/crontabs
A directory containing **crontab** files.

/usr/spool/cron/log
A file that maintains a history of the commands being run. The systems administrator should truncate this file periodically.

/usr/spool/cron/pid
A file that **cron** uses to ensure that only one version of **cron** is currently running on the system.

/usr/lib/cron/at.allow
Contains a list of the users who can use the **at** command (one per line).

/usr/lib/cron/at.deny
Contains a list of the users who cannot use the **at** command (one per line).

/usr/lib/cron/cron.allow
Contains a list of the users who can use the **crontab** command (one per line).

/usr/lib/cron/cron.deny
Contains a list of the users who cannot use the **crontab** command (one per line).

/usr/lib/cron/queuedefs
The queue description file (see the description of **queuedefs** in 826).

Related Information

at, **crontab**

Appendix I also explains how to set the local time zone with the **TZ** environment variable.

---

crontab — Schedule regular background jobs

Format

crontab [**–e**|**–l**|**–r**] [**–u user**] [*file*]

Description

crontab creates or changes your crontab entry. The crontab is a system facility that automatically runs a set of commands for you on a regular schedule. For example, you might set up your crontab entry so it runs a job every night at midnight, or once a week during low-use hours. This job could perform regular maintenance chores, for example, backing up files or getting rid of unnecessary work files.

To set up a crontab entry, use:

```
crontab file
```

If you omit the **file** argument, **crontab** takes input from standard input (**stdin**).

Note: In this mode, you must provide your entire crontab file. This replaces any other existing crontab entries. If you issue **crontab** with no options, do not enter the end-of-file character or you will end up with an empty crontab file. Press INTERRUPT instead.

Input consists of six fields, separated by blanks. The first five give a date and time in the following form:

- A minute, expressed as a number from 0 through 59
- An hour, expressed as a number from 0 through 23
- A day of the month, expressed as a number from 1 through 31
- A month of the year, expressed as a number from 1 through 12
- A day of the week, expressed as a number from 0 through 6 (with 0 standing for Sunday)

Important Note: All times use a system default time zone. Your system administrator can tell you what it is. The **cron** daemon does not use the value of the environment variable **TZ** when **crontab** is invoked.

Any of these fields may contain an asterisk (*) standing for all possible values. For example, if you have an * as the day of the month, the job runs every day of the month. A field can also contain a set of numbers separated by commas, or a range of numbers, with the first number followed by a minus sign – followed by the second number. If you give specific days for both day of the month and day of the week, the two are ORed together. Here are some examples:

```
0 0 * * *          -- Midnight every day
0 0 * * 1-5        -- Midnight every weekday
0 0 1,15 * *       -- Midnight on 1st and 15th of month
0 0 1 * 5          -- Midnight on 1st of month and every Friday
```

The sixth field of a crontab entry is a string that your shell executes at the specified time. When the shell executes this string, it sets the **HOME**, **LOGNAME**, **PATH**, and **SHELL** environment variables to default values for you.

If the string in your crontab entry contains percent characters %, the shell interprets them as newline characters, splitting your string in several logical lines. The first logical line (up to the first %) is interpreted as the command you want to execute; subsequent logical lines are used as standard input to the command. If any real (not logical) line in the file is blank or begins with #, the shell ignores the line (treats it as a comment).

To obtain the output of the command in your crontab entry, redirect the standard output (**stdout**) and the standard error (**stderr**) into a file. If you do not do this, the system mails you the output from the command.

at, **batch**, and **crontab** submit jobs to **cron**; the data for those jobs may contain doublebyte characters. When the jobs are run, the data in the jobs are interpreted in the locale that **cron** is using. Because it is strongly recommended that **cron** be started in the POSIX locale, doublebyte characters in the jobs may not be interpreted correctly. You can get around this by calling **setlocale()** in the job itself.

Options

–e Lets you edit your crontab entry. **crontab** invokes an editor to edit the entry. If you have an **EDITOR** environment variable defined, **crontab** assumes that the variable's value is the name of the editor you want to use. If you do not have **EDITOR** defined, **crontab** uses **vi**.

If you do not have a crontab entry, **crontab** sets up a blank entry for you. When you exit from the editor, **crontab** uses the edited entry as your new entry.

–l Displays your current crontab entry on **stdout**.

–r Removes (deletes) your current crontab entry.

–u user
Uses the crontab entry of **user**. This requires the appropriate privileges.

You can specify only one of the **–e**, **–l**, or **–r** options.

Environment Variables

cron uses the following environment variables:

EDITOR Specifies the editor that the **–e** option invokes. The default editor is **vi**.

HOME Is set to your user ID's home directory (not necessarily the current value of **HOME**) when the commands in your crontab entry are run.

LOGNAME
Is set to your user ID when the commands in your crontab entry are run.

PATH Is set to a system-wide default value when the commands in your crontab entry are run.

TZ Is not used in time calculations. The **cron** daemon does, however, use this variable when **cron** is first started, usually when the system is started.

Localization

crontab uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion

1 Returned if the command fails for any reason. In this case, **crontab** does not change your crontab entry.

Portability

POSIX.2 User Portability Extension, X/Open Portability Guide, UNIX systems.

The **–u** option is an extension to the POSIX standard.

Related Information

at, **batch**, **bg**, **cron**

Also see the **queuedef** file format description in Appendix H.

---

csplit — Split text files

Format

csplit [**–Aaks**] [**–f** *prefix*] [**–n** *number*] *file arg arg ...*

Description

csplit takes a text file as input and breaks up its contents into pieces, based on criteria given by the *arg* value on the command line. For example, you can use **csplit** to break up a text file into chunks of ten lines each, then save each of those chunks in a separate file. See "Splitting Criteria" for more information. If you specify **–** as the *file* argument, **csplit** uses the standard input (**stdin**).

The files created by **csplit** normally have names of the form

xx*number*

where *number* is a 2-digit decimal number that begins at zero and increments by one for each new file that **csplit** creates.

csplit also displays the size, in bytes, of each file that it creates.

Options

–A Uses uppercase letters in place of numbers in the number portion of created filenames. This generates names of the form xxAA, xxAB, and so on.

–a Uses lowercase letters in place of numbers in the number portion of created filenames. This generates names of the form xxaa, xxab, and so on.

–f *prefix*
Specifies a prefix to use in place of the default xx when naming files. If it causes a filename longer than NAME_MAX bytes, an error occurs and **csplit** exits without creating any files.

–k Leaves all created files intact. Normally, when an error occurs, **csplit** removes files that it has created.

–n *number*
Specifies the number of digits in the number portion of created filenames.

–s Suppresses the display of file sizes.

Splitting Criteria

csplit processes the *arg*s on the command line sequentially. The first argument breaks off the first chunk of the file, the second argument breaks off the next chunk (beginning at the first line remaining in the file), and so on. Thus each chunk of the file begins with the first line remaining in the file and goes to the line given by the next *arg*.

arg values can take any of the following forms:

/regexp/ Takes the chunk as all the lines from the current line up to but not including the next line that contains a string matching the regular expression *regexp*. After **csplit** obtains the chunk and writes it to an output file, it sets the current line to the line that matched *regexp*.

/regexp/offset Is the same as the previous criterion, except that the chunk goes up to but not including the line that is a given *offset* from the first line containing a string that matches *regexp*. The *offset* can be a positive or negative integer. After **csplit** has obtained the chunk and written it to an output file, it sets the current line to the line that matched *regexp*.

Note: This current line is the first one that was not part of the chunk just written out.

%*regexp***%** Is the same as /*regexp*/, except that **csplit** does not write the chunk to an output file. It simply skips over the chunk.

%*regexp***%***offset* Is the same as /*regexp*/*offset*, except **csplit** does not write the chunk to an output file.

linenumber Obtains a chunk beginning at the current line and going up to but not including the *linenumber*th line. After **split** writes the chunk to an output file, it sets the current line to *linenumber*.

{*number***}** Repeats the previous criterion *number* times. If it follows a regular expression criterion, it repeats the regular expression process *number* more times. If it follows a *linenumber* criterion, **csplit** splits the file every *linenumber* lines, *number* times, beginning at the current line. For example,

```
csplit file 10 {10}
```

obtains a chunk from line 1 to line 9, then every 10 lines after that, up to line 109.

Errors occur if any criterion tries to "grab" lines beyond the end of the file, if a regular expression does not match any line between the current line and the end of the file, or if an *offset* refers to a position before the current line or past the end of the file.

Localization

csplit uses the following localization variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion

1 Failure due to any of the following:

- **csplit** could not open the input or output files
- A write error on the output file

2 Failure due to any of the following:

- Unknown command-line option
- The *prefix* name was missing after **–f**
- The *number* of digits was missing after **–n**
- The input *file* was not specified
- No *arg* values were specified
- The command ran out of memory
- An *arg* was incorrect

- The command found end-of-file before it was expected
- A regular expression in an *arg* was badly formed
- A line offset/number in an *arg* was badly formed
- A {number} repetition count was misplaced or badly formed
- Too many filenames were generated when using **–n**
- Generated file names would be too long

Portability

POSIX.2 User Portability Extension, X/Open Portability Guide, UNIX systems.

The **–A** and **–a** options are extensions to the POSIX standard.

Related Information

awk, **sed**

For more information about **regexp**, see Appendix C.

---

ctags — Create tag files for ex, more, and vi

Format

ctags [**–aBFwx**] [**–f** *tagfile*] *sourcefile ...*

Description

ctags creates a file named **tags** in the current directory. It summarizes the locations of various objects in the C source files named on the command line. All files with a **.c** or **.h** suffix are treated as C source files.

For C source code, **ctags** summarizes function, macro and `typedef` definitions. See Appendix H for a description of the format of the **tags** file.

The **tags** file is used by **ex**, **more**, and **vi** to support the **tag** command. The **tag** command can be used to edit the file containing a *name* in the **tags** file.

For **ex** and **vi**, the command is:

`: tag` *name*

For **more**, the command is:

`:t`*name*

After these commands are run, the **tags** file is searched for *name*. If it is found, the file associated in the **tags** file with that name is loaded and the line containing the *name* is made the current line.

Options

–a Appends output to the existing **tags** file rather than overwriting the file.

–B Produces a **tags** file that searches backward from the current position to find the pattern matching the tag.

–F Searches for tag patterns in the forward direction. This is the default.

–f Generates a file named **tagfile** rather than the default **tags** file.

–w Suppresses warning messages.

–x Produces a report on the standard output. The report gives the definition name, the line number of where it appears in the file, the name of the file in which it appears, and the text of that line. **ctags** arranges this output in columns and sorts it in order by tag name according to the current locale's collation sequence. This option does not produce a **tags** file.

Localization

ctags uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLECT**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_TIME**

See Appendix F for more information.

Files

ctags uses the following file:

tags Output tags file

Usage Notes

1. It can be difficult to recognize a function definition in C source code. Because **ctags** does not know which C preprocessor symbols are defined, there may be some misplaced function definition information if sections of code within **#if...#endif** are not complete blocks.
2. **ctags** invokes the **sort** internally.
3. **ctags** makes special provision for the **main()** function, which may occur in several C source files. The **tags** file contains an entry for the first **main()** routine found. For all occurrences of **main()**, including the first, the **tags** file contains an entry for `M`*name*, where name is the name of the input source file, with the **.c** suffix and any leading pathname components removed. For example, a **tags** file created for a C source code file named **foo.c** would contain an entry for **Mfoo**, which represents the **main()** routine in **foo.c**).
4. **ctags** uses **sort** to sort the file by tag name, according to the POSIX locale's collation sequence.

Exit Values

`0` Successful completion

`1` Failure due to any of the following:
- Unknown command-line option
- Cannot create the output file
- Cannot open the output file
- One of the input files was unrecognized

Portability

POSIX.2, X/Open Portability Guide, 4.2BSD and higher.

This utility only understands characters from the POSIX locale.

The **–B**, **–F**, and **–w** options are extensions to the POSIX and XPG standards.

Related Information

more, **sort**, **vi**

See the **tags** file format description in Appendix H.

---

cu — Call up another system (stub only)

Format

cu [**–dehot**] [**–l** *device_name*] [**–s** *speed*] [*system_name* | *phone_num*]
cu –n [**–dehot**] [**–l** *device_name*] [**–s** *speed*]

Description

cu connects to remote systems specified in the UUCP configuration file. You can use it for simple terminal connections, or to do simple file transfer with no error checking.

cu is recognized, but its functions are disabled. Traditionally, it is used for simple terminal connections to remote systems specified in the UUCP configuration file. **cu** requires a direct connection (such as with a modem) to the remote system, but this is not supported by OS/390.

---

cut — Cut out selected fields from each line of a file

Format

cut –b *list* [**–n**] [*file*...]
cut –c *list* [*file*...]
cut –f *list* [**–d** *char*] [**–s**] [*file*...]

Description

cut reads input from files, each specified with the *file* argument, and selectively copies sections of the input lines to the standard output (**stdout**). If you do not specify any *file*, or if you specify a file named –, **cut** reads from standard input (**stdin**).

Options

–b *list*
Invokes byte position mode. After this comes a list of the byte positions you want to display. This list may contain multiple byte positions, separated by commas (**,**) or blanks or ranges of positions separated by dashes (–). Since the list must be a single argument, shell quoting is necessary if you use

blanks. You can combine these to allow selection of any byte positions of the input.

Attention: When using the **–b** option with doublebyte characters, you should also specify the **–n** option to ensure that entire characters are displayed. If you do not specify the **–n** option, **cut** simply assumes that the low byte of a range is the first byte of a character and that the high byte of a range is the last byte of a doublebyte character, possibility resulting in the misinterpretation of the characters represented by those byte positions.

–c *list*
Invokes character-position mode. After this comes a list of character positions to retain in the output. This list can contain many character positions, separated by commas (**,**) or blanks or ranges of positions separated by a dash (–). Since the list must be a single argument, shell quoting is necessary if you use blanks. You can combine these to allow selection of any character positions of the input.

–d *char*
Specifies *char* as the character that separates fields in the input data; by default, this is the horizontal tab.

–f *list*
Invokes field delimiter mode. After this comes a list of the fields you want to display. You specify ranges of fields and multiple field numbers in the same way you specify ranges of character positions and multiple character positions in **–c** mode.

–n Does not split characters. If the low byte in a selected range is not the first byte of a character, **cut** extends the range downward to include the entire character; if the high byte in a selected range is not the last byte of a character, **cut** limits the range to include only the last entire character before the high byte selected. If **–n** is selected, **cut** does not list ranges that do not encompass an entire character, and these ranges do not cause an error.

–s Does not display lines that do not contain a field separator character. Normally, **cut** displays lines that do not contain a field separator character in their entirety.

Example

```
cd /bin
ls –al | cut –c 42–48,54–66
```

prints a directory listing containing file creation dates and filenames of files in the working directory.

Localization

cut uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion

1 Failure due to any of the following:

- Cannot open the input file
- Out of memory

2 Failure due to any of the following:

- An incorrect command-line argument
- You did not specify any of **–b**, **–c**, or **–f**
- You omitted the *list* argument
- Badly formed *list* argument

Portability

POSIX.2, X/Open Portability Guide, UNIX System V.

Related Information

paste, **uname**

---

date — Display the date and time

Format

date [**–cu**] [**+***format*]

Description

date displays the operating system's idea of the current date and time. The following example shows the default format of the date:

```
Wed Feb 26 14:01:43 EST 1986
```

Options

date accepts the following options:

–c Displays the date and displays the time according to Greenwich Mean Time (Coordinated Universal Time) using *CUT* as the time zone name.

–u Displays the date and displays the time according to Greenwich Mean Time (Coordinated Universal Time) using *GMT* as the time zone name.

If the argument to **date** begins with a + character, **date** uses *format* to display the date. **date** writes all characters in *format*, with the exception of the % and the character that immediately follows it, directly to the standard output. After **date** exhausts the *format* string, it outputs a newline character. The % character introduces a special format field similar to the **printf()** function in the C library. **date** supports the following field descriptors:

%A The full weekday name (for example, Sunday).

%a The three-letter abbreviation for the weekday (for example, Sun).

%B The full month name (for example, February).

%b The three-letter abbreviation for the month name (for example, Feb).

%C The first two digits of the year (00 to 99).

%c The local representation of the date and time (see %D and %T).

%D The date in the form *mm*/*dd*/*yy*.

%d The two-digit day of the month as a number (01 to 31).

%e The day of the month in a two-character, right-justified, blank-filled field.

%H The two-digit hour (00 to 23).

%h The three-letter abbreviation for the month (for example, June).

%I The hour in the 12-hour clock representation (01 to 12).

%j The numeric day of the year (001 to 366).

%M The minute (00 to 59).

%m The month number (01 to 12).

%n The newline character.

%p The local equivalent of a.m. or p.m.

%r The time in a.m.–p.m. notation (11:53:29 a.m.).

%S The seconds (00 to 61). There is an allowance for two leap seconds.

%T The time (14:53:29).

%t A tab character.

%U The week number in the year, with Sunday being the first day of the week (00 to 53).

%W The week number in the year, with Monday being the first day of the week (00 to 53).

%w The weekday number, with Sunday being 0.

%X The local time representation (see %T).

%x The local date representation (see %D).

%Y The year.

%y The two-digit year.

%Z The time zone name (for example, EDT).

%% A percent-sign character.

The **date** command also supports the following modified field descriptors to indicate a different format as specified by the locale indicated by **LC_TIME**. If the current locale does not support a modified descriptor, **date** uses the unmodified field descriptor value.

%EC
The name of the base year (period) in the current locale's alternate representation.

%Ec
The current locale's alternate date and time representation.

%Ex
The current locale's alternate date representation.

%EY
The full alternate year representation.

%Ey
The offset from %EC (year only) in the current locale's alternate representation.

%Od
The day of the month using the current locale's alternate numeric symbols.

%Oe
The day of the month using the current locale's alternate numeric symbols.

%OH
The hour (24-hour clock) using the current locale's alternate numeric symbols.

%OI
The hour (12-hour clock) using the current locale's alternate numeric symbols.

%OM
The minutes using the current locale's alternate numeric symbols.

%Om
The month using the current locale's alternate numeric symbols.

%OS
The seconds using the current locale's alternate numeric symbols.

%OU
The week number of the year (0–53) (with Sunday as the first day of the week) using the current locale's alternate numeric symbols.

%OW
The week number of the year (0–53) (with Monday as the first day of the week) using the current locale's alternate numeric symbols.

%Ow
The weekday as a number using the current locale's alternate numeric symbols (Sunday=0).

%Oy
The year (offset from %C) using the current locale's alternate numeric symbols.

Example

The command:

```
date '+%a %b %e %T %Z %Y'
```

produces the date in the default format—as shown at the start of this command description.

Environment Variable

date uses the following environment variables:

TZ
Gives the time zone for **date** to use when displaying the times. This is ignored if you specify either the **–c** or the **–u** option.

For information on setting the local time zone with the **TZ** environment variable, see Appendix I.

Localization

date uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_TIME**
- **NLSPATH**

See Appendix F for more information.

Exit Values

`0` Successful completion

`>0`
Failure due to any of the following:

- An incorrect command-line option
- Too many arguments on the command line
- A bad date conversion
- A formatted date that was too long
- You do not have permission to set the date

Messages

Possible error messages include:

Bad format character *x*
A character following "%" in the *format* string was not in the list of field descriptors.

No permission to set date
The system has denied you the right to set the date.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The **–c** option is an extension of the POSIX standard.

Related Information

touch

Appendix I also explains how to set the local time zone with the **TZ** environment variable.

---

dbx — Use the debugger

Format

dbx [**–a** *ProcessID*] [**–A** *ProcessID*] [**–c** *file*] [**–d** *NestingDepth*] [**–f**] [**–I** *directory*] [**–r**] [**–u**] [*ObjectFile*[*ProgramArguments*]]

Description

dbx provides an environment to debug and run programs. **dbx** provides a symbolic debug program for C programs, allowing you to carry out operations such as the following:

- Examine object files
- Provide a controlled environment for running a program
- Set breakpoints at selected statements or run the program one line at a time
- Debug using symbolic variables and display them in their correct format

The *ObjectFile* argument is an object (executable) file produced by a compiler. When compiling your program to produce the information the **dbx** command needs, use the **–g** (generate symbol table) option on the **c89/cc/c++** command.

Note: If the object file is not compiled with the **–g** option on the **c89/cc/c++** command, or if the user compiles with optimization, the capabilities of the **dbx** command are limited.

When the **dbx** command is started, it checks for a **.dbxsetup** file in the user's working directory. If the file is not found, **dbx** checks the user's **$HOME** directory. If a **.dbxsetup** file exists, its subcommands run before most of dbx's initialization is complete. This allows tailoring of dbx's operational behavior before the debug program is loaded via **set** subcommands. Use an editor to create a **.dbxsetup** file.

Also on start-up, **dbx** checks for a **.dbxinit** file in the user's working directory. If the file is not found, **dbx** checks the **$HOME** directory. If a **.dbxinit** file exists, its subcommands run at the beginning of the debug sesssion. To create a **.dbxinit** file, use an editor.

You can use the **man** command to view manual descriptions of **dbx** subcommands. To do this, you must prefix all subcommands with **dbx**. For example, to view a description of the **dbx alias** subcommand, you would enter the following:

```
man dbxalias
```

Options

–a *ProcessID*
: Attaches the debug program to a running process. The debug program becomes active as soon as the process wakes up. To attach the debug program, you need authority to use the **kill** command on this process.

–A *ProcessID*
: Reattaches the debug program to a running process that is already being debugged by **dbx**. Use this option to reattach a child process that was created when a debugged parent process did a fork while multiprocess debugging mode was active. To reattach to the debug program, you need authority to use the **kill** command on this process.

–c *file*
: Runs the **dbx** subcommands in the file before reading from standard input.

–d *NestingDepth*
Sets the limit for the nesting of program blocks. The default nesting depth limit is 25.

–f Starts **dbx** in lazy reading mode. After the **dbx** session is started, only the required symbol table information is read. No session has been started. Using this flag speeds up **dbx** startup (especially for debugging large executables) and reduces memory usage by **dbx**.

–I *directory*
Includes *directory* in the list of directories searched for source files. The default is to look for source files in the working directory and in the directory where the object file is located. The search path is also set with the **use** subcommand.

–r Runs the object file immediately. If it ends successfully, the **dbx** debug program is exited. Otherwise, the debug program is entered and the reason for termination is reported.

–u Causes the **dbx** command to prepend filename symbols with an **@**. This flag reduces the possibility of ambiguous symbol names.

Note: Unless **–r** is specified, the **dbx** command prompts the user and waits for a command. However, you can specify program arguments on **dbx** even when **–r** is not used. For example:

```
dbx myprog p1 p2 p3
```

Expression Handling

Specify expressions in **dbx** with a subset of C syntax. A prefix ***** or a postfix denotes indirection. Specify portions of an array by separating the lower and upper bounds with **..** (two periods).

Use [] (square brackets) or () (parentheses) to enclose array subscripts. Use the field reference operator **.** (period) with pointers and records.

Note: The field reference operator **.** (period) makes the C operator → unnecessary (although it is supported).

When displaying variables and expressions, the **dbx** command resolves names first using the *static scope* of the current function. The *dynamic scope* is used if the name is not defined in the first scope. If static and dynamic searches do not yield a result, an arbitrary symbol is chosen and the system prints the message (using *Module.Variable*). The *Module.Variable* construction is the name of an identifier qualified with a block name. Override the name resolution procedure by qualifying an identifier with a block name. Source files are treated as modules named by the filename without the language suffix (such as the **.c** suffix on a C language program).

The **dbx** command debug program checks types of expressions. Override types of expressions by using *TypeName (Expression).* When there is no corresponding named type, use the **&***TypeName* special construct to represent a pointer to the named type. Represent a pointer to **enum**, **struct**, or **union** tag with the **$$***TagName* construct.

A condition can be any **dbx** expression that evaluates to a true or false value. This pertains to four **dbx** subcommands: **stop**, **stopi**, **trace**, and **tracei**.

The following operators are valid in expressions:

Expression	Operators
exp (exponentiation) Algebraic	**+**, –, *****, / (floating), **div** (integral), **mod**
Bitwise	–, **|**, **bitand**, **xor**, ^, <<, >>
Logical	**or**, **and**, **not**, **||**, &&;
Comparison	<, >, **<=**, **>=**, <> or **!=**, **=** or **==**
Other	**sizeof**

Files

a.out The object file; **a.out** contains object code. This is the default program object if nothing is specified for the **dbx** command.

.dbxinit Contains initial commands.

Related Information

c89/cc/c++

? subcommand for dbx: Search backward for a pattern

Format

? [*RegularExpression*]

Description

The **?** subcommand searches backward in the current source file for the pattern specified by the *RegularExpression* argument. Entering the **?** subcommand with no arguments causes **dbx** to search backward for the previous regular expression.

Usage Note

The **?** subcommand can be run only while the **dbx** debug program is running.

Examples

1. To search backward in the current source file for the letter z, enter:

   ```
   ?z
   ```

2. To repeat the previous search, enter:

   ```
   ?
   ```

Related Information

The / (search) subcommand.

/ subcommand for dbx: Search forward for a pattern

Format

/ [*RegularExpression*]

Description

The **/** subcommand searches forward in the current source file for the pattern specified by the *RegularExpression* argument. Entering the **/** subcommand with no arguments causes **dbx** to search forward for the previous regular expression.

Usage Note

The **/** subcommand can be run only while the **dbx** debug program is running.

Examples

1. To search forward in the current source file for the number 12, enter:

   ```
   / 12
   ```

2. To repeat the previous search, enter:

   ```
   /
   ```

Related Information

The **?** (search) subcommand.

alias subcommand for dbx: Display and assign subcommand aliases

Format

alias [*name*] [*string*]

Description

The **alias** subcommand creates aliases for **dbx** subcommands. The *name* argument is the alias being created. The *string* argument is a series of **dbx** subcommands that, after the execution of this subcommand, can be referred to by *name*. If the **alias** subcommand is used without aliases, it displays all current aliases.

Usage Note

The **alias** subcommand can be run only while the **dbx** debug program is running.

Examples

1. To set **tracef1** to be an alias for **trace** in **f1**, enter:

   ```
   alias tracef1 "trace in f1"
   ```

2. To define a **stopf** alias with *file* and *line* arguments to allow shorthand for setting a breakpoint within a file, enter:

   ```
   alias stopf(file, line) "stop at \"file\":line"
   ```

args subcommand for dbx: Display program arguments

Format

args

Description

The **args** subcommand displays the argument count and a list of arguments that are passed to the user's program when **dbx** starts debugging the program.

Usage Note

The **args** subcommand can be run only while the **dbx** debug program is running.

Examples

To display the current arguments, enter:

```
args
```

Related Information

The **rerun** and **run** subcommands.

assign subcommand for dbx: Assign a value to a variable

Format

assign [*variable=expression*]

Description

The **assign** subcommand assigns the value specified by the *expression* argument to the variable specified by the *variable* argument.

Usage Notes

1. The **assign** subcommand can be run only while the **dbx** debug program is running.
2. Functions cannot be specified with the *expression* argument.

Examples

1. To assign the value `5` to a variable *x*, enter:

   ```
   assign x = 5
   ```

2. To assign the value of a variable *y* to a variable *x*, enter:

   ```
   assign x = y
   ```

3. To assign a value to a storage location, enter:

   ```
   assign 0x02e0f7f0 = 0xff
   ```

4. To assign a value to a register, enter:

   ```
   assign $r7 = 123
   ```

5. To change the exit_status of a specific thread, enter:

```
assign $t1.exit_status=&$void(0x2d95840);
```

case subcommand for dbx: Change how dbx interprets symbols

Format

case [**default** | **mixed** | **lower** | **upper**]

Description

The **case** subcommand changes how the **dbx** debug program interprets symbols. Use **case** if a symbol needs to be interpreted in a way not consistent with the default behavior.

Entering **case** with no parameters displays the current case mode.

Options

default Varies with the current language.
mixed Causes symbols to be interpreted as they actually appear.
lower Causes symbols to be interpreted as lowercase.
upper Causes symbols to be interpreted as uppercase.

Usage Note

The **case** subcommand can be run only while the **dbx** debug program is running.

Examples

1. To instruct **dbx** to interpret symbols as they actually appear, enter:

   ```
   case mixed
   ```

2. To instruct **dbx** to interpret symbols as uppercase, enter:

   ```
   case upper
   ```

catch subcommand for dbx: Start trapping a signal

Format

catch [*signalnumber* | *signalname*]

Description

The **catch** subcommand starts the trapping of a specified signal before that signal is sent to the application program. This subcommand is useful when the application program being debugged handles such signals as interrupts. The signal to be trapped can be specified by number or by name using either the *signalnumber* or the *signalname* argument, respectively. Signal names are case-insensitive, and the **SIG** prefix is optional. All signals are caught by default except the **SIGDUMP**, **SIGHUP**, **SIGCHLD**, **SIGALRM**, and **SIGKILL** signals. If no arguments are specified, the current list of signals to be caught is displayed.

Usage Note

The **catch** subcommand can be run only while the **dbx** debug program is running.

Examples

1. To display a current list of signals to be caught by **dbx**, enter:

   ```
   catch
   ```

2. To trap signal **SIGALRM**, enter:

   ```
   catch SIGALRM
   ```

   or:

   ```
   catch ALRM
   ```

   or:

   ```
   catch 14
   ```

Related Information

The **ignore** subcommand.

---

clear subcommand for dbx: Remove all stops at a given source line

Format

clear [*sourceline*]

Description

The **clear** subcommand removes all stops at a given source line.

The *sourceline* argument can be specified in two formats:

- As an integer
- As a filename string followed by a **:** (colon) and an integer

Usage Note

The **clear** subcommand can be run only while the **dbx** debug program is running.

Example

To remove breakpoints set at line 19, enter:

```
clear 19
```

Related Information

The **cleari** and **delete** subcommands.

cleari subcommand for dbx: Remove all breakpoints at an address

Format

cleari *address*

Description

The **cleari** subcommand clears all the breakpoints at the address specified by the *address* argument.

Usage Note

The **cleari** subcommand can be run only while the **dbx** debug program is running.

Examples

1. To remove a breakpoint set at address 0X100001B4, enter:

   ```
   cleari 0x100001b4
   ```

2. To remove a breakpoint set at the address of the **main()** procedure, enter:

   ```
   cleari Created by ActiveSystems 10/25/96 Entity not defined.
   ```

Related Information

The **clear** and **delete** subcommands.

condition subcommand for dbx: Display a list of active condition variables

Format

condition [*number ...*]
condition wait
condition nowait

Description

The **condition** subcommand displays a list of active condition variables for the application program. All active condition variables are listed unless you use the *number* parameter to specify the condition variables you want listed. You can also select condition variables with or without waiters by using the **wait** or **nowait** options.

In order to capture the condition variables, **dbx** must be debugging your program before the condition variable is created. You must have coded your application in one of the following ways:

- Add the following line at the top of the C program:

  ```
  #pragma runopts(TEST(ALL))
  ```

  Or:

- Code an assembler program, CEEUOPT, to invoke the CEEXOPT macro, which specifies TEST(ALL). For examples of how to code this program, see *OS/390 C/C++ Programming Guide* .

Usage Note

The **condition** subcommand can be run only while the **dbx** debug program is running.

Examples

1. To display all condition variables, enter:

   ```
   condition
   ```

2. To display condition variables number 1 and number 4, enter:

   ```
   condition 1 4
   ```

3. To display all condition variables with waiters, enter:

   ```
   condition wait
   ```

4. To display all condition variables without waiters, enter:

   ```
   condition nowait
   ```

---

cont subcommand for dbx: Continue program execution

Format

cont [*signalnumber* | *signalname*]

Description

The **cont** subcommand continues the execution of the program from the current stopping point until either the program finishes, another breakpoint is reached, a signal is received that is trapped by the **dbx** command, or an event occurs (such as a fork, an exec, or a program abend).

If a signal is specified, either by the number specified in the *signalnumber* argument or by the name specified in the *signalname* argument, the program continues as if that signal had been received by the focus thread.

If a signal is not specified, the **dbx** debug program variable **$sigblock** is set, and a signal caused the debugged program to stop, then the program resumes execution. If a signal is not specified, the **dbx** debug program variable **$sigblock** is not set, and a signal caused the debugged program to stop, then typing in the **cont** command with no signal causes the program to continue as if it had received the original signal.

Signal names are not case-sensitive, and the **SIG** prefix is optional. If no signal is specified, the program continues as if it had not been stopped.

Usage Note

The **cont** subcommand can be run only while the **dbx** debug program is running.

Examples

1. To continue program execution from the current stopping point, enter:

   ```
   cont
   ```

2. To continue program execution as though it had received the signal **SIGQUIT**, enter:

   ```
   cont SIGQUIT
   ```

Related Information

The **step**, **next**, **goto**, and **skip** subcommands.

---

delete subcommand for dbx: Remove traces and stops

Format

delete [**all** | *number...*]

Description

The **delete** subcommand removes traces and stops from the program. You can specify the traces and stops to be removed through the *number* arguments, or you can remove all traces and stops by using the **all** option. To display the numbers associated by the **dbx** debug program with a trace or stop, use the **status** subcommand.

Option

all Removes all traces and stops.

Usage Note

The **delete** subcommand can be run only while the **dbx** debug program is running.

Examples

1. To remove all traces and stops from the program, enter:

   ```
   delete all
   ```

2. To remove traces and stops for event number 4, enter:

   ```
   delete 4
   ```

Related Information

The **status**, **clear**, and **cleari** subcommands.

---

detach subcommand for dbx: Continue program execution without dbx control

Format

detach [*signalnumber* | *signalname*]

Description

The **detach** subcommand continues the execution of a program from the current stopping point without control of **dbx**.

If a signal is specified, either by the number specified in the *signalnumber* argument or by the name specified in the *signalname* argument, the program continues without **dbx** control as if that signal had been received by the focus thread. If the signal is not specified, the program continues with no signal and without **dbx** control.

Signal names are not case-sensitive, and the SIG prefix is optional. If no signal is specified, the program continues without **dbx** control as if it had not been stopped.

Usage

The **detach** subcommand can be run only while the **dbx** debug program is running.

Examples

1. To continue program execution from the current stopping point without **dbx** in control, enter:

   ```
   detach
   ```

2. To continue program execution without **dbx** control as though it had received the signal SIGQUIT, enter:

   ```
   detach SIGQUIT
   ```

Related Information

The **cont** subcommand.

---

display memory subcommand for dbx: Display the contents of memory

Format

address, address/ [**b** | **Bd** | **Bf** | **Bq** | **c** | **d** | **D** | **f** | **g** | **h** | **i** | **I** | **ld**| **lo** |**lx** | **o** | **O** | **q** | **s** | **S** | **W** | **x** | **X**] [>**file**]

address/ [**count**] [**b** | **Bd** | **Bf** | **Bq** | **c** | **d** | **D** | **f** | **g** | **h** | **i** | **I** | **ld** | **lo** | **lx** | **o** | **O** | **q** | **s** | **S** | **o** | **W** | **X**] [>**file**]

Description

The **display memory** subcommand, which does not have a keyword to start the command, displays a portion of memory controlled by the following factors:

- The range of memory displayed is controlled by specifying:
  - Two *address* arguments, in which case all lines between those two addresses are displayed (**address**/**address**), or

—- One **address** argument, where the display starts, and **count**, which determines the number of lines displayed from **address** (**address**/**count**).

. Used in place of the first *address* argument, this displays from the point where you left off (see example 3 on page 170).

Specify symbolic addresses by preceding the name with an **&** (ampersand). Addresses can be expressions made up of other addresses and the operators +, –, and ***** (indirection). Any expression enclosed in parentheses is interpreted as an address.

- The format in which the memory is displayed is controlled by the *mode* argument. The default for the *mode* argument is the current mode. The initial value of *mode* is **X**. The possible modes include:

  b Prints a byte in octal
  Bd Prints a double-precision binary real number
  Bf Prints a single-precision binary real number
  Bq Prints a long double-precision binary real number
  c Prints a byte as a character
  d Prints a short word in decimal
  D Prints a long word in decimal
  f Prints a single-precision real number
  g Prints a double-precision real number
  h Prints a byte in hexadecimal
  i Prints the machine instruction
  l Prints a wint_t character
  ld Prints a long long in decimal
  lo Prints a long long in octal
  lx Prints prints a long long in hexidecimal
  o Prints a short word in octal
  O Prints a long word in octal
  q Prints a long double precision floating-point number
  s Prints a string of characters terminated by a null byte
  S Prints a wchar_t string
  W Prints a wint_t string
  x Prints a short word in hexadecimal
  X Prints a long word in hexadecimal

Option

>**file** Redirects output to the specified file.

Usage Note

The **display memory** subcommand can be run only while the **dbx** debug program is running.

Examples

1. To display one long word of memory content in hexadecimal starting at the address 0X3FFFE460, enter:

   ```
   0x3fffe460 / x
   ```

2. To display 2 bytes of memory content as characters starting at the address of variable *y*, enter:

   ```
   &y/2c
   ```

3. To display from the point where you left off, when using **.** (period) in place of one of the addresses, enter:

```
0x100 / 2              which displays 2 words starting at x'100'
```

followed by:

```
. / 3                  which displays 3 words starting at x'108'
```

down subcommand for dbx: Move the current function down the stack

Format

down [*count*]

Description

The **down** subcommand moves the current function down the stack *count* number of levels. The current function is used for resolving names. The default for the *count* argument is 1.

Usage Note

The **down** subcommand can be run only while the **dbx** debug program is running.

Examples

1. To move one level down the stack, enter:

```
down
```

2. To move three levels down the stack, enter:

```
down 3
```

Related Information

The **up** and **where** subcommands.

dump subcommand for dbx: Display the names and values of variables in a procedure

Format

dump [*procedure*] [>*file*]

Description

The **dump** subcommand displays the names and values of all variables in the specified procedure. If the *procedure* argument is **.** (dot), all active variables are displayed. If the *procedure* argument is not specified, the current procedure is used. If the >*file* option is used, the output is redirected to the specified file.

Option

>*file* **dump** output to the specified file.

Usage Note

The **dump** subcommand can be run only while the **dbx** debug program is running. **dump** redirects output to the specified file.

Examples

1. To display names and values of variables in the current procedure, enter:

   ```
   dump
   ```

2. To display names and values of variables in the **add_count** procedure, enter:

   ```
   dump add_count
   ```

3. To redirect names and values of variables in the current procedure to the **var.list** file, enter:

   ```
   dump > var.list
   ```

---

edit subcommand for dbx: Invoke an editor

Format

edit [*procedure* | *file*]

Description

The **edit** subcommand invokes an editor on the specified file. The file can be specified through the *file* argument or through the *procedure* argument (in which case the editor is invoked on the file containing that procedure). If no file is specified, the editor is invoked on the current source file. The default editor is the **ed** editor. Override the default by resetting the **EDITOR** environment variable to the name of the desired editor.

Usage Note

The **edit** subcommand can be run only while the **dbx** debug program is running.

Examples

1. To invoke an editor on the current source file, enter:

   ```
   edit
   ```

2. To invoke an editor on the **main.c** file, enter:

   ```
   edit main.c
   ```

3. To invoke an editor on the file containing the **do_count** procedure, enter:

   ```
   edit do_count
   ```

Related Information

The **ed** editor.

The **list** subcommand for the **dbx** command.

---

file subcommand for dbx: Change the current source file

Format

file [*file*]

Description

The **file** subcommand changes the current source file to the file specified by the *file* argument; it does not write to that file. If the *file* argument is not specified, the **file** subcommand displays the name of the current source file.

Usage Note

The **file** subcommand can be run only while the **dbx** debug program is running.

Examples

1. To change the current source file to the **main.c** file, enter:

   ```
   file main.c
   ```

2. To display the name of the current source file, enter:

   ```
   file
   ```

---

func subcommand for dbx: Change the current function

Format

func [*procedure*]

Description

The **func** subcommand changes the current function to the procedure or function specified by the *procedure* argument. If the *procedure* argument is not specified, the default current function is displayed. Changing the current function implicitly changes the current source file to the file containing the new function; the current scope used for name resolution is also changed.

Usage Notes

The **func** subcommand can be run only while the **dbx** debug program is running.

Examples

1. To change the current function to the **do_count** procedure, enter:

   ```
   func do_count
   ```

2. To display the name of the current function, enter:

   ```
   func
   ```

goto subcommand for dbx: Run a specified source line

Format

goto *sourceline*

Description

The **goto** subcommand causes the specified source line to be run next. Normally, the source line must be in the same function as the current source line. To override this restriction, use the **set** subcommand with the **$unsafegoto** variable.

Usage Note

The **goto** subcommand can be run only while the **dbx** debug program is running.

Example

To change the next line to be executed to line 6, enter:

```
goto 6
```

Related Information

The **cont**, **gotoi**, and **set** subcommands.

gotoi subcommand for dbx: Change the program counter address

Format

gotoi *address*

Description

The **gotoi** subcommand changes the program counter address to the address specified by the *address* argument.

Usage Note

The **gotoi** subcommand can be run only while the **dbx** debug program is running.

Example

To change the program counter address to address 0X100002B4, enter:

```
gotoi 0x100002b4
```

Related Information

The **goto** subcommand.

help subcommand for dbx: Display a subcommand synopsis

Format

help [*subcommand*] [*topic*]

Description

The **help** subcommand displays a synopsis of common **dbx**

subcommands.

Usage Note

The **help** subcommand can be run only while the **dbx** debug program is running.

Examples

To obtain a synopsis of common **dbx** subcommands, enter one of the following:

```
help
help subcommand
help topic
```

The **help** subcommand with no arguments lists available **dbx** subcommands and topics.

help *subcommand*, where *subcommand* is one of the **dbx** subcommands, displays a synopsis and brief description of the subcommand. **help** *topic*—where *topic* is execution, expression, files, machine, scope, usage, or variables—displays a synopsis and brief description of the topic.

history subcommand for dbx: Display commands in a history list

Format

history

Usage Note

The **history** subcommand displays the commands in the history list. As each command is entered, it is appended to the history list. A mechanism for history substitution is provided through the exclamation (**!**) operator. The allowable forms are **!!** for a previous command, *!n* for the *n*th command, and **!***string* for the previous command that starts with *string*. The number of commands retained and displayed is controlled by the **dbx** internal variable **$historywindow**.

ignore subcommand for dbx: Stop trapping a signal

Format

ignore [*signalnumber* | *signalname*]

Description

The **ignore** subcommand stops the trapping of a specified signal before that signal is sent to the program. This subcommand is useful when the program being debugged handles such signals as interrupts.

The signal to be trapped can be specified by:

- Number, with the *signalnumber* argument
- Name, with the *signalname* argument

Signal names are not case-sensitive. The **SIG** prefix is optional.

If neither the *signalnumber* nor the *signalname* argument is specified, all signals except the **SIGDUMP**, **SIGHUP**, **SIGCHLD**, **SIGALRM**, and **SIGKILL** signals are ignored by default. The **dbx** debug program cannot catch **SIGKILL** or **SIGDUMP**. If no arguments are specified, the list of currently ignored signals is displayed.

Usage Note

The **ignore** subcommand can be run only while the **dbx** debug program is running.

Examples

To cause **dbx** to ignore alarm clock timeout signals sent to the program, enter:

```
ignore sigalrm
```

or:

```
ignore alrm
```

or:

```
ignore 14
```

Related Information

The **catch** subcommand.

---

list subcommand for dbx: Display lines of the current source file

Format

list [*procedure* | *SourcelineExpression*] [,*SourcelineExpression*]

Description

The **list** subcommand displays a specified number of lines in the source file. The number of lines displayed are specified in one of two ways:

- By specifying a procedure using the *procedure* argument. In this case, the **list** subcommand displays lines before the first executable line of source in the specified procedure and until the list window is filled.

- By specifying a starting and ending source line number using the *SourcelineExpression* argument. Use the current filename or source filename if specified.

  The *SourcelineExpression* argument should consist of a valid line number followed by an optional + or – and an integer. In addition, a *SourcelineExpression* of **$** can be used to denote the current line number, and a *SourcelineExpression* of **@** can be used to denote the next line number to be listed.

  All lines from the first line number specified to the second line number specified, inclusive, are then displayed, provided these lines fit in the list window.

  If the second source line is omitted, ten lines are printed, beginning with the line number specified in the *SourcelineExpression* argument.

If the **list** subcommand is used without arguments, the default number of lines are printed, beginning with the current source line. The default is `10`.

To change the number of lines to list by default, set the special debug program variable, **$listwindow**, to the number of lines you want. Initially, **$listwindow** is set to `10`.

Usage Note

The **list** subcommand can be run only while the **dbx** debug program is running.

Examples

1. To list the lines 1 through 10 in the current file, enter:

   ```
   list 1,10
   ```

2. To list 10, or **$listwindow**, lines around the first executable line in the **main** procedure, enter:

   ```
   list main
   ```

3. To list 11 lines around the current line, enter:

   ```
   list $-5,$+5
   ```

Related Information

The **edit**, **listi**, **move**, and **set** subcommands.

---

listfiles subcommand for dbx: Display the list of source files

Format

listfiles [*loadmap-index*]

Description

The **listfiles** subcommand displays the list of files associated with each module in the load map.

If the **listfiles** subcommand is used without arguments, the files for every module in the load map will be listed.

Usage Notes

The **listfiles** subcommand can be run only while the **dbx** debug program is running.

Examples

1. To list all files in all modules, enter:

   ```
   listfiles
   ```

2. To list the files only for module with index 0 in the loadmap, enter:

   ```
   listfiles 0
   ```

Related Information

The **listfuncs** and **map** subcommands.

---

listfuncs subcommand for dbx: Display the list of functions

Format

listfuncs [*filename*]

Description

The **listfuncs** subcommand displays a list of functions associated with each file in the program.

If the listfuncs subcommand is used without arguments, the function for every file in the program will be listed.

Usage Notes

The **listfuncs** subcommand can be run only while the **dbx** debug program is running.

Examples

1. To list all functions in all files, enter:

   ```
   listfuncs
   ```

2. To list the function only for file **mypgm.c**, enter:

   ```
   listfuncs mypgm.c
   ```

Related Information

The **func** subcommand.

---

listi subcommand for dbx: List instructions from the program

Format

listi [*procedure* | **at** | *sourceline* | *address*] [,*address*]

Description

The **listi** subcommand displays a specified set of instructions from the source file. The instructions displayed are specified by:

- Providing the *procedure* argument, in which case the **listi** subcommand lists instructions from the beginning of *procedure* until the list window is filled.
- Using the **at***sourceline* option, in which case the **listi** subcommand displays instructions beginning at the specified source line and continuing until the list window is filled.
- Specifying a beginning and ending address using the *address* arguments, in which case all instructions between the two addresses, inclusive, are displayed.

If the **listi** subcommand is used without options or arguments, the next **$listwindow** instructions are displayed. To change the current size of the list window, use the **set $listwindow**=*value* subcommand.

Option

at *sourceline* Specifies a starting source line for the listing.

Usage Note

The **listi** subcommand can be run only while the **dbx** debug program is running.

Examples

1. To list the next 10, or **$listwindow**, instructions, enter:

   ```
   listi
   ```

2. To list the machine instructions beginning at source line 10, enter:

   ```
   listi at 10
   ```

3. To list the instructions between addresses 0X10000400 and 0X10000420, enter:

   ```
   listi 0x10000400, 0x10000420
   ```

Related Information

The **list** and **set** subcommands.

---

map subcommand for dbx: Display load characteristics

Format

map [>*file*]

Description

The **map** subcommand displays characteristics for each loaded portion of the program. This information includes the name, text origin, text length, text end, text subpool, data origin, data length, data subpool, and file descriptor for each loaded module. The data origin, data length, data subpool, and file descriptor do not contain meaningful information.

Option

>*file* Redirects output to the specified file.

Usage Note

The **map** subcommand can be run only while the **dbx** debug program is running.

Example

To examine the characteristics of the loaded portions of the application, enter:

```
map
```

---

move subcommand for dbx: Change the next line to be displayed

Format

move *sourceline*

Description

The **move** subcommand changes the next line to be displayed to the line specified by the *sourceline* argument. This subcommand changes the value of the **@** variable.

Usage Note

The **move** subcommand can be run only while the **dbx** debug program is running.

Example

To change the next line to be listed to line 12, enter:

```
move 12
```

Related Information

The **list** subcommand.

---

multproc subcommand for dbx: Enable or disable multiprocess debugging

Format

multproc
multproc [**on**]
multproc [**off**]

Description

The **multproc** subcommand enables or disables multiprocess debugging. Multiprocess debugging is disabled when the **dbx** debug program is started. If no options are specified, the **multproc** subcommand returns the current status of multiprocess debugging.

Options

on Enables multiprocess debugging.
off Disables multiprocess debugging.

Usage Note

The **multproc** subcommand can be run only while the **dbx** debug program is running.

Examples

1. To check the current status of multiprocess debugging, enter:

   ```
   multproc
   ```

2. To enable multiprocess debugging, enter:

   ```
   multproc on
   ```

3. To disable multiprocess debugging, enter:

   ```
   multproc off
   ```

Related Information

The **fork()** function.

---

mutex subcommand for dbx: Display a list of active mutex objects

Format

mutex [*number ...*]
mutex lock
mutex unlock
mutex wait
mutex nowait

Description

The **mutex** subcommand displays a list of active mutex objects for the application program. All active mutex objects are listed unless you use the *number* parameter to specify the mutex objects you want listed. You can also select only locked or unlocked mutexes, or mutexes with or without waiters, by using the **lock**, **unlock**, **wait**, or **nowait** options.

In order to capture the mutex variables, **dbx** must be debugging your program before the mutex variable is created. You must have coded your application in one of the following ways:

- Add the following line at the top of the C program:

  ```
  #pragma runopts(TEST(ALL))
  ```

  Or:

- Code an assembler program, CEEUOPT, to invoke the CEEXOPT macro, which specifies TEST(ALL). For examples of how to code this program, see *OS/390 C/C++ Programming Guide* .

Usage Note

The **mutex** subcommand can be run only while the **dbx** debug program is running.

Examples

1. To display all mutex objects, enter:

   ```
   mutex
   ```

2. To display mutex objects number 1 and number 4, enter:

   ```
   mutex 1 4
   ```

3. To display all locked mutex objects, enter:

   ```
   mutex lock
   ```

4. To display all unlocked mutex objects, enter:

   ```
   mutex unlock
   ```

5. To display all mutex objects with waiters, enter:

   ```
   mutex wait
   ```

6. To display all mutex objects without waiters, enter:

   ```
   mutex nowait
   ```

---

next subcommand for dbx: Run the program up to the next source line

Format

next [*number*]

Description

The **next** subcommand runs the application program up to the next source line. The *number* argument specifies the number of times the **next** subcommand runs. If the *number* argument is not specified, **next** runs once only.

Usage Notes

1. The **next** subcommand can be run only while the **dbx** debug program is running.
2. If the *$hold_next* variable is defined, **dbx** holds all threads except the focus thread during a **next** subcommand. Then **dbx** unholds the threads after the **next** subcommand finishes.

Examples

1. To continue execution up to the next source line, enter:

   ```
   next
   ```

2. To continue execution up to the third source line following the current source line, enter:

   ```
   next 3
   ```

Related Information

The **cont**, **goto**, **nexti**, and **step** subcommands.

---

nexti subcommand for dbx: Run the program up to the next machine instruction

Format

nexti [*number*]

Description

The **nexti** subcommand runs the application program up to the next instruction. The *number* argument specifies the number of times the **nexti** subcommand is to be run. If the *number* argument is not specified, **nexti** runs only once.

Usage Notes

1. The **nexti** subcommand can be run only while the **dbx** debug program is running.
2. If the *$hold_next* variable is defined, **dbx** holds all threads except the focus thread during a **nexti** subcommand. Then **dbx** unholds the threads after the **nexti** subcommand finishes.

Examples

1. To continue execution up to the next machine instruction, enter:

   ```
   nexti
   ```

2. To continue execution up to the third machine instruction following the current machine instruction, enter:

   ```
   nexti 3
   ```

Related Information

The **gotoi**, **next**, and **stepi** subcommands.

---

object subcommand for dbx: Load an object file

Format

object *filename*

Description

The **object** subcommand loads the specified object file for execution, without the overhead of reloading **dbx**.

Usage Note

The **object** subcommand can be run only while the **dbx** debug program is running.

Example

To complete debugging of the current program, and to start debugging a new program without reloading **dbx**, enter:

```
object myprog
```

---

onload subcommand for dbx: Evaluate stop/trace after dll load

Format

- **onload delete [all** | *number ...***]**
- **onload list**
- **onload stop at** *sourceline*
- **onload stop in** *procedure*
- **onload trace at** *sourceline*
- **onload trace in** *procedure*

Description

The **onload** subcommand defers building of stop or trace events until the *procedure* or *sourceline* is defined in the program **dbx** is debugging. **dbx** will evaluate the **onload** list after a DLL is loaded and generate stop/trace events if the *procedure* or *sourcefile* is now known to **dbx** after symbolics for the DLL are processed. If the *procedure* or *sourceline* is already known to **dbx**, then a normal stop or trace event will be generated and no event will be added to the **onload** list.

Usage Note

The **onload** subcommand can be run only while the **dbx** debug program is running.

Examples

To defer the building of a stop or trace event, enter:

```
onload stop in myfunc
onload stop in myclass::memfunc
onload stop in myclassvar.memfunc
onload stop in "mypgm.c":52
onload trace in myfunc
onload trace in myclass::memfunc
onload trace in myclassvar.memfunc
onload trace in "mypgm.c":52
```

Related Information

The **stop** and **trace** subcommands.

---

print subcommand for dbx: Print the value of an expression

Format

print [*expression,...*] [*(parameters)*]

Description

The **print** subcommand prints the value of a list of expressions, specified by the *expression* arguments.

Usage Note

The **print** subcommand can be run only while the **dbx** debug program is running.

Examples

1. To display the value of *x* and the value of *y* shifted left 2 bits, enter:

   ```
   print x, y << 2
   ```

2. To display a specific condition variable, enter:

   ```
   print $c1
   ```

3. To display the number of waiters for a specific mutex object, enter:

   ```
   print $m1.num_wait
   ```

4. To display the exit value of a specific thread, enter:

   ```
   print $t1.exit_status
   ```

Related Information

The **assign** and **set** subcommands.

---

prompt subcommand for dbx: Change the dbx command prompt

Format

prompt [*"string"*]

Description

The **prompt** subcommand changes the **dbx** command prompt to the string specified by the *string* argument.

Usage Note

The **prompt** subcommand can be run only while the **dbx** debug program is running.

Example

To change the prompt to dbx>, enter:

```
prompt "dbx>"
```

---

quit subcommand for dbx: End the dbx debugging session

Format

quit

Description

The **quit** subcommand ends the **dbx** debugging session.

Usage Note

The **quit** subcommand can be run only while the **dbx** debug program is running.

Example

To exit the **dbx** debug program, enter:

```
quit
```

---

readwritelock subcommand for dbx: Display a list of active read/write lock objects

Format

readwritelock [*number ...*]
readwritelock lock
readwritelock unlock
readwritelock holder
readwrite noholder

Description

The **readwritelock** subcommand displays a list of active read/write lock objects for the application program. All active read/write lock objects are listed unless you use the *number* parameter to specify the read/write lock objects you want listed. You can also select only locked or unlocked read/write locks, or read/write locks with or without holders, by using the **lock**, **unlock**, **holder**, or **noholder** options.

In order to capture the read/write lock variables, **dbx** must be debugging your program before the read/write lock variable is created. You must have coded your application in one of the following ways:

- Add the following line at the top of the C program:

  ```
  #pragma runopts(TEST(ALL))
  ```

Or:

- Code an assembler program, CEEUOPT, to invoke the CEEXOPT macro, which specifies TEST(ALL). For examples of how to code this program, see *OS/390 C/C++ Programming Guide* .

Or:

- Specify test(all) in the _CEE_RUNOPTS environment variable:

```
export _CEE_RUNOPTS="test(all)"
```

Usage Note

The **readwritelock** subcommand can be run only while the **dbx** debug program is running.

Examples

1. To display all read/write lock objects, enter:

   ```
   readwritelock
   ```

2. To display read/write lock objects number 1 and number 4, enter:

   ```
   readwritelock 1 4
   ```

3. To display all locked read/write lock objects, enter:

   ```
   readwritelock lock
   ```

4. To display all unlocked read/write lock objects, enter:

   ```
   readwritelock unlock
   ```

5. To display all read/write lock objects with holders, enter:

   ```
   readwritelock holder
   ```

6. To display all read/write lock objects without holders, enter:

   ```
   mutex noholders
   ```

---

record subcommand for dbx: Append user's commands to a file

Format

record *filename*

Description

The **record** subcommand appends the user's command lines to the specified file until a **record** command is entered with no parameters.

The **record** subcommand is started by specifying a file name on the **record** command. A second **record** command with no parameters will stop the current **record** process and close the file.

Usage Notes

The **record** subcommand can be run only while the **dbx** debug program is running.

Examples

1. To start recording the **dbx** commands to file **/tmp/mycmds**, enter:

   ```
   record /tmp/mycmds
   ```

2. To stop the previous recording to file **/tmp/mycmds**, enter:

   ```
   record
   ```

Related Information

The **source** subcommand.

---

registers subcommand for dbx: Display the value of registers

Format

registers [>*file*]

Description

The **registers** subcommand displays the values of general-purpose registers, system control registers, floating-point registers, and the current instruction register, such as the program status word (PSW) for OS/390.

- General-purpose registers are denoted by the **$r***number* variable, where the *number* argument indicates the number of the register.
- Floating-point registers are denoted by the **$fr***number* variable. By default, the floating-point registers are not displayed. To display the floating-point registers, use the **unset $noflregs dbx** subcommand.

Option

>*file* Redirects output to the specified file.

Usage Note

The **registers** subcommand can be run only while the **dbx** debug program is running.

Example

To display the registers, enter:

```
registers
```

Related Information

The **set** and **unset** subcommands.

rerun subcommand for dbx: Begin running a program with the previous arguments

Format

rerun [*arguments*] [<*file* | >*file* | **2**>*file* | >>*file* | **2**>*file* | >**&***file* | >>**&***file*]

Description

The **rerun** subcommand begins execution of the object file. The values specified with the *arguments* argument are passed as command-line arguments. If the *arguments* argument is not specified, the arguments from the last **run** or **rerun** subcommand are reused.

Options

<*file* Redirects input so that input is received from *file*.

>*file* Redirects output to *file*.

2>*file* Redirects standard error to *file*.

>>*file* Appends redirected output to *file*.

2>>*file* Appends redirected standard error to *file*.

>**&***file* Redirects output and standard error to *file*.

>>**&***file* Appends output and standard error to *file*.

Usage Note

The **rerun** subcommand can be run only while the **dbx** debug program is running.

Example

To rerun the program with the previously entered arguments, enter:

```
rerun
```

Related Information

The **run** subcommand.

return subcommand for dbx: Continue running a program until a return is reached

Format

return [*procedure*]

Description

The **return** subcommand causes the program to run until a return to the procedure specified by the *procedure* argument is reached. If the *procedure* argument is not specified, execution ceases when the current procedure returns.

Usage Note

The **return** subcommand can be run only while the **dbx** debug program is running.

Examples

1. To continue execution to the calling routine, enter:

   ```
   return
   ```

2. To continue execution to the **main** routine, enter.

   ```
   return main
   ```

---

run subcommand for dbx: Run a program

Format

run [*arguments*] [<*file* | >*file* | **2**>*file* | >>*file* | **2**>*file* | >**&***file* | >>**&***file*]

Description

The **run** subcommand begins execution of the object file. The values specified with the *arguments* argument are passed as command-line arguments.

Options

<*file* Redirects input so that input is received from *file*.

>*file* Redirects output to *file*.

2>*file*
: Redirects standard error to *file*.

>>*file*
: Appends redirected output to *file*.

2>>*file*
: Appends redirected standard error to *file*.

>**&***file*
: Redirects output and standard error to *file*.

>>**&***file*
: Appends output and standard error to *file*.

Usage Note

The **run** subcommand can be run only while the **dbx** debug program is running.

Example

To run the application with the arguments *blue* and *12*, enter:

```
run blue 12
```

Related Information

The **rerun** subcommand.

---

set subcommand for dbx: Define a value for a dbx variable

Format

set [*variable=expression*]

Description

The **set** subcommand defines a value, which is specified by the *expression* argument, for the **dbx** debug program variable, which is specified by the *variable* argument. The name of the variable should not conflict with names in the program being debugged. A variable is expanded to the corresponding expression within other commands. If the **set** subcommand is used without arguments, the variables currently set are displayed.

Variables

The following variables are set with the **set** subcommand:

$asciichars When set, any **dbx** operation that displays the value of a character will interpret the binary representation of the character as ascii.

$asciistrings
When set, any **dbx** operation that displays the value of a string will interpret the binary representation of the string as ascii.

$c<**n**> Condition variables

$catchbp Catches breakpoints during the execution of the next command.

$columns Specifies the number of columns used to display arrays. Currently this is only defined for FORTRAN.

$current Defined as a constant with the value of the focus thread.

$cv_events When set, **dbx** notifies the user but does not stop when a condition variable event is processed. The following trace information is sent to the user for the different events:

```
(dbx) cont
.
.
cv initialize, object=0x2e04567
cv wait, object=0x2e04567, mutex=0x2d04567, thid=0x0102030405060708
cv unwait, object=0x2e04567, mutex=0x2d04567, thid=0x0102030405060708
cv destroy, object=0x2e04567
.
.
```

$dll_loads Set by default. When set, **dbx** processes symbolics for dlls as they are loaded.

$dll_loadstop
Set by default. When set, **dbx** will attempt to stop the function call that caused the dll to be loaded. If the dll was loaded due to a variable reference or an explicit load, **dbx** will stop at the source line that caused the dll to be loaded.

$expandunions
Displays values of each part of variant records or unions.

$flprecision
Determines the precision in bytes of floating-point registers when used in expressions.

$fr<n> Hexidecimal floating-point register.

$frb<n> Binary floating-point register

$frame Uses the stack frame pointed to by the address designated by the value of **$frame** for doing stack tracebacks and accessing local variables.

$hexchars Prints characters as hexadecimal values.

$hexin Interprets addresses in hexadecimal.

$hexints Prints integers as hexadecimal values.

$hexstrings
Prints character pointers in hexadecimal.

$historywindow
Specifies the number of subcommands to display and retain in the history list.

$hold_next When set, **dbx** automatically holds all threads except the focus thread during the **next**, **nexti**, **step**, or **stepi** command execution. If not set, all threads resume execution and may reach the breakpoint set by the **next**, **nexti**, **step**, or **stepi** command execution.

$KERNEL_dlls
When set, **dbx** will use kernel interfaces to process and recognize DLLS. This flag is mutually exclusive with $LE_dlls.

$l<n> Read/write locks variables.

$LE_dlls When set, **dbx** will use LE interfaces to process and recognize DLLs. This flag is mutually exclusive with $KERNEL_dlls.

$LE-hookstep
When set, **dbx** will process source level stepping using an LE interface instead of machine instruction trace.

$listwindow
Specifies the number of lines to list around a function and to list when the **list** subcommand is used without parameters.

$lv_events When set, **dbx** notifies the user but does not stop when a read/write lock object event is processed. The following trace information is sent to the user for the different events:

```
(dbx) cont
.
.
lv initialize, object=0x2d04567
lv wait, object=0x2d04567, thid=0x0102030405060708
lv unwait, object=0x2d04567, thid=0x0102030405060708
lv lock, object=0x2d04567, thid=0x0102030405060708
lv unlock, object=0x2d04567, thid=0x0102030405060708
lv relock, object=0x2d04567, thid=0x0102030405060708
lv unrelock, object=0x2d04567, thid=0x0102030405060708
lv destroy, object=0x2d04567
.
.
```

$m<n> Specifies mutex variables.

$mv_events

When set, **dbx** notifies the user but does not stop when a mutex object event is processed. The following trace information is sent to the user for the different events:

```
(dbx) cont
.
.
mv initialize, object=0x2d04567
mv wait, object=0x2d04567, thid=0x0102030405060708
mv unwait, object=0x2d04567, thid=0x0102030405060708
mv lock, object=0x2d04567, thid=0x0102030405060708
mv unlock, object=0x2d04567, thid=0x0102030405060708
mv relock, object=0x2d04567, thid=0x0102030405060708
mv unrelock, object=0x2d04567, thid=0x0102030405060708
mv destroy, object=0x2d04567
.
.
```

$noargs Omits arguments from subcommands, such as **where**, **up**, **down**, and **dump**.

$noflregs Omits the display of floating-point registers from the **registers** subcommand.

$noflbregs Omits the display of binary floating-point registers from the **registers** subcommand.

$octin Interprets addresses in octal.

$octints Prints integers in octal.

$pc Program counter register.

$psw First word of the Program Status Word register.

$psw0 First word of the Program Status Word register.

$psw1 Second word of the Program Status Word register.

$r<n> General register.

$repeat Repeats the previous command if no command was entered.

$showbases

When set, **dbx** will show base class data when a derived class is printed.

$sigblock Blocks all signals from reaching the program being debugged.

$sticky_debug

When set, **dbx** will recognize sticky bit programs and DLLs in the loadmap.

$t<n> Thread variables

$tv_events When set, **dbx** notifies the user but does not stop when a thread object event is processed. The following trace information is sent to the user for the different events:

```
(dbx) cont
.
.
IPT create, thid=0x1234567890123456, stack=5200
IPT exit, thid=0x1234567890123456
tv create, thid=0x1234567890123456,  created thid=0x1234567890123422,
           stack=5200
tv created, thid=0x1234567890123456, stack=5200
tv exit, thid=0x1234567890123456
tv wait, thid=0x1234567890123456, joining thid=0x1234567890123422
tv unwait, thid=0x1234567890123456, joined thid=0x1234567890123422
```

$unsafeassign
Turns off strict type checking between the two sides of an **assign** subcommand.

$unsafebounds
Turns off subscript checking on arrays.

$unsafecall Turns off strict type checking for arguments to subroutines or function calls.

$unsafegoto
Turns off the **goto** subcommand destination checking.

$vardim Specifies the dimension length to use when printing arrays with unknown bounds. The default value is `10`. This variable is not supported.

$vectint Displays the vector register contents as integer values. (This is the default). This variable is not supported.

Usage Notes

1. The **$unsafe** variables limit the usefulness of the **dbx** debug program in detecting errors.
2. The **set** subcommand can be run only while the **dbx** debug program is running.

Examples

1. To change the default number of lines to be listed to 20, enter:

   ```
   set $listwindow=20
   ```

2. To disable type checking on the **assign** subcommand, enter:

   ```
   set $unsafeassign
   ```

Related Information

The **unset** subcommand.

sh subcommand for dbx: Pass a command to the shell for execution

Format

sh [*command*]

Description

The **sh** subcommand passes the command specified by the *command* parameter to the shell for execution. The **SHELL** environment variable determines which shell is used. The default is the **sh** shell. If no argument is specified, control is transferred to the shell.

Usage Note

The **sh** subcommand can be run only while the **dbx** debug program is running.

Examples

1. To run the **ls** command, enter:

   ```
   sh ls
   ```

2. To escape to a shell, enter:

   ```
   sh
   ```

skip subcommand for dbx: Continue from the current stopping point

Format

skip [*number*]

Description

The **skip** subcommand continues execution of the application program from the current stopping point. A number of breakpoints equal to the value of the *number* argument are skipped, and execution then ceases when the next breakpoint is reached or when the program finishes. If the *number* argument is not specified, it defaults to a value of 1.

Usage Note

The **skip** subcommand can be run only while the **dbx** debug program is running.

Example

To continue execution until the second breakpoint is encountered, enter:

```
skip 1
```

Related Information

The **cont** subcommand.

source subcommand for dbx: Read subcommands from a file

Format

source *file*

Description

The **source** subcommand reads **dbx** subcommands from the file specified by the *file* argument.

Usage Note

The **source** subcommand can be run only while the **dbx** debug program is running.

Example

To read the **dbx** subcommands in the **cmdfile** file, enter:

```
source cmdfile
```

status subcommand for dbx: Display the active trace and stop subcommands

Format

status [>*file*]

Description

The **status** subcommand displays the **trace** and **stop** subcommands currently active. The > option sends the output of the **status** subcommand to a file specified in the *file* argument.

Option

>*file* Redirects output to *file*.

Usage Note

The **status** subcommand can be run only while the **dbx** debug program is running.

Examples

1. To display the currently active **trace** and **stop** subcommands, enter:

   ```
   status
   ```

2. To stop at line 52 only when thread $t2 reaches that line, enter:

   ```
   stop at 52 if $t2==$current
   ```

Related Information

The **clear**, **delete**, **stop**, and **trace** subcommands.

---

step subcommand for dbx: Run one or more source lines

Format

step [*number*]

Description

The **step** subcommand runs source lines of the program. Specify the number of lines to be run with the *number* argument. If the *number* argument is omitted, it defaults to a value of `1`.

Usage Notes

1. The **step** subcommand can be run only while the **dbx** debug program is running.
2. If the *$hold_next* variable is defined, **dbx** holds all threads except the focus thread during a **step** subcommand. Then **dbx** unholds the threads after the **step** subcommand finishes.

Examples

1. To continue execution for one source line, enter:

   ```
   step
   ```

2. To continue execution for five source lines, enter:

   ```
   step 5
   ```

Related Information

The **cont**, **goto**, **next**, and **stepi** subcommands.

---

stepi subcommand for dbx: Run one or more machine instructions

Format

stepi [*number*]

Description

The **stepi** subcommand runs instructions of the program. Specify the number of instructions to be run in the *number* argument. If the *number* argument is omitted, it defaults to `1`.

Usage Notes

1. The **stepi** subcommand can be run only while the **dbx** debug program is running.
2. If the *$hold_next* variable is defined, **dbx** holds all threads except the focus thread during a **stepi** subcommand. Then **dbx** unholds the threads after the **stepi** subcommand finishes.

Examples

1. To continue execution for one machine instruction, enter:

   ```
   stepi
   ```

2. To continue execution for five machine instructions, enter:

   ```
   stepi 5
   ```

Related Information

The **gotoi**, **nexti**, and **step** subcommands.

---

stop subcommand for dbx: Stop execution of a program

Format

stop if *condition*
stop [*variable*] **at** *sourceline* [**if** *condition*]
stop [*variable*] **in** *procedure* [**if** *condition*]
stop *variable* [**if** *condition*]

Description

The **stop** subcommand stops execution of the program when certain conditions are fulfilled. The program is stopped when:

- The *condition* is true, if the **if***condition* option is used.
- The *sourceline* line number is reached, if the **at***sourceline* option is used.
- The *procedure* is called, if the **in***procedure* option is used.
- The *variable* is changed, if the *variable* argument is specified.

The **dbx** debug program associates event numbers with each **stop** subcommand. To view these numbers, use the **status** subcommand. To turn **stop** off, use the **delete** or **clear** subcommand.

Options

at*sourceline*
Specifies the line number.

if*condition*
Specifies the condition, such as true.

in*procedure*
Specifies the procedure to be called.

Usage Note

The **stop** subcommand can be run only while the **dbx** debug program is running.

Examples

1. To stop execution at the first executable statement in the **main** procedure, enter:

   ```
   stop in main
   ```

2. To stop execution when the value of the *x* variable is changed on line 12 of the execution, enter:

   ```
   stop x at 12
   ```

3. To stop execution at a specified line (line 23), when the value of the variable (*myvar*) is greater than 2, enter:

   ```
   stop at 23 if myvar > 2
   ```

4. To stop at line 52 only when thread $t2 reaches that line, enter:

   ```
   stop at 42 if $t2==$current
   ```

Related Information

The **stopi**, **delete**, **clear**, and **trace** subcommands.

---

stopi subcommand for dbx: Stop at a specified location

Format

stopi *address* [**if** *condition*]
stopi [*address*] **at** *address* [**if** *condition*]
stopi [*address*] **in** *procedure* [**if** *condition*]

Description

The **stopi** subcommand sets a stop at the specified location.

- With the **if***condition* option, the program stops when the condition is true.
- With the *address* argument, the program stops when the contents of *address* change.
- With the **at***address* option, a stop is set at the specified address.
- With the **in***procedure* option, the program stops when the procedure specified with the *procedure* argument is called.

Options

if*condition*
Specifies the condition, such as true.

in*procedure*
Specifies the procedure to be called.

at*address*
Specifies the machine instruction address.

Usage Note

The **stopi** subcommand can be run only while the **dbx** debug program is running.

Examples

1. To stop execution at address 0X100020F0, enter:

   ```
   stopi at 0x100020f0
   ```

2. To stop execution when the contents of address 0X100020F0 change, enter:

   ```
   stopi 0x100020f0
   ```

3. To stop at address 0x2d04567 only when thread $t2 reaches that address, enter:

   ```
   stopi at 0x2d04567 if $t2=$current
   ```

Related Information

The **stop** subcommand.

---

thread subcommand for dbx: Display a list of active threads

Format

thread [*number ...*]
thread hold [*number ...*]
thread unhold [*number ...*]
thread info [*number ...*]
thread current [*number ...*]
thread activ
thread async
thread dead
thread pcanc

Description

The **thread** subcommand displays a list of active threads for the application program. All active threads are listed unless you use the **number** parameter to specify the threads you want listed. You can also select threads by their states using the **activ**, **async**, **dead**, or **pcanc** options.

You can use the **info** option to display full information about a thread, and threads can be held or unheld with the **hold** or **unhold** options. The focus thread is defaulted to the running thread; **dbx** uses it as the context for normal **dbx** subcommands such as **register**. You can use the current option to switch the **dbx** focus thread.

Examples

1. To display all thread objects, enter:

   ```
   thread
   ```

2. To display thread objects number 1 and 2, enter:

   ```
   thread 1 2
   ```

3. To display all active threads, enter:

```
thread activ
```

4. To display all threads in dead state, enter:

   ```
   thread dead
   ```

5. To display all threads in async state (that is, threads with a cancelability type of PTHREAD_INTR_ASYNCHRONOUS) that are waiting to be scheduled), enter:

   ```
   thread async
   ```

   Because this thread was created with the PTATASYNCHRONOUS attribute and the limit was reached, this thread was queued for execution. For example, if the thread limit is set to ten and there are 12 threads, two of them will be shown as async for the dbx thread command.

6. To display all threads in pcanc state (that is, threads that have been requested to be canceled by **pthread_cancel()**, enter:

   ```
   thread pcanc
   ```

7. To hold all threads, enter:

   ```
   thread hold
   ```

8. To hold thread number 1 and 4, enter:

   ```
   thread hold 1 4
   ```

9. To unhold thread number 1 and 4, enter:

   ```
   thread unhold 1 4
   ```

10. To display the focus thread, enter:

    ```
    thread current
    ```

11. To set the focus thread to thread number 1, enter:

    ```
    thread current 1
    ```

12. To get information about thread number 3, enter:

    ```
    thread info 3
    ```

---

trace subcommand for dbx: Print tracing information

Format

trace [**if** *condition*]
trace *procedure* [**if** *condition*]
trace [*variable*] **at** *sourceline* [**if** *condition*]
trace [*variable*] **in** *procedure* [**if** *condition*]
trace *sourceline* [**if** *condition*]
trace *expression* **at** *sourceline* [**if** *condition*]

Description

The **trace** subcommand prints tracing information for the specified procedure, function, source line, expression, or variable when the program runs. A condition can be specified. The **dbx** debug program associates a number with each **trace** subcommand. To view these numbers, use the **status** subcommand. To turn tracing off, use the **delete** subcommand.

Options

at*sourceline*
Specifies the source line at which to find the expression being traced.

if*condition*
Specifies a condition for the beginning of the trace. The trace begins only **if***condition* is true.

in*procedure*
Specifies the procedure in which to find the procedure or variable being traced.

Usage Note

The **trace** subcommand can be run only while the **dbx** debug program is running.

Examples

1. To trace each call to the **printf()** procedure, enter:

   ```
   trace printf
   ```

2. To trace each execution of line 22 in the **hello.c** file, enter:

   ```
   trace "hello.c":22
   ```

3. To trace changes to the **x** variable within the **main** procedure, enter:

   ```
   trace x in main
   ```

4. To trace at line 52 only when mutex $m1 is not held, enter:

   ```
   trace at 52 if $m2.lock == 0
   ```

Related Information

The **tracei** subcommand.

---

tracei subcommand for dbx: Turn on tracing

Format

tracei [**if** *condition*]
tracei *address* [**at** *address*] [**if** *condition*]
tracei *address* [**in** *procedure*] [**if** *condition*]
tracei *expression* **at** *address* [**if** *condition*]

Description

The **tracei** subcommand turns on tracing when:

- The contents of the storage at the address change, if the *address* argument is specified.
- The instruction at the specified address is executed, if the **at***address* option is specified.
- The procedure specified by *procedure* is active, if the **in***procedure* option is included.
- The condition specified by the *condition* argument is true, if the **if***condition* option is included.

Options

at*address*
Specifies an address. Tracing is enabled when the contents of this address change.

if*condition*
Specifies a condition, the meeting of which causes tracing to be enabled.

in*procedure*
Specifies a procedure. Tracing is enabled when this procedure is active.

Usage Note

The **tracei** subcommand can be run only while the **dbx** debug program is running.

Examples

1. To trace each instruction run, enter:

   ```
   tracei
   ```

2. To trace each time the instruction at address 0X100020F0 is run, enter:

   ```
   tracei at 0x100020f0
   ```

3. To trace each time the contents of memory location 0X20004020 change while the **main** procedure is active, enter:

   ```
   tracei 0x20004020 in main
   ```

4. To trace at address 0x2d04567 only when thread $t2 reaches that address, enter:

   ```
   tracei at 0x2d04567 if $t2=$current
   ```

Related Information

The **trace** subcommand.

---

unalias subcommand for dbx: Remove an alias

Format

unalias *name*

Description

The **unalias** subcommand removes the alias specified by the *name* argument.

Usage Note

The **unalias** subcommand can be run only while the **dbx** debug program is running.

Example

To remove an alias named **printx**, enter:

```
unalias printx
```

Related Information

The **alias** subcommand.

---

unset subcommand for dbx: Delete a variable

Format

unset *name*

Description

The **unset** subcommand deletes the **dbx** debug program variable associated with the name specified by the *name* argument.

Usage Note

The **unset** subcommand can be run only while the **dbx** debug program is running.

Example

To delete the variable inhibiting the display of floating-point registers, enter:

```
unset $noflregs
```

Related Information

The **set** subcommand.

---

up subcommand for dbx: Move the current function up the stack

Format

up [*count*]

Description

The **up** subcommand moves the current function up the stack *count* number of levels. The current function is used for resolving names. The default for the *count* argument is 1.

Usage Note

The **up** subcommand can be run only while the **dbx** debug program is running.

Examples

1. To move the current function up the stack two levels, enter:

   ```
   up 2
   ```

2. To display the current function on the stack, enter:

   ```
   up 0
   ```

Related Information

The **down** subcommand.

---

use subcommand for dbx: Set the list of directories to be searched

Format

use [*directory...*]

Description

The **use** subcommand sets the list of directories to be searched

when the **dbx** debug program looks for source files. If the **use** subcommand is specified without arguments, the current list of directories to be searched is displayed.

If the C primary source is in an MVS data set, the **use** subcommand can be specified with a double-slash (//) argument to indicate that the source file be sought outside the hierarchical file system.

Usage Note

The **use** subcommand can be run only while the **dbx** debug program is running.

Examples

1. To change the list of directories to be searched to the working directory, the parent directory, and **/tmp**, enter:

   ```
   use . .. /tmp
   ```

2. To change the list of directories to be searched to look for the C source as an MVS data set, enter:

   ```
   use //
   ```

Related Information

The **edit** and **list** subcommands.

---

whatis subcommand for dbx: Display the type of program components

Format

whatis *name*

Description

The **whatis** subcommand displays the declaration of *name*, where the *name* argument designates a variable, procedure, or function name, optionally qualified with a block name.

Usage Notes

1. Variables declared with the **const** attribute (in your C program) are displayed without the **const** attribute.
2. The **whatis** subcommand can be run only while the **dbx** debug program is running.

Examples

1. To display the declaration of the *x* variable, enter:

   ```
   whatis x
   ```

2. To display the declaration of the **main** function, enter:

   ```
   whatis main
   ```

3. To display the declaration of the *x* variable within the **main** function, enter:

   ```
   whatis main.x
   ```

4. To display the declaration of a specific condition variable, $c1, enter:

   ```
   whatis $c1
   ```

5. To display the declaration of a specific mutex object, $m1, enter:

   ```
   whatis $m1
   ```

6. To display the declaration of a specific thread, $t1, enter:

   ```
   whatis $t1
   ```

---

where subcommand for dbx: List active procedures and functions

Format

where [>*file*]

Description

The **where** subcommand displays a list of active procedures and functions. By using the >*file* option, you can redirect the output of this subcommand to the specified file.

Options

>*flag*
Redirects output to the specified file.

Usage Note

The **where** subcommand can be run only while the **dbx** debug program is running.

Example

To display the list of active routines, enter:

```
where
```

Related Information

The **up** and **down** subcommands.

---

whereis subcommand for dbx: Display the full qualifications of symbols

Format

whereis *identifier*

Description

The **whereis** subcommand displays the full qualifications of all the symbols whose names match the specified identifier. The order in which the symbols print is not significant.

Usage Note

The **whereis** subcommand can be run only while the **dbx** debug program is running.

Example

To display the qualified names of all symbols named *x*, enter:

```
whereis x
```

Related Information

The **which** subcommand.

---

which subcommand for dbx: Display the full qualification of an identifier

Format

which *identifier*

Description

The **which** subcommand displays the full qualification of the given identifier. The full qualification consists of a list of the outer blocks with which the identifier is associated.

Usage Note

The **which** subcommand can be run only while the **dbx** debug program is running.

Example

To display the full qualification of the *x* symbol, enter:

```
which x
```

Related Information

The **whereis** subcommand.

---

dd — Convert and copy a file

Format

dd [**bs=***size*] [**cbs=***size*] [**conv=***conversion*] [**count=***n*] [**ibs=***size*] [**if=***file*] [**imsg=***string*] [**iseek=***n*] [**obs=***s*] [**of=***file*] [**omsg=***string*] [**seek=***n*] [**skip=***n*]>

Description

dd reads and writes data by blocks. It can convert data between formats. It is frequently used for such devices as tapes that have discrete block sizes, or for fast multisector reads from disks. **dd** performs conversions to accommodate nonprogrammable terminals, which require deblocking, conversion to and from EBCDIC, and fixed-length records.

dd processes the input data as follows:

1. **dd** reads an input block.
2. If this input block is smaller than the specified input block size, **dd** pads it to the specified size with null bytes. When you also specify a **block** or **unblock** conversion, **dd** uses spaces instead of null bytes.
3. If you specified **bs=***s* and requested no conversion other than **sync** or **noerror**, **dd** writes the padded (if necessary) input block to the output as a single block and omits the remaining steps.
4. If you specified the *swab* conversion, **dd** swaps each pair of input bytes. If there is an odd number of input bytes, **dd** does not attempt to swap the last byte.
5. **dd** performs all remaining conversions on the input data independently of the input block boundaries. A fixed-length input or output record may span these boundaries.

6. **dd** gathers the converted data into output blocks of the specified size. When **dd** reaches the end of the input, it writes the remaining output as a block (without padding if **conv=***sync* is not specified). As a result, the final output block may be shorter than the output block size.

Options

bs=*size*
Sets both input and output block sizes to *size* bytes. You can suffix this decimal number with **w**, **b**, **k**, or **x** *number*, to multiply it by 2, 512, 1024, or *number,* respectively. You can also specify *size* as two decimal numbers (with or without suffixes) separated by **x** to indicate the product of the two values. Processing is faster when **ibs** and **obs** are equal, since this avoids buffer copying. The default block size is 1B. **bs=***size* supersedes any settings of **ibs=***size* or **obs=***size*.

If you specify **bs=***size* and you request no other conversions than **noerror**, **notrunc**, or **sync**, **dd** writes the data from each input block as a separate output block; if the input data is less than a full block and you did not request **sync** conversion, the output block is the same size as the input block.

cbs=*size*
Sets the size of the conversion buffer used by various **conv** options.

conv=*conversion***[,** *conversion, ...***]**
conversion can be any one of the following:

ascii
Converts EBCDIC input to ASCII for output; it is provided for compatibility purposes only.

To copy a file and convert between a shell code page and ASCII, use **iconv**, not **dd**.

block
Converts variable-length records to fixed-length records. **dd** treats the input data as a sequence of variable-length records (each terminated by a newline or an EOF character) independent of the block boundaries. **dd** converts each input record by first removing any newline characters and then padding (with spaces) or truncating the record to the size of the conversion buffer. **dd** reports the number of truncated records on standard error (**stderr**). You must specify **cbs=***size* with this conversion.

Note: When working with doublebyte characters, **dd** truncates the record after the last complete doublebyte character that will fit in the conversion buffer. **dd** then pads the record with spaces if it is shorter than the conversion buffer size.

convfile
Uses **convfile** as a translation table if it is not one of the conversion formats listed here and it is the name of a file of exactly 256 bytes.

You can perform multiple conversions at the same time by separating arguments to **conv** with commas; however, some conversions are mutually exclusive (for example, **ucase** and **lcase**).

Notes:

1. When you specify one or more of the character set conversions (**ascii**, **ebcdic**, **ibm**, or **convfile**), **dd** assumes that all characters are singlebyte characters, regardless of the locale. Do not use these conversions with doublebyte character sets.
2. When working with DBCS text, **dd** treats the input and output files as character strings and handles DBCS characters correctly (no splitting and retaining of proper shift states). This happens only if any of the conversion options (**block**, **unblock**, **ucase**, or **lcase**) are specified. Otherwise, DBCS strings can be corrupted with the **seek**, **count**, or **iseek** processing.

ebcdic
Converts ASCII input to EBCDIC for output; it is provided for compatibility purposes only.

To copy a file and convert between a shell code page and ASCII, use **iconv**, not **dd**.

ibm
Like **ebcdic**, converts ASCII to EBCDIC; it is provided for compatibility purposes only.

To copy a file and convert between code page 01047 (used in the OS/390 shell) and ASCII, use **iconv**, not **dd**.

lcase
Converts uppercase input to lowercase.

noerror
Ignores errors on input.

notrunc
Does not truncate the output file. **dd** preserves blocks in the output file that it does not explicitly write to.

swab
Swaps the order of every pair of input bytes. If the current input record has an odd number of bytes, this conversion does not attempt to swap the last byte of the record.

sync
Specifies that **dd** is to pad any input block shorter than **ibs** to that size with NUL bytes before conversion and output. If you also specified *block* or *unblock*, **dd** uses spaces instead of null bytes for padding.

ucase
Converts lowercase input to uppercase.

unblock
Converts fixed-length records to variable-length records by reading a number of bytes equal to the size of the conversion buffer, deleting all trailing spaces, and appending a newline character. You must specify **cbs=***size* with this conversion.

count=*n*
Copies only *n* input blocks to the output.

ibs=*size*
Sets the input block *size* in bytes. You specify it in the same way as with the **bs** option.

if=*file*
Reads input data from *file*. If you don't specify this option, **dd** reads data from standard input (**stdin**).

imsg=*string*
Displays *string* when all data has been read from the current volume, replacing all occurrences of `%d` in *string* with the number of the next volume to be read. **dd** then reads and discards a line from the controlling terminal.

iseek=*n*
seeks to the *n*th block of the input file. The distinction between this and the **skip** option is that **iseek** does not read the discarded data. There are some devices, however, such as tape drives and communication lines, on which seeking is not possible, so only **skip** is appropriate.

obs=*size*
Sets the output block *size* in bytes. You specify it in the same way as the **bs** value. The size of the destination should be a multiple of the value chosen for *size*. For example, if you choose **obs=**`10K`, the destination's size should be a multiple of 10K.

of=*file*
Writes output data to *file*. If you don't specify this option, **dd** writes data to standard output (**stdout**). **dd** truncates the output file before writing to it, unless you specified the **seek=***n* operand. If you specify **seek=***n*, but do not specify **conv=***notrunc*, **dd** preserves only those blocks in the output file over which it seeks. If the size of the seek plus the size of the input file is less than the size of the output file, this can result in a shortened output file.

omsg=*string*
Displays *string* when **dd** runs out of room while writing to the current volume. Any occurrences of `%d` in *string* are replaced with the number of the next volume to be written. **dd** then reads and discards a line from the controlling terminal.

seek=*n*
Initially seeks to the *n*th block of the output file.

Note: Use caution when working with DBCS characters and the **seek** option. Seeking into the output file that contains DBCS characters can cause the DBCS string in the output file to be corrupted. Be sure that the seek count is not aligned with an existing DBCS string in the output file. Otherwise, part of the existing DBCS string either is written over with singlebyte data or has extra shift codes from the input file's DBCS data.

skip=*n*
Reads and discards the first *n* blocks of input.

Example

Entering:

```
dd if=in of=out conv=ascii cbs=80 ibs=6400 obs=512
```

converts 80-byte fixed-length EBCDIC card images in 6400-byte input blocks to variable-length ASCII lines, 512 bytes to the output block.

Localization

dd uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion

1 Failure due to any of the following:

- I/O errors on read/write
- Incorrect command-line option
- Incorrect arguments to a conversion

2 Failure resulting in a usage message such as:

- An option that should contain **=** does not
- Unknown or incorrect command-line option

Messages

Possible error messages include:

badly formed number *number*
A value specified as a number (for example, a block size) does not have the form of a number as recognized by **dd**. For example, you may have followed the number with a letter that **dd** does not recognize as a block-size unit (**w**, **b**, **k**).

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The **conv=ascii**, **conv=ebcdic**, **conv=ibm**, **conv=convfile**, **iseek**, **imsg**, and **omsg** options plus the **w** suffix described in the **bs=** option are all extensions of the POSIX standard.

Related Information

cp, **cpio**, **iconv**, **mv**, **tr**

df — Display the amount of free space in the file system

Format

df [–kPtv]][*filename*]] **...**

Description

df shows the amount of free space left on a file system, and the total amount available (free plus used). It measures space in units of 512-byte disk sectors. You can specify a particular file system by naming any filename on that file system. If you do not give an argument, **df** reports space for all mounted file systems known to the system, in the following format:

- File system root
- File system name
- Space available and total space

  The total space reported is the space in the already allocated extents (primary and any already allocated secondary extents) of the HFS data set that holds this file system. Therefore, the total space may increase as new extents are allocated.
- Number of free files (inodes).

  This number is only meaningful for file systems created using DFSMS 1.3.0 and later. For file systems created with earlier versions of DFSMS, this number will always be 4 294 967 295.
- File system status

Options

–k Uses 1024-byte (1KB) units instead of the default 512-byte units when reporting space information.

–P Lists complete information on space used, in the following order:

- File system name
- Total space
- Space used
- Space free
- Percentage of space used
- File system root

–t Display total allocated file slots, in addition to the total number of free files that are already displayed.

–v Lists more detailed information on the file system status.

- File system root
- File system name
- Space available and total space
- Number of free files (inodes)
- File system status
- File system type, and mode bits
- File system mount parm data

For systems in a shared HFS environment, the following additional fields are displayed:

- File system ID (owner/mounted file system server)
- File system ID issuing a quiesce request

If **df** is issued in a non-shared HFS environment, these two fields are not relevant and won't be displayed.

Example

If you issue a **df –v** on a file system whose owner is participating in shared HFS, status information such as the following will be displayed:

```
Mounted on       Filesystem       Avail/Total    Files      Status
/u               (W40LIE3.A.HFS)   992/1440      4294967286   Available
                 HFS, Read/Write
                 File System Owner:  SY1

/SY1/ict/rofs    (ROFS.HFS)       1208/1440     4294967290    Available
                 HFS, Read Only
                 File System Owner:  SY1

/SY1/ict/rwfs    (W40LIE2.A.HFS)   1208/1440    429467290     Available
                 HFS, Read/Write
                 File System Owner:  SY1

/SY1             (POSIX.SY1.HFS)   9096/11808   4294967043    Available
                 HFS, Read/Write
                 File System Owner:  SY1

                 (POSIX.FS02)       Not Available             Quiesced
                 HFS, Read/Write
                 File System Owner:  SY1
                 Quiesce Owner:  SY2
```

Localization

df uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion
1 Failure due to any of the following:
- Inability to access *filename*
- Inability to access *device*
- *device* is not a device

2 Incorrect command-line option

Portability

POSIX.2 User Portability Extension, X/Open Portability Guide, UNIX systems.

Related Information

du, **ls**

---

diff — Compare two text files and show the differences

Format

diff [**–befHhimnrsw**] [**–C** *n*] [**–c**[*n*]] [**–D***ifname*] *path1 path2*

Description

The **diff** command attempts to determine the minimal set of changes needed to convert a file whose name is specified by the *path1* argument into the file specified by the *path2* argument.

Input files must be text files if either (but only one) filename is **–**, **diff** uses a copy of the standard input (**stdin**) for that file. If exactly one of *path1* or *path2* is a directory, **diff** uses a file in that directory with the same name as the other filename. If both are directories, **diff** compares files with the same filenames under the two directories; however, it does not compare files in subdirectories unless you specify the **–r** option. When comparing two directories, **diff** does not compare character special files, or FIFO special files with any other files.

By default, output consists of descriptions of the changes in a style like that of the **ed** text editor. A line indicating the type of change is given. The three types are `a` (append), `d` (delete), and `c` (change). The output is symmetric: A delete in *path1* is the counterpart of an append in *path2*. **diff** prefixes each operation with a line number (or range) in *path1* and suffixes each with a line number (or range) in *path2*. After the line giving the type of change, **diff** displays the deleted or added lines, prefixing lines from *path1* with `<` and lines from *path2* with `>`.

Options

Options that control the output or style of file comparison are:

–b Ignores trailing blanks and tabs and considers adjacent groups of blanks and tabs elsewhere in input lines to be equivalent.

For example, if one file contained a string of three spaces and a tab at a given location while the other file contained a string of two spaces at the same location, **diff** would not report this as a difference.

–C *n*

Shows *n* lines of context before and after each change. **diff** marks lines removed from *path1* with **–**, lines added to *path2* with +, and lines changed in both files with !.

–c[*n***]**

Is equivalent to **–C***n*, but *n* is optional. The default value for *n* is 3. **diff** marks lines removed from *path1* with **–**, lines added to *path2* with **+**, and lines changed in both files with `!`.

–D*ifname*
Displays output that is the appropriate input to the C preprocessor to generate the contents of *path2* when *ifname* is defined, and the contents of *path1* when *ifname* is not defined.

–e Writes out a script of commands for the **ed** text editor, which converts *path1* to *path2*. **diff** sends the output to the standard output (**stdout**).

–f Writes a script to **stdout** (but it will be in a form not suitable for use with the **ed** editor) showing the modifications necessary to convert *path1* to *path2* in the reverse order of that produced by the **–e** option. The commands produced will be reversed from that produced by **–e**, and the line number ranges will be separated by spaces, rather than commas. This option conflicts with the **–m** option.

–H Uses the half-hearted (**–h**) algorithm only if the normal algorithm runs out of system resources.

–h Uses a fast, half-hearted algorithm instead of the normal **diff** algorithm. This algorithm can handle arbitrarily large files; however, it is not particularly good at finding a minimal set of differences in files with many differences.

–i Ignores the case of letters when doing the comparison.

–m Produces the contents of *path2* with extra formatter request lines interspersed to show which lines were added (those with vertical bars in the right margin) and deleted (indicated by a * in the right margin).

–M Is an IBM internal option and is not supported.

–n Is an IBM internal option and is not supported.

–N Is an IBM internal option and is not supported.

–r Compares corresponding files under the directories, and recursively compares corresponding files under corresponding subdirectories under the directories. You can use this option when you specify two directory names on the command line.

–s Compares two directories, file by file, and prints messages for identical files between the two directories.

–w Ignores white space when making the comparison.

Examples

The following example illustrates the effect of the **–c** option on the output of the **diff** command. The following two files, **price1** and **price2**, are compared with and without the use of the **–c** option.

The contents of **price1** are as follows:

```
Company X Price List:
$  0.39  -- Package of Groat Clusters
$  5.00  -- Candy Apple Sampler Pack
$ 12.00  -- Box of Crunchy Frog Chocolates
$ 15.99  -- Instant Rain (Just Add Water)
$ 20.00  -- Asparagus Firmness Meter
$ 25.00  -- Package of Seeds for 35 Herbs
$ 30.00  -- Child's Riding Hood (Red)
$ 35.00  -- Genuine Placebos
$ 45.00  -- Case of Simulated Soy Bean Oil
$ 75.88  -- No-Name Contact Lenses
$ 99.99  -- Kiddie Destructo-Bot
$125.00  -- Emperor's New Clothes
```

The contents of **price2** are as follows:

```
Company X Price List:
$  0.39  -- Package of Groat Clusters
$  5.49  -- Candy Apple Sampler Pack
$ 12.00  -- Box of Crunchy Frog Chocolates
$ 15.99  -- Instant Rain (Just Add Water)
$ 17.00  -- Simulated Naugahyde cleaner
$ 20.00  -- Asparagus Firmness Meter
$ 25.00  -- Package of Seeds for 35 Herbs
$ 30.00  -- Child's Riding Hood (Red)
$ 35.00  -- Genuine Placebos
$ 45.00  -- Case of Simulated Soy Bean Oil
$ 75.88  -- No-Name Contact Lenses
$ 99.99  -- Kiddie Destructo-Bot
```

The command:

```
diff price1 price2
```

results in the following output:

```
3c3
< $  5.00  -- Candy Apple Sampler Pack
---
> $  5.49  -- Candy Apple Sampler Pack
5a6
> $ 17.00  -- Simulated Naugahyde cleaner
13d13
< $125.00  -- Emperor's New Clothes
```

The addition of the **–c** option, as in:

```
diff -c price1 price2
```

results in the following output:

```
*** price1 Wed Oct 1 13:59:18 1997
--- price2 Wed Oct 1 14:03:36 1997
***************
*** 1,8 ****
Company X Price List:

  $  0.39  -- Package of Groat Clusters
! $  5.00  -- Candy Apple Sampler Pack
  $ 12.00  -- Box of Crunchy Frog Chocolates
  $ 15.99  -- Instant Rain (Just Add Water)
  $ 20.00  -- Asparagus Firmness Meter
  $ 25.00  -- Package of Seeds for 35 Herbs
  $ 30.00  -- Child's Riding Hood (Red)
--- 1,9 ----
  Company X Price List:

  $  0.39  -- Package of Groat Clusters
! $  5.49  -- Candy Apple Sampler Pack
  $ 12.00  -- Box of Crunchy Frog Chocolates
  $ 15.99  -- Instant Rain (Just Add Water)
+ $ 17.00  -- Simulated Naugahyde cleaner
  $ 20.00  -- Asparagus Firmness Meter
  $ 25.00  -- Package of Seeds for 35 Herbs
  $ 30.00  -- Child's Riding Hood (Red)
***************
*** 10,13 ****
  $ 45.00  -- Case of Simulated Soy Bean Oil
  $ 75.88  -- No-Name Contact Lenses
  $ 99.99  -- Kiddie Destructo-Bot
- $125.00  -- Emperor's New Clothes
--- 11,13 ----
```

diff –c marks lines removed from **price1** with **–**, lines added to **price1** with **+** and lines changed in both files with `!`. In the example, **diff** shows the default three lines of context around each changed line. One line was changed in both files (marked with `!`), one line was added to **price1** (marked with **+**), and one line was removed from **price1** (marked with –).

Note: If there are no marks to be shown in the corresponding lines of the file being compared, the lines are not displayed. Lines 11 to 13 of **price2** are suppressed for this reason.

Localization

diff uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_TIME**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 No differences between the files compared.

1 **diff** compared the files and found them to be different.

2 Failure due to any of the following:

- Incorrect command-line argument
- Inability to find one of the input files
- Out of memory
- Read error on one of the input files

4 At least one of the files is a binary file containing embedded NUL (\0) bytes or newlines that are more than LINE_MAX bytes apart.

Messages

Possible error messages include:

file *filename***: no such file or directory**
The specified *filename* does not exist. *filename* was either typed explicitly, or generated by **diff** from the directory of one file argument and the basename of the other.

Files *file1* **and** *file2* **are identical**
The **–s** option was specified and the two named files are identical.

Common subdirectories: *name* **and** *name*
This message appears when **diff** is comparing the contents of directories, but you have not specified **–r**. When **diff** discovers two subdirectories with the same name, it reports that the directories exist, but it does not try to compare the contents of the two directories.

Insufficient memory (try diff –h)
diff ran out of memory for generating the data structures used in the file differencing algorithm. (See “Limits” on page 219.) The **–h** option of **diff** can handle any size file without running out of memory.

Internal error—cannot create temporary file
diff was unable to create a working file that it needed. Ensure that you either have a directory /**tmp** or that the environment contains a variable **TMPDIR,** which names a directory where **diff** can store temporary files. Also, ensure that there is sufficient file space in this directory.

Missing *ifdef* **symbol after -D**
You did not specify a conditional label on the command line after the **–D** option.

Only one file may be –
Of the two input files normally found on the command line of **diff**, only one can be the standard input (**stdin**).

Too many lines in *filename*
A file of more than the maximum number of lines (see “Limits” on page 219) was given to **diff**.

Limits

The longest input line is 1024 bytes. Except under **–h**, files are limited to INT_MAX lines. INT_MAX is defined in **limits.h**.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The **–D**, **–f**, **–H**, **–h**, **–i**, **–m**, **–s**, and **–w** options, and the *n* argument to the **–c** option, are extensions of the POSIX standard.

Related Information

cmp, **comm**, **patch**

J. W. Hunt and M. D. McIlroy, *An Algorithm for Differential File Comparison*, Report 41, from Computing Science, Bell Laboratories, Murray Hill, NJ 07974, (June 1976), 9 pages.

---

dircmp — Compare directories

Format

dircmp [**–d**] [**–s**] [**–w***n*] *dir1 dir2*

Description

dircmp examines *dir1* and *dir2* and generates listings about the contents of the directories. Listings of files that are unique to each directory are generated for all the options. If no option is entered, a list is output indicating whether the filenames common to both directories have the same contents.

Options

–d Compare the contents of files with the same name in both directories and output a list telling what must be changed in the two files to bring them into agreement. The list format is described in **diff**.

–s Suppress messages about identical files.

–w*n* Change the width of the output line to *n* characters. The default width is 72.

Localization

dircmp uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Related Information

cmp, **diff**

---

dirname — Return the directory components of a pathname

Format

dirname *pathname*

Description

dirname deletes the trailing part of a filename. The result is the pathname of the directory that contains the file. This is useful in shell scripts. **dirname** does not try to validate the pathname. For validation, use **pathchk**.

dirname follows these rules:

1. If *pathname* is `//`, return it.
2. Otherwise, if it is all slashes, return one slash.
3. Otherwise, remove all trailing slashes.
4. If there are no slashes remaining in *pathname*, return period (`.`).
5. Otherwise, remove trailing nonslash characters.
6. If the remaining string is `//`, return it.
7. Otherwise, remove any trailing slashes.
8. If the resulting string is empty, return period (`.`).
9. Otherwise, return the resulting string.

Examples

The command:

```
dirname src/lib/printf.c
```

produces:

```
src/lib
```

Localization

dirname uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

`0` Successful completion
`1` Failed
`2` Unknown command-line option

Portablity

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Information

basename, **pathchk**

. (dot) — Run a shell file in the current environment

Format

. *file* [*argument* **...**]

Description

. (**dot**) runs a shell script in the current environment and then returns. Normally, the shell runs a command file in a child shell so that changes to the environment by such commands as **cd**, **set**, and **trap** are local to the command file. The **.** (**dot**) command circumvents this feature.

If there are slashes in the filename, **.** (**dot**) looks for the named file. If there are no slashes **.** (**dot**) uses the search **PATH** variable to find *file*. This may surprise some people when they use dot to run a file in the working directory, but their search rules are not set up to look at the working directory. As a result, the shell doesn't find the shell file. If you have this problem, you can use:

```
. ./file
```

This indicates that the shell file you want to run is in the working directory. Also, the file need not be executable, even if it is looked for on the **PATH**. If you specify an argument list *argument ...*, **.** (**dot**) sets the positional parameters to this list before execution.

Usage Notes

1. **.** (**dot**) is a special built-in shell command.
2. The file specified is treated as a shell script containing shell commands. Files which are not shell scripts (e.g. REXX execs, executable programs) should not be specified as *file*.

Localization

. (**dot**) uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

1 The path search failed
2 Failure because of an incorrect command-line option

Otherwise, the exit status is the exit status of the last command run from the script.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Information

cd, **set**, **sh**, **trap**

---

dspcat — Display all or part of a message catalog

Format

dspcat [**–gt**] **CatalogName** [**SetNumber** [**MessageNumber**]]

Description

dspcat displays a particular message, all of the messages in a set, or all of the messages in a catalog. Messages are displayed as they are specified in the message catalog; no substitution of variables takes place.

It directs the messages to standard output (**stdout**).

It has the following parameters:

- The **CatalogName** parameter specifies a message catalog.
- The **SetNumber** parameter specifies a set in the catalog specified by the **CatalogName** parameter. If you specify a nonexistent **SetNumber** value, all messages in the catalog are displayed.
- The **MessageNumber** parameter specifies a particular message in the set specified by the **SetNumber** parameter.

If you include all three parameters, **dspcat** displays a particular message. If you do not include the **MessageNumber** parameter, or if the **MessageNumber** value is in error, all the messages in the set are displayed. If you specify only the **CatalogName** parameter, all the messages in the catalog are displayed. You must include the **SetNumber** parameter if you include the **MessageNumber** parameter.

Use the **NLSPATH** environment variable to find the specified message catalog if slash (/) characters are not used in the value of the **CatalogName** parameter.

Options

–g Formats the output so it can be used as input to the **gencat** command. The **MessageNumber** parameter is not valid when **–g** is specified.

–t Displays the timestamp of the message catalog.

Examples

To display message number 2 in set number 1 of **test.cat,** enter:

```
dspcat test.cat 1 2
```

---

dspmsg — Display selected messages from message catalogs

Format

dspmsg [**–d]** [**–s SetNumber**] **CatalogName MessageNumber** [**'DefaultMessage'**[**Arguments**]]

Description

dspmsg displays either the text of a particular message from a message catalog generated with the **gencat** command or, if the message cannot be retrieved, a default message supplied as a parameter to the command. **dspmsg** directs the message to standard output. This command is intended for use in shell scripts as a replacement for the **echo** command.

The **NLSPATH** environment variable and the **LANG** category are used to find the specified message catalog if / (slash) characters are not used in the value of the **CatalogName** parameter. If the catalog named by the **CatalogName** parameter is not found or if the message named by the **MessageNumber** parameter (and optional **SetNumber** value) is not found, then the supplied **DefaultMessage** value is displayed. If a **DefaultMessage** value is not specified, a system-generated error message is displayed.

dspmsg allows up to ten string arguments to be substituted into the message if it contains the **%s** or **%n$s**, fprintf() conversion specification. Only string variables are allowed. If arguments are specified, then a **DefaultMessage** must also be specified.

Missing arguments for conversion specifications result in a **dspmsg** error message. Normal **fprintf()** subroutine control character escape codes (for example, **–n**) are recognized.

Options

–d If you are receiving the default message, use this option to request debugging information on why **dspmsg** cannot get the message from the message catalog.

–s SetNumber
Specifies an optional set number. The default value for the **SetNumber** variable is 1.

Examples

To display set number 1, message number 2 of the **test.cat** catalog, enter:

```
dspmsg –s 1 test.cat 2 'message %s not found' 2
```

If the message is not found, `message 2 not found` is displayed.

du — Summarize usage of file space

Format

du [**–al–s[–krtx] [** *pathname ...*]

Description

du reports the amount of file space used by the files indicated by the given pathname. If the pathname is a directory, **du** reports the total amount of file space used by all files in that directory and in each subdirectory in its hierarchy. If you do not specify a pathname, **du** assumes the current directory. Files with multiple links are only counted once. On systems supporting symbolic links, only the disk space used by the symbolic link is counted.

du measures file space in 512-byte units.

Options

–a Generates a report for all files in *pathname*.

–k Displays file sizes in 1024-byte (1KB) units.

–r Reports files that cannot be opened and directories that cannot be read; this is the default.

–s Does not display file size totals for subdirectories.

–t Displays the total amount of space used by all pathnames examined.

–x Displays file sizes for only those files contained on the same device as *pathname*.

Usage Notes

du computes file space in units of 512 bytes. The actual disk space used by files and directories may be more, since some systems allocate space in units of some multiple of a sector. On UNIX System V, it is usually two sectors; on UNIX Version 7, it is one sector.

OS/390 UNIX System Services allocates files in blocks of 4096 bytes.

Localization

du uses the following localization variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion

1 Failure due to any of the following:

- Incorrect command-line option
- Cannot access a directory
- Cannot read a directory
- Cannot access file information

Portability

POSIX.2 User Portability Extension, X/Open Portability Guide, UNIX systems.

The **–t** option is an extension to the POSIX standard.

Related Information

df, **find**, **ls**

---

echo — Write arguments to standard output

Format

echo *argument ...*

tcsh shell: **echo** [**-n**] *word ...*

Description

echo writes its arguments, specified with the *argument* argument, to standard output. **echo** accepts these C-style escape sequences:

\a	Bell
\b	Backspace
\c	Removes any following characters, including **\n** and **\r**.
\f	Form feed
\n	Newline
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\0*num*	The byte with the numeric value specified by the zero to three-digit octal *num*.
\–	Backslash

echo follows the final argument with a newline unless it finds `\c` in the arguments. Arguments are subject to standard argument manipulation.

echo in the tcsh shell

In the tcsh shell, **echo** writes each word to the shell's standard output, separated by spaces and terminated with a newline.

tcsh **echo** accepts these C-style escape sequences:

\a	Bell
\b	Backspace

\e	Escape
\f	Form feed
\n	Newline
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
nnn	The EBCDIC character corresponding to the octal number *nnn*

See “tcsh — Invoke a C shell” on page 556.

Examples

1. One important use of **echo** is to expand filenames on the command line, as in:

   ```
   echo *.[ch]
   ```

   This displays the names of all files with names ending in **.c** or **.h**—typically C source and include (header) files. **echo** displays the names on a single line. If there are no filenames in the working directory that end in **.c** or **.h**, **echo** simply displays the string ***.[ch]**.

2. **echo** is also convenient for passing small amounts of input to a filter or a file:

   ```
   echo 'this is\nreal handy' > testfile
   ```

Usage Note

echo is a built-in shell command.

Localization

echo uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Value

echo always returns the following exit status value:

`0` Successful completion

Portability

POSIX.2, X/Open Portability Guide, UNIX System V.

The POSIX.2 standard does not include escape sequences, so a strictly conforming application cannot use them. **printf** is suggested as a replacement.

Related Information

sh, **tcsh**

---

ed — Use the ed line-oriented text editor

Format

ed [–bs] [**–p** *prompt*] [*file*]

Description

ed is a line-oriented text editor that lets you manipulate text files interactively. **ed** reads the text of a file into memory and stores it in an area called a *buffer*. Various subcommands let you edit the text in the buffer. You can also write the contents of the buffer back out to the file, thereby overwriting the old contents of the file.

Options

–b Lets you edit larger files by restricting the amount of memory dedicated to paging. This frequently makes **ed** run slower.

–p *prompt*
Displays the given *prompt* string prompting you to input a subcommand. By default, **ed** does not usually prompt for subcommand input. See the description of the **P** subcommand for more on subcommand prompting (see "Subcommands" on page 228).

–s Puts **ed** into a quiet mode, in which **e**, **E**, **r**, and **w**, subcommands do not display file size counts; the **q** and **e** subcommands do not check buffer modification; and ! is not displayed after calling the shell to run a subcommand. This mode is particularly useful when you invoke **ed** from within a shell script.

If the optional *file* argument is present on the command line, **ed** reads the specified *file* into the editor by simulating an **e***file* subcommand.

Addresses

You can prefix subcommands in **ed** with zero, one, or two addresses. These addresses let you refer to single lines or ranges of lines in the buffer. You do not need to specify addresses for certain subcommands that use default addresses. Consult the description for a particular subcommand. You can construct each address out of the following components:

. The single *dot* character represents the *current line* in the buffer. Many subcommands set the *current line*; for example the **e** command sets it to the last line of the new file being edited.

$ The dollar sign refers to the last line in the buffer.

n The number *n* refers to the *n*th line in the buffer.

/regexp/
This searches for a line containing a string that matches the regular expression, **regexp**. (For information on regular expressions, see Appendix F.) The search begins at the line immediately following the current line. It proceeds *forward* through the buffer; if **ed** reaches the end of the buffer without finding a match, it wraps around to the first line of the buffer and con-

tinues the search. If **ed** does not find a match, the search ends when it reaches the original current line. If it does find a match, the address */regexp/* refers to the first matching line. If you omit *regexp*, the last used regular expression becomes the object of the search. You can omit the trailing /. Within *regexp*, \/ represents a literal slash and not the *regexp* delimiter.

?regexp?
This is similar to the previous address form, except that the search goes *backward* through the buffer. If the search reaches the first line in the buffer without finding a match, **ed** wraps around and continues searching backward from the last line in the buffer. If you omit *regexp*, the last used regular expression becomes the object of the search. You can omit the trailing *?*. Within *regexp*, *?* represents a literal question mark and not the *regexp* delimiter.

'l The address is the line marked with the mark name *l*. The name *l* must be a lowercase letter set by the **k** subcommand.

You can combine these basic addresses with numbers using the + and **–** operators, with the usual interpretation. Missing left operands default to . (dot); missing right operands default to 1. Missing right operands also have a cumulative effect; so an address of – – refers to the current line number less two.

You can specify address ranges in the following ways:

a1,a2 Specifies a range of addresses from address *a1* to address *a2*, inclusive. If you omit *a1* and *a2* (that is, the comma alone is specified), this is equivalent to the range 1**,**$.

*a1***;***a2* Is similar to the previous form except that **ed** resets the current line after calculating the first address, *a1*, so that the second address, *a2*, is relative to *a1*. If you omit *a1* and *a2* (that is, the semicolon alone is specified), this is equivalent to **.;**$. If you specify only *a1* and the command requires both *a1* and *a2*, the command operates as though you specified a range of:

*a1***;.** *command*

> Is equivalent to **.,.**+22 (that is, page forward), except that it never attempts to address any line beyond $.

< Is equivalent to **.**–22**,.** (that is, page backward), except that it never addresses any line before line 1.

Subcommands

An **ed** command has the form [address] *command*

All commands end with a newline; you must press <Enter>. Most commands allow only one command on a line, although you can modify commands by appending the **ln**, **n**, and **p** commands.

Subcommands generally take a maximum of zero, one, or two addresses, depending upon the particular subcommand. In the following descriptions, we show commands with their default addresses (that is the addresses used when you don't specify any addresses) in a form that shows the maximum number of permitted addresses for the command. In any of the subcommands that take a *file* argument, *file* can be a pathname or:

!*command-line*

If you use the **!** form, **ed** runs the given command line, reading its standard output (**stdout**) or writing its standard input (**stdin**), depending on whether the **ed** command does reading or writing.

ed accepts the following subcommands:

.a Appends text *after* the specified line. Valid addresses range from `0` (text is placed after the last line of the buffer, before the first line) to `$` (text is placed after the last line of the buffer). **ed** reads lines of text from the workstation until a line consisting solely of an unescaped **.** (dot) is entered. **ed** sets the current-line indicator to the last line appended.

.,.c Changes the addressed range of lines by deleting the lines and then reading new text in the manner of the **a** or **i** subcommands.

.,.d Deletes the addressed range of lines. The line after the last line deleted becomes the new current line. If you delete the last line of the buffer, **ed** sets the current line to the new last line. If no lines remain in the buffer, it sets the current line to `0`.

E[*file***]**
Is similar to the **e** command, but **ed** gives no warning if you have changed the buffer.

e [*file***]**
Replaces the contents of the current buffer with the contents of *file*. If you did not specify *file*, **ed** uses the remembered filename, if any. In all cases, the **e** subcommand sets the remembered filename to the file that it has just read into the buffer. **ed** displays a count of the bytes in the file unless it is in *quiet* mode. If you have changed the current buffer since the last time its contents were written, **ed** warns you if you try to run an **e** subcommand, and does not run the subcommand. If you enter the **e** subcommand a second time, **ed** goes ahead and runs the command.

f [*file***]**
Changes the remembered filename to *file*. **ed** displays the new remembered filename. If you do not specify *file*, **ed** displays the current remembered filename.

1,$G/*regexp/*
Is similar to the **g** command except that when **ed** finds a line that matches *regexp*, it prints the line and waits for you to type in the subcommand to be run. You cannot use the **a**, **c**, **i**, **g**, **G**, **v**, and **V** subcommands. If you enter `&`, the **G** subcommand reruns the last subcommand you typed in. If you just press <Enter>, **G** does not run any subcommand for that line.

1,$g/*regexp/command*
Performs *command* on all lines that contain strings matching the regular expression *regexp*. This subcommand works in two passes. In the first pass, **ed** searches the given range of lines and marks all those that contain strings matching the regular expression *regexp*. The second pass actually performs *command* on those lines. You cannot use **!**, **g**, **G**, **V**, or **v** as *command. command* consists of one or more **ed** subcommands, the first of which must appear on the same line as the **g** subcommand. All lines of a multiline command list, except the last, must end with a backslash (\). If *command* is empty, **ed** assumes it to be the **p** subcommand. If no lines match *regexp*, **ed** does not change the current line number; otherwise, the current line number

is the one set by the last subcommand in *command*. Instead of the slash (/) to delimit *regexp*, you can use any character other than space or newline.

H Tells **ed** to display more descriptive messages when errors occur. If **ed** is already printing descriptive messages, **H** returns to terse error messages. Normally, **ed** indicates error messages by displaying a ?. When you turn on descriptive error messages with this subcommand, **ed** also displays the descriptive message for the most recent ? message.

h Provides a brief explanation of the last error that occurred. This does not change the current line number.

.i Works similarly to the **a** subcommand, except that **ed** places the text *before* the addressed line. Valid addresses range from line 1 to $ (the last line). **ed** sets the current line number to the last inserted line.

.,.+1j
Joins a range of lines into one line. To be precise, the **j** command removes all newline characters from the addressed range of lines, except for the last one. **ed** sets the current line number to the resulting combined line.

.kx Marks the addressed line with the mark name *x*, which is any single lowercase letter of the alphabet. This lets you refer to a marked line with the construct **'x**. This is called an absolute address, because it always refers to the same line, regardless of changes to the buffer.

.,.l Displays the addressed range of lines, representing nonprintable (control) characters in a visible manner. **ed** sets the current line to the last line so displayed. You can append this subcommand to most other commands, to check on the effect of those subcommands.

.,.m*a*
Moves the addressed lines to the point immediately following the line given by the address *a*. The address *a* must not be in the range of addressed lines. If address *a* is 0, **ed** moves the lines to the beginning of the buffer. The last line moved becomes the new current line.

.,.n Displays the addressed lines in a way similar to the **p** command, but **ed** puts the line number and a tab character at the beginning of each line. The last line displayed becomes the new current line. You can append **n** to any subcommand (except for **E**, **e**, **f**, **Q**, **r**, **w**, or **!**) so that you can check on the effect that the subcommands had.

P Turns on subcommand prompting if it is not already on. If you specified the **–p** *prompt* option on the **ed** command line, **ed** displays the *prompt* string whenever it is ready for you to type in another subcommand. If you did not include the **–p** option, **ed** uses the * character as a prompt. If subcommand prompting is currently turned on, issuing the **P** subcommand turns it off.

.,.p Displays (prints) the addressed lines. The last line displayed becomes the new current line. You can append **p** to most subcommands, so that you can check on the effect that the subcommands had.

You can append **p** to any subcommand (except for **E**, **e**, **f**, **Q**, **r**, **w**, or **!**) so that you can check on the effect that the subcommands had.

Q Quits unconditionally, without checking for buffer changes.

q Causes the editor to exit. If you have made changes to the buffer since the last save and you try to quit, **ed** issues a warning. Entering the **q** subcommand again lets you quit, regardless of unsaved changes.

$r [*file***]**
Reads the contents of the *file* into the buffer after the addressed line. If the address is `0`, **ed** places the text before the first line in the buffer. If you do not specify *file*, **ed** uses the remembered filename; if no remembered filename exists, *file* becomes the new remembered name. If *file* contains bytes that are not valid in the current character set, they are replaced by the rubout character.

The **r** subcommand displays the number of bytes read from *file* unless you specified the **–s** option. The last line read from the file becomes the new current line. If *file* is replaced by **!**, the rest of the line is considered a shell command line, the output of which is to be read.

.,.s/*regexp/new/[flags]*
Searches the specified range of lines for strings matching the regular expression *regexp*. Normally the **s** subcommand replaces the first such matching string in each line with the string *new*. The **s** subcommand sets the current line to the last line on which a substitution occurred. If **ed** makes no such replacements, **ed** considers it an error.

flags can be one of the following:

n Replaces the *n*th matching string in the line instead of the first one.
g Replaces *every* matching string in each line, not just the first one.
l Displays the new current line in the format of the **l** subcommand.
n Displays the new current line in the format of the **n** subcommand.
p Displays the new current line in the format of the **p** subcommand.

You can use any single printable character other than space or newline instead of / to separate parts of the subcommand provided that you use the same character to delimit all parts of the subcommand. You can omit the trailing delimiter.

You can include a newline in the *new* string by putting a \ immediately in front of the newline. This is a good way to split a line into two lines. If *new* consists only of the `%` character, **s** uses the *new* string from the previous **s** command. If `&` appears anywhere in *new*, **ed** replaces it with the text matching the *regexp*. If you want *new* to contain a literal ampersand, or percent sign, put a backslash (\) in front of the `&` or `%` character.

.,.t*a* Copies the addressed lines to the point *after* the line given by the address *a*. The address *a* must not fall in the range of addressed lines. If address *a* is `0`, **ed** copies the lines to the beginning of the buffer. This sets the current line to the last line copied.

u Rolls back the effect of the last subcommand that changed the buffer. For the purposes of **u**, subcommands that change the buffer are: **a**, **c**, **d**, **g**, **G**, **i**, **j**, **m**, **r**, **s**, **t**, **v**, **V**, and (of course) **u**. This means that typing **u** repeatedly switches the most recent change back and forth. This subcommand sets the current line number to the value it had immediately before the subcommand being undone started.

1,$V/*regexp/*
Is similar to the **G** subcommand, except that this subcommand gives you the chance to edit only those lines that do *not* match the given regular expression.

1,$v/*regexp/commands*
Is similar to the **g** (global) command, except that **ed** applies the given *commands* only to lines that do *not* match the given regular expression.

1,$W [file]
Is similar to the **w** subcommand, except that this command appends data to the given *file* if the file already exists.

1,$w [*file***]**
Writes the addressed lines of the buffer to the named *file*. This does not change the current line. If you do not provide *file*, **ed** uses the remembered filename; if there is no remembered filename, *file* becomes the remembered name. If the output file does not exist, **ed** creates it. **ed** displays the number of characters written unless you had specified the **–s** option.

X Prompts you to enter an encryption key. All subsequent **e**, **r**, and **w** subcommands use this key to decrypt or encrypt text read from or written to files. To turn encryption off, issue an **X** subcommand and press <Return> in response to the prompt for an encryption key.

!*command*
Runs *command* as if you typed it to your chosen command interpreter. If *command* contains the `%` character, **ed** replaces it with the current remembered filename. If you want a subcommand to contain a literal `%`, put a backslash (\) in front of the character. As a special case, typing **!!** reruns the previous *command*.

$= Displays the line number of the addressed line. This does not change the current line.

.= Displays the current line number.

.+1.,.+1
If you supply zero, one, or two addresses without an explicit subcommand, **ed** displays the addressed lines in the mode of the last print subcommand: **p**, **l**, or **n**. This sets the current line number to the last line displayed.

Environment Variables

ed uses the following environment variables:

COLUMNS
Contains the terminal width in columns. **ed** folds lines at that point. If it is not set, **ed** uses the appropriate value from the terminfo database or if that is not available, it uses a default of `80`.

HOME Contains the pathname of your home directory.

SHELL Contains the full pathname of the current shell.

TMPDIR The pathname of the directory being used for temporary files. If it is not set, **ed** uses **/tmp**.

Files

ed uses the following files:

/tmp/e* This is the *paging file*. It holds a copy of the file being edited. You can change the directory for temporary files using the environment variable **TMPDIR**.

ed.hup **ed** writes the current buffer to this file when it receives a hangup signal.

Localization

ed uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion

1 Failure due to any of the following:

- Addressed line out of range
- Only one filename is allowed
- No space for the line table
- Temporary file error
- Badly constructed regular expression
- No remembered regular expression
- File read error
- Out of memory
- Unknown command
- Command suffix not permitted
- No match found for regular expression
- Wrong number of addresses for the subcommand
- Not enough space after the subcommand
- The name is too long
- Badly formed name
- Subcommand redirection is not permitted
- Restricted shell
- No remembered filename
- The mark name must be lowercase
- The mark name is not defined
- **m** and **t** subcommands require a destination address
- The destination cannot straddle source in **m** and **t**
- A subcommand not allowed inside **g**, **v**, **G**, or **V**
- The **x** subcommand has become X (uppercase)
- The global command is too long
- Write error (no disk space)

2 Usage error

Messages

The error messages are issued only if **h** or **H** subcommands are used after **ed** displays ?. Possible error messages include:

Destination cannot straddle source in m and t
The range of lines being moved or copied by **m** or **t** cannot include the destination address.

Global command too long
There is a limit on the length of a global instruction (**g** or **v**). See Limits for this limit.

m and t require destination address
You must follow the **m** or **t** subcommands with an address indicating where you want to move or copy text. You omitted this address.

No remembered filename
You tried to run a subcommand that used a remembered filename (for example, you used **w** to write without specifying an output filename). However, there is no remembered filename at present. Run the subcommand again, but specify a filename this time.

Restricted shell
The command line invoked the restricted form of **ed**, but you tried an action that was not allowed in the restricted editor (the **!** subcommand).

Temporary file error
You ran out of space on disk or encountered other errors involving the page file stored in the temporary file.

Warning: file not saved
You entered a subcommand to quit editing the current file, for example, **q** or **e** to edit a new file; however, you have changed the file since the last time you saved it. **ed** is suggesting that you save the file before you exit it; otherwise, your recent changes will be lost. To save the file, use the **w** command. If you really do not want to save the recent changes, use **q** to quit or **e** to edit a new file.

?file
An error occurred during an attempt to open or create *file*. This is applicable to the **e**, **r**, and **w** subcommands.

? An unspecified error occurred. Use the **h** or **H** subcommand for more information. If the input to **ed** comes from a script rather than from a workstation, **ed** exits when any error occurs.

Limits

ed allows a limit of 1024 bytes per line and 28 000 lines per file. It does not allow the NUL ('\0') character. The maximum length of a global command is 256 characters, including newlines.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The addresses < and >, the **–b** option, and the **W** and **X** subcommands are extensions of the POSIX standard.

Related Information

awk, **diff**, **env**, **ex**, **grep**, **sed**, **vi**

See Appendix C for more information about **regexp**.

egrep — Search a file for a specified pattern

Format

egrep [**–bcilnqsvx**] [**–e** *pattern*] ... [**–f** *patternfile*] ... [*pattern*] [*file* ...]

Description

See **grep**.

env — Display or set environment variables for a process

Format

env [**–i**] [*variable*=*value* ...] [*command argument* ...]
env [**–**] [*variable*=*value* ...] [*command argument* ...]

Description

If you enter **env** with no arguments, it displays the environment variable that it received from its parent (presumably the shell).

Arguments of the form *variable*=*value* let you add new environment variables or change the value of existing environment variables.

If you specify *command*, **env** calls *command* with the arguments specified with the *argument* argument that appear on the command line, passing the accumulated environment variable to this command. The *command* is run directly as a program found in the search **PATH,** and is not interpreted by a shell.

In a doublebyte locale, environment variable values may contain doublebyte characters. The equal sign (=) must be singlebyte.

Options

env supports the following two options, both of which have the same effect.

–i Specifies that the environment variable inherited by **env** not be used.

– Specifies that the environment variable inherited by **env** not be used.

Examples

Compare the output of the following two examples:

```
env foo=bar env
env –i foo=bar env
```

Localization

env uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion

1 Failure due to any of the following:

- Not enough memory
- Name is too long

2 Incorrect command-line argument

126
env found *command* but could not invoke it

127
env was unable to find *command*

Portability

POSIX.2, X/Open Portability Guide, UNIX System V.

printenv on Berkeley UNIX systems works like **env**.

Related Information

env, **sh**

---

eval — Construct a command by concatenating arguments

Format

eval [*argument ...*]

tcsh shell: **eval** *argument ...*

Description

The shell evaluates each argument as it would for any command. **eval** then concatenates the resulting strings, separated by spaces, and evaluates and executes this string in the current shell environment.

eval in the tcsh shell

In the tcsh shell, **eval** treats the arguments as input to the shell and executes the resulting command(s) in the context of the current shell. This is usually used to execute commands generated as the result of command or variable substitution, since parsing occurs before these substitutions. See “tcsh — Invoke a C shell” on page 556.

Examples

The command:

```
for a in 1 2 3
do
        eval x$a=fred
done
```

sets variables *x1*, *x2*, and *x3* to `fred`. Then:

```
echo $x1 $x2 $x3
```

produces: fred fred fred

Usage Note

eval is a special built-in shell command.

Localization

eval uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Value

The only possible exit status value is:

`0` You specified no arguments or the specified arguments were empty strings

Otherwise, the exit status of **eval** is the exit status of the command that **eval** runs.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Information

exec, **sh**, **tcsh**

---

ex — Use the ex text editor

Format

ex [**–eRrsvx**] [**+** *command*] [**–c** *command*] [**–t** *tag*] [**–w** *size*] [*file ...*]

Description

ex is the line-editor mode of the **vi** text editor.

Options

The **ex** internal commands are described in **vi**. It supports the following options:

+*command*
: Begins the editing session by running the specified editor *command*. To specify multiple commands, separate them with an “or” bar (|).

–c *command*
: Begins editing by executing the specified editor command. You can specify multiple commands by separating them with an “or” bar (|). *command* can be any **ex** command except those that enter input mode, such as **insert** or **append**.

–e Invokes **ex**. This option is intended for use with **vi**.

–r Lets you recover named files after an editor or system fails. If you don't specify a file argument, **ex** lists all recoverable files and then exits.

–R Sets read-only mode.

–s Suppresses all interactive feedback (quiet mode). This is for batch mode operation; **ex** assumes the terminal cannot display text and ignores the value of **TERM**. **ex** also ignores all startup files and ignores the value of **EXINIT**.

–t *tag*
Edits the file containing the specified *tag* and sets the virtual position in the edit buffer to point of definition for the tag.

–v Invokes **vi**.

–w *size*
Sets the option variable `window` equal to *size.*

–x Uses encryption.

Localization

ex uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Portability

POSIX.2 User Portability Extension, X/Open Portability Guide, UNIX systems.

–x is an extension to the POSIX standard.

Related Information

ed, **vi**

---

exec — Run a command and open, close, or copy the file descriptors

Format

exec [*command_line*]

tcsh shell: **exec** *command*

Description

The *command_line* argument for **exec** specifies a command line for another command. **exec** runs this command without creating a new process. Some people picture this action as *overlaying* the command on top of the currently running shell. Thus, when the command exits, control returns to the parent of the shell.

Input and output redirections are valid in *command_line*. You can change the input and output descriptors of the shell by giving only input and output redirections in the command. For example:

```
exec 2>errors
```

redirects the standard error stream to **errors** in all subsequent commands ran by the shell.

If you do not specify *command_line*, **exec** returns a successful exit status.

exec in the tcsh shell

In the tcsh shell, **exec** executes the specified command in place of the currrent shell. See "tcsh — Invoke a C shell" on page 556.

Usage Note

exec is a special built-in shell command.

Localization

exec uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

If you specify *command_line*, **exec** does not return to the shell. Instead, the shell exits with the exit status of *command_line* or one of the following exit status values:

1–125 A redirection error occurred.

126 The command in *command_line* was found, but it was not an executable utility.

127 The given *command_line* could not be run because the command could not be found in the current **PATH** environment.

If you did not specify *command_line*, **exec** returns with an exit value of zero.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Information

sh, **tcsh**

exit — Return to the shell's parent process or to TSO/E

Format

exit [*expression*]

tcsh shell: **exit** [*expr*]

Description

exit ends the shell. If there is an *expression*, the value of the *expression* is the exit status of the shell.

The value of expression should be between `0` and `255`. The **EXIT** trap is raised by the **exit** command, unless **exit** is being called from inside an **EXIT** trap.

If you have a shell background job running, you cannot exit from the shell until it completes. However, you can switch to subcommand mode and exit.

exit in the tcsh shell

The shell exits either with the value of the specified expression or, without expression, with the value of the **status** variable. The value of expression should be between `0` and `255`. See "tcsh — Invoke a C shell" on page 556.

Usage Note

exit is a special built-in shell command.

Localization

exit uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

exit returns the value of the arithmetic expression specified by the *expression* argument to the parent process as the exit status of the shell. If you omit *expression*, **exit** returns the exit status of the last command run.

Related Information

return, **sh**, **tcsh**

The **exit()** ANSI C function, the **_exit** callable service, and the **_exit()** POSIX C function are unrelated to the **exit** shell command.

expand — Expand tabs to spaces

Format

expand [**–t** *tablist*] [*file ...*]
expand [**–***number*] [**–***number,number ...*] [*file ...*]

Description

expand reads text input from the files specified on the command line, converts tabs into spaces, and writes the result to the standard output (**stdout**). If you do not specify any files on the command line, **expand** reads from the standard input (**stdin**).

expand preserves backspace characters. By default, tab stops are set every eight columns. A tab after the last tabstop is replaced by a space.

Options

The first syntax of **expand** supports the following option:

–t *tablist*
Sets tab stops at positions indicated by *tablist*. Numbers in *tablist* must be in ascending order (origin 0) and separated by commas or blanks; however, the list must be one argument so you need shell quoting if you are using blanks. The list can consist of a single number, in which case tabs are set every *tablist* positions apart.

The second syntax of **expand** (which the POSIX standard considers obsolete) supports the following options:

–*number*
Sets tab stops every *number* columns.

–*number,number ...*
Sets tab stops at each column *number* (origin 0).

Localization

expand uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion
1 Failure due to any of the following:
- Cannot open the input file
- Insufficient memory
- Incorrect tab stop specification

Portability

POSIX.2 User Portability Extension, X/Open Portability Guide, 4.2BSD and higher.

Related Information

pr, **unexpand**

---

export — Set a variable for export

Format

export [*name* [=*value*] ...]
export –p

Description

export marks each variable *name* so that the current shell makes it automatically available to the environment of all commands run from that shell. Exported variables are thus available in the environment to all subsequent commands. Several commands (for example, **cd**, **date** and **vi**) look at environment variables for configuration or option information.

Variable assignments of the form *name=value* assign *value* to *name* as well as marking *name* for export. The *name* can contain only the underscore and alphanumeric characters from the portable character set.

Calling **export** without arguments lists, with appropriate quoting, the names and values of all variables in the format *Variable="value"*. If you reinput this format to another shell, variables are assigned appropriately but not exported. The **–p** option lists variables in a format suitable for reinput to the shell (see the description of the **–p** option).

Option

–p Lists variables in a form that is suitable for reinput to the shell:

```
export name="value"
```

Usage Note

export is a special built-in shell command.

Localization

export uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion

1 Failure due to incorrect command-line argument

2 Failure, usually due to incorrect an incorrect command-line argument, that results in a usage message

Portability

POSIX.2, X/Open Portability Guide.

Assigning a value to *name*, and the behavior given for calling **export** with arguments are extensions of the POSIX standard.

Related Information

cd, **date**, **set**, **sh**, **typeset**, **vi**

---

expr — Evaluate arguments as an expression

Format

expr *expression*

Description

The set of arguments passed to **expr** constitutes an expression to be evaluated. Each command argument is a separate token of the expression. **expr** writes the result of the expression on the standard output. This command is primarily intended for arithmetic and string manipulation on shell variables.

expr supports the following operators. Operators explained together have equal precedence; otherwise, they are in increasing order of precedence. **expr** stores an expression as a string and converts it to a number during the operation. If the context requires a Boolean value, a numeric value of 0 (zero) or a null string ("") is *false*, and any other value is *true*. Numbers have an optional leading sign, followed by either a hexadecimal, an octal, or a decimal number. The shell differentiates between hex, octal, and decimal as follows:

- Any number that starts with 0x is hex.
- Any number that starts with 0 is octal.
- Any number that does not start with 0x or 0 is decimal.

Numbers are manipulated as long integers.

expr1 **|** *expr2*
Results in the value *expr1* if *expr1* is true; otherwise, it results in the value of *expr2*.

expr1 **&** *expr2*
Results in the value of *expr1* if both expressions are true; otherwise, it results in 0.

expr1 **<=** *expr2* **|** *expr1* **<** *expr2* **|** *expr1* **=** *expr2* **|** *expr1* **!=** *expr2* **|** *expr1* **>=** *expr2* **|** *expr1* **>** *expr2*
If both *expr1* and *expr2* are numeric, **expr** compares them as numbers; other-

wise, it compares them as strings. If the comparison is true, the expression results in `1`; otherwise, it results in `0`.

expr1 + *expr2* **|** *expr1* **–** *expr2*
Performs addition or subtraction on the two expressions. If either expression is not a number, **expr** exits with an error.

expr1 ***** *expr2* **|** *expr1* **/** *expr2* **|** *expr1* **%** *expr2*
Performs multiplication, division, or modulus on the two expressions. If either expression is not a number, **expr** exits with an error.

expr1 **:** *re* **|** **match** *expr1 re*
matches the regular expression *re* against *expr1* treated as a string. The regular expression is the same as that accepted by **ed**, except that the match is always anchored—that is, there is an implied leading ^. Therefore, **expr** does not consider ^ to be a metacharacter. If the regular expression contains \(**...**\), **\)** and it matches at least part of *expr1*, **expr** results in only that part; if there is no match, **expr** results in `0`. If the regular expression doesn't contain this construct, the result is the number of characters matched. The function **match** performs the same operation as the colon operator.

substr *expr1 expr2 expr3*
Results in the substring of *expr1* starting at position *expr2* (origin 1) for the length of *expr3*.

index *expr1 expr2*
Searches for any of the characters in *expr2* in *expr1* and results in the offset of any such character (origin 1), or `0` if no such characters are found.

length *expr1*
Results in the length of *expr1*.

(*expr***)**
Groups expressions.

Usage Note

The parser stack depth is limited to 150 levels. Attempting to process extremely complicated expressions may result in an overflow of this stack, causing an error.

Examples

1. The example

   ```
   fname=src/fn_abs.c
   expr $fname : '*_\(.*\)\.c'.
   ```

   returns `abs`.

2. The example

   ```
   a= expr $a + 1
   ```

   adds 1 to the value of the shell variable *a*.

Localization

expr uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**

- **LC_MESSAGES**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 The result of *expression* is true.
1 The result of *expression* is false.
2 Failure due to any of following:
- Not enough memory.
- Command-line syntax error.
- Too few arguments on the command line.
- Incorrect regular expression.
- Regular expression is too complicated.
- Nonnumeric value found where a number was expected.

Messages

Possible error messages include:

internal tree error
Syntax errors or unusual expression complexity make it impossible for **expr** to evaluate an expression. If an expression has syntax errors, correct them; if not, simplify the expression (perhaps by breaking it into parts).

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

In the shell, **let** largely supersedes this command.

match, **substr**, **length**, and **index** are not documented on all UNIX systems, though they do appear to exist. They are extensions of the POSIX standard.

Related Information

ed, **let**, **sh**, **test**

See Appendix C for more information about **regexp**.

---

exrecover daemon — Retrieve vi and ex files

Format

exrecover [**–s**] [*name_file* ...]
exrecover [**–v**]

Description

The **exrecover** daemon recovers text files from working files created by **vi** and **ex**. (These working files are in one or more temporary directories.) It is normally invoked from a system startup file before these working files are purged.

Options

–s Suppresses error messages.

–v Displays the version number of **exrecover**.

Environment Variables

exrecover uses the following environment variables:

TMPDIR
The default directory. When this environment variable is set, **exrecover** looks in this directory for the **ex** and **vi** working files.

TMP If **TMPDIR** is not set, **TMP** specifies the directory to be searched when looking for the **ex** and **vi** working files.

If both **TMPDIR** and **TMP** are not set, **exrecover** uses the directory that the C/C++ Run-Time Library function **tempnam()** would use.

Localization

exrecover uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Files

exrecover uses the following files:

/tmp/Vll*
Line table files.

/tmp/Vln*
Name files.

/tmp/Vlt*
Paged text files.

/etc/recover
The directory containing subdirectories of user names whose files have been recovered. Only users with the appropriate privileges, such as the system administrator, can create the **/etc/recover** directory.

/etc/recover/$LOGNAME/Vln*
Contains the name of the file that was being edited.

/etc/recover/$LOGNAME/Vlt*
Contains the recovered text of the file that corresponds to the Vln* file

Note: Using a TFS for **/tmp** will make it impossible to recover **vi** files after a system crash. **vi** writes temporary files to **TMPDIR** (**/tmp** by default) and if the system crashes, those files can be recovered by the **exrecover** command, which automatically runs from **/etc/rc**. If **/tmp** is a TFS then all the **vi** temporary files will be wiped out.

Usage Notes

1. To recover all the files in the temporary directory, this command must be run with appropriate permissions (for example, superuser privileges) so the recovered files can be stored in the **/etc/recover** directory with the appropriate ownerships and permissions.

   For example, the following is a shell script to recover the files from **TMPDIR**, where **TMPDIR** is the default directory:

   ```
   export TMPDIR=/tmp
   exrecover
   ```

2. If it is invoked by a nonprivileged user (for example, a user who is not a root user), then only those files owned by that user are recovered. Because **vi** and **ex** create their working files in directories specified by the **TMPDIR** or **TMP** environment variables, one of these environment variables must be set before **exrecover** can be issued.

   For example, the following is a shell script that recovers files from **$HOME/tmp**:

   ```
   export TMPDIR=$HOME/tmp
   exrecover
   ```

3. **exrecover** is also invoked by **vi** or **ex** when you issue the **ex preserve** command or when **exrecover** receives a SIGHUP signal. The working files created by **vi** and **ex** are found in a default temporary directory (such as **/tmp**) or in the directory specified by the **TMPDIR** or the **TMP** environment variable if either one is set. Three working files are created:

   name_file Contains the actual name of the **vi** file. The names of all *name_files* begin with **VIn**.

   line_table_file
   Contains a dummy page followed by data that gives, in line number order, the offset for each line of text in the corresponding *paged_text_file*. The page size is typically 1K, but may vary on some systems. The names of all line table files begin with **VIl**.

   paged_text_file
   Contains lines of text that are at most LINE_MAX bytes in length. Lines shorter than LINE_MAX byte are ended by a newline. The names of all paged text files begin with **VIt**.

4. You can also run the program by specifying *name_file* on the command line. For example:

   ```
   exrecover /tmp/VInaaaa.111 /tmp/VInbbbb.222 ...
   ```

   exrecover searches for a *name_file* and tries to open the associated line table and paged text files. If all these files are found, **exrecover** builds, from the line table and paged text files, a text file and stores it in the directory **/recover/$LOGNAME**.

   It also stores a corresponding *name_file* to identify the file that was recovered and sends mail, using the **mailx** utility, to the owner of the file indicating the date, time, and name of the file recovered. You can retrieve recovered files in one of the following ways:

```
vi –r file
ex –r file
:recover file
```

Each command loads the oldest occurrence of the file recovered from a system failure or the **ex preserve** command. If **vi** successfully loads the file, it removes the preserved file.

Exit Values

0 Successful completion

1 Failure due to any of the following:

- Memory allocation error
- No working files were found
- No **/etc/recover** directory errors that affect the overall operation of the **exrecover** command
- An incomplete set of working files were found

2 Usage error

3 An error occurred while recovering a specific file. Some, but not all, files were recovered.

Related Information

ex, **vi**

---

extattr — Set, reset, and display extended attributes for files

Format

extattr [+**apsl**] [–**apsl]** *file ...*

Note: **l** is a lower case L, not an upper case i.

Description

extattr sets, resets, and displays extended attributes for files.

Options

The following extended attributes are defined:

a When this attribute is set (**+a**) on an executable program file (load module), it behaves as if loaded from an APF-authorized library. For example, if this program is exec()ed at the job step level and the program is linked with the AC=1 attribute, the program will be executed as APF-authorized.

To be able to use the **extattr** command for the **+a** option, you must have at least READ access to the BPX.FILEATTR.APF FACILITY class profile. For more information, see *OS/390 UNIX System Services Planning*.

l When this attribute is set (**+l**) on an executable program file (load module), it will be loaded from the shared library region.

To be able to use the **extattr** command for the **+l** option, you must have at

least READ access to the BPX.FILEATTR.SHARELIB FACILITY class. For more information, see *OS/390 UNIX System Services Planning*.

Note: **l** is a lower case L, not an upper case i.

p When this attribute is set (**+p**) on an executable program file (load module), it causes the program to behave as if an RDEFINE had been done for the load module to the PROGRAM class. When this program is brought into storage, it does not cause the environment to be marked dirty.

To be able to use the **extattr** command for the **+p** option, you must have at least READ access to the BPX.FILEATTR.PROGCTL FACILITY class. For more information, see *OS/390 UNIX System Services Planning*.

s When this attribute is not set (**–s**), the **_BPX_SHAREAS** environment variable is ignored when the file is spawn()ed. By default, this attribute is set (**+s**) for all executable files.

Note: To specify any of these attributes, the user must be the owner of the file or have superuser authority.

Example

To have the **c89** and **tso** utilities not run in an address space shared with other processes, issue:

```
extattr -s /bin/c89 /bin/tso
```

Related Information

ls, **ISHELL**

---

false — Return a nonzero exit code

Format

false [*argument ...*]

Description

false returns an exit status value of 1 (failure). It ignores any arguments given on the command line. This can be useful in shell scripts.

Usage Note

false is a built-in shell command.

Localization

false uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Value

false always returns an exit status value of 1.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Information

sh

---

fc — Process a command history list

Format

fc [–r] [**–e** *editor*] [*first*[*last*]]
fc –l [**–nr**] [*first*[*last*]]
fc –s [*old*=*new*] [*specifier*]

Description

fc displays, edits, and reenters commands that have been input to an interactive shell. **fc** stands for "fix commands." If the variable **HISTSIZE** is not defined, 128 commands are accessible. The number of commands that are accessible is determined by the **HISTSIZE** variable.

The shell stores these commands in a history file. When the **HISTFILE** environment variable is defined as the name of a writable file, the shell uses this as the history file. Otherwise, the history file is **$HOME /.sh_history**, if **HOME** is defined and the file is writable. If the **HOME** variable is not defined, or the file is not writable, the shell attempts to create a temporary file for the history. If a temporary file cannot be created, the shell does not keep a history file.

Note: A shell shares history (commands) with all shells that have the same history file. A login shell truncates the history file if it is more than **HISTSIZE** lines long.

Normally, the shell does not keep a history of commands run from a profile file or the **ENV** file. By default, however, it begins recording commands in the history file when it encounters a function definition in either of these setup files. This means that the **HISTSIZE** and **HISTFILE** variables must be set up appropriately before the first function definition. If you do not want the history file to begin at this time, use:

```
set -o nolog
```

For further information, see **sh** and **set**. Any variable assignment or redirection that appears on the **fc** command line affects both the **fc** command itself and the commands that **fc** produces.

The first form of **fc** in "Format" puts you into an editor with a range of commands to edit. When you leave the editor, **fc** inputs the edited commands to the shell.

The first and last command in the range are specified with *first* and *last*. There are three ways to specify a command.

- If the command specifier is an unsigned or positive number, **fc** edits the command with that number.
- If the command specifier is a negative number *–n*, **fc** edits the command that came *n* commands before the current command.
- If the command specifier is a string, **fc** edits the most recent command beginning with that string.

The default value of *last* is *first*. If you specify neither *first* nor *last*, the default command range is the previous command entered to the shell.

Options

–e *editor*
Invokes *editor* to edit the commands. If you do not specify the **–e** option, **fc** assumes that the environment variable **FCEDIT**, if defined, contains the name of the editor for **fc** to use. If **FCEDIT** is not defined, **fc** invokes **ed** to edit the commands.

–l Displays the command list. This option does not edit or reenter the commands. If you omit *last* with this option, **fc** displays all commands from the one indicated by *first* through to the previous command entered. If you omit both *first* and *last* with this option, the default command range is the 16 most recently entered commands.

–n Suppresses command numbers when displaying commands.

–r Reverses the order of the commands in the command range.

–s Reenters exactly one command without going through an editor. If a command *specifier* is given, **fc** selects the command to reenter as described earlier; otherwise, **fc** uses the last command entered. To perform a simple substitution on the command before reentry, use a parameter of the form *old*=*new*. The string *new* replaces the first occurrence of string *old*. **fc** displays the (possibly modified) command before reentering it.

Environment Variables

fc uses the following environment variables:

FCEDIT Contains the default editor to be used if none is specified with the **–e** option.

HISTFILE Contains the pathname of the history file.

HISTSIZE
Gives the maximum number of previous commands that are accessible.

Files

/tmp
Used to store temporary files. You can use the **TMPDIR** environment variable to dictate a different directory to store temporary files.

$HOME/.sh_history
This default history file is created.

Localization

fc uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Usage Notes

1. **fc** is a built-in shell command.
2. **r** is a built-in alias for **fc –s**. **history** is a built-in alias for **fc –l**.

Exit Values

0 If you specified **–l**, this indicates successful completion.

1 Failure due to any of the following:

- Missing history file
- Inability to find the desired line in the history file
- Inability to create temporary file

2 An incorrect command-line option or argument

If **fc** runs one or more commands, the exit status of **fc** is the exit status of the last run command.

Messages

Possible error messages include:

Cannot create temporary file
fc must create a temporary file to do some operations, such as editing. It prints this message when it cannot create its temporary file—for example, because the disk is full.

No command matches *string*
You asked to edit a command beginning with a particular *string*, but there was no such command in the history file.

Portability

POSIX.2.

Related Information

alias, **ed**, **print**, **read**, **sh**, **vi**

fg — Bring a job into the foreground

Format

fg [**%***job-identifier*]

tcsh shell: **fg** [*%job ...*]

Description

fg restarts a suspended job or moves a job from the background to the foreground. To identify the job, you give a *job-identifier* (preceded by **%**) as given by the **jobs** command.

If you do not specify *job-identifier*, **fg** uses the most recent job to be suspended (with the **kill** command) or placed in the background (with the **bg** command). **fg** is available only if you have enabled job control. See the **–m** option of **set** for more information.

fg in the tcsh shell

In the tcsh shell, **fg** brings the specified jobs (or, without arguments, the current job) into the foreground, continuing each if it is stopped. *job* may be '', %, +, –, a number, or a string. See also the **run-fg-editor** editor command described in “tcsh — Invoke a C shell” on page 556.

Localization

fg uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

`0` Successful completion

`>0`
No current job

Messages

Possible error messages include:

Not a stopped job Job was not stopped.

Portability

POSIX.2 User Portability Extension.

Related Information

bg, **jobs**, **kill**, **ps**, **tcsh**

---

fgrep — Search a file for a specified pattern

Format

fgrep [**–bcilnqsvx**] [**–e** *pattern*] ... [**–f** *patternfile*] ... [*pattern*] [*file* ...]

Description

See **grep**.

---

file — Determine file type

Format

file [**–c**] [**–f** *filelist*] [**–m** *magic*] *file ...*

Description

file determines the format of each file by inspecting the attributes and (for an ordinary file) reading an initial part of the file. **file** compares each file on the command line to templates found in a system-maintained **magic** file to determine their file type.

file then divides files that do not match a template in the **magic** file into text files and binary data. Then, by reading an initial segment of the text files and making an informed guess based on the contents, **file** further divides text files into various types such as C programs, assembler programs, files of commands to the shell, and **yacc** or **lex** programs.

file displays the name of each file along with the file type that it believes the file to be.

Options

–c Only checks the template file of **magic** numbers for validity of format.

–f *filelist*
Examines the files listed in the file *filelist*.

–m *magic*
Uses the file **magic** rather than the default file of filetype templates.

Localization

file uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Files

file uses the following file:

/etc/magic Default template file of **magic** numbers.

For **file** to work, you need to copy the **magic** file from the **/samples** directory to the **/etc** directory.

For more information on enabling **file**, see "Enabling the **file** Utility" in *OS/390 UNIX System Services Planning*. Additional information on the **magic** file can be found in "magic — Format of the /etc/magic file" on page 822.

Usage Notes

LC_SYNTAX only affects the interpretation of the input file that did not match any **magic** file template. It does not affect the interpretation of the **magic** file.

Because of this, an input file that contains characters from a code page other than IBM-1047 cannot match the **magic** file, which contains IBM-1047 characters.

Exit Values

0 Successful completion

1 Failure due to any of the following:

- A missing *filelist* after **–f**
- More than one **–f** option on the command line
- Cannot find the **magic** file
- Incorrect command-line option
- Too few command-line arguments
- Cannot access a specified file
- Cannot open *filelist*
- Cannot open the **magic** file
- A format error in the **magic** file
- Out of memory for reading or **magic** entries
- A bad number in the **magic** file
- A misplaced > in the **magic** file.

Portability

POSIX.2 User Portability Extension, X/Open Portability Guide, UNIX systems.

All options are extensions to the POSIX standard.

Related Information

See Appendix H for more information about the **magic** file format.

filecache — Manage file caches

Format

/usr/sbin/filecache –a *file...*

/usr/sbin/filecache –d *file...*

/usr/sbin/filecache [–p | –q | –r]

Description

filecache manages the kernel file cache for files that are only read. To prime the file cache at system startup, you can run this command from the **/etc/rc** script to automatically load the file cache with files known to be used for read access only. Files can be either data or executable files residing in a local file system (such as HFS file systems). Specify all file names as full pathnames.

This command requires superuser authority.

Options

–a Adds the specified files to the file cache.

–d Deletes the specified files from the file cache.

–p Purges all files in the file cache.

–q Lists the files in the file cache.

–r Refreshes all files in the file cache.

Examples

To load the message catalogs for the shell and most utilities into the cache:

```
/usr/sbin/filecache -a /usr/lib/nls/msg/C/fom* /usr/lib/nls/msg/C/fsum*
```

Usage Notes

- When specifying *file* on the filecache command, use the full pathname.
- Files that have been modified may no longer be eligible for caching until the file system that the file resides in is unmounted and then re-mounted, or until the system is IPLed.

find — Find a file meeting specified criteria

Format

find *path ... expression*

Description

find searches a given file hierarchy specified by *path*, finding files that match the criteria given by *expression*. Each directory, file, and special file is “passed through” *expression*. If you use the **–exec**, **–ok**, or **–cpio** primary, *expression* runs a specified command on each file found. A nonexistent *expression* or an *expression* with

commands to run automatically uses the **–print** primary to display the name of any file that matches the criteria of *expression*.

find builds *expression* from a set of primaries and operators; juxtaposition of two primaries implies a logical AND operator.

Operators and Primaries

find supports the following operators:

–a Used between primaries for a logical AND. You can omit this operator to get the same result, since logical AND is assumed when no operator is used between two primaries.

–o Used between primaries for a logical OR.

! Precedes an expression in order to negate it.

You can group primaries and operators using parentheses. You must delimit all primaries, operators, numbers, arguments, and parentheses with white space. Each *number* noted in the primary list is a decimal number, optionally preceded by a plus or minus sign. If a number is given without a sign, **find** tests for equality; a plus sign implies “greater than” or “older than,” and a minus sign implies “less than” or “newer than”.

Options

find supports the following primaries:

–aaudit *auditmask*
The **-aaudit** primary is used to match the auditor audit bits. See **–audit** *auditmask*.

–audit *auditmask*
The **-audit** primary is used to match the user audit bits. *auditmask* can be in octal or in symbolic form. The mask can be preceded by a **-** character (as in the **perm** primary), but it is ignored. Symbolic mode is an *operation=condition* list, separated by commas:

```
[rwx]=[sf]
```

where:

=sf	Success or failure on any of **rwx**
r=s	Success on **read**
r=s, x=sf	Success on **read** or **exec**, failure on **exec**
r, w=s	Incorrect
x	Incorrect

–atime *number*
Matches if someone has accessed the file exactly *number* days ago.

–cpio *cpio-file*
Writes the file found to the target file *cpio-file* in **cpio** format. This is equivalent to:

```
find ... | cpio -o >cpio-file
```

This primary matches if the command succeeds.

–ctime *number*
Matches if someone has changed the attributes of the file exactly *number* days ago.

–depth
Processes directories after their contents. If present, this primary always matches.

–exec *command* **;**
Takes all arguments between **–exec** and the semicolon as a command line, replacing any argument that is exactly {} (that is, the two brace characters) with the current filename. It then executes the resulting command line, treating a return status of zero from this command as a successful match, nonzero as failure. You must delimit the terminal semicolon with white space.

Note: The semicolon is a shell metacharacter. To use it in *expression*, you must you must escape it, either by enclosing it in single quotes or by preceding it with –.

–follow
Follows symbolic links. If present, this primary always matches.

–group *name*
Matches if the group owner is *name*. If *name* is not a valid group name, it is treated as a group ID.

–inum *number*
Matches if the file has inode number *number*.

–level *number*
Does not descend below *number* levels.

–links *number*
Matches if there are *number* links to the file.

–mtime *number*
Matches if someone has modified the file exactly *number* days ago.

–name *pattern*
Compares the current filename with *pattern*. If there is no match, the expression fails. The pattern uses the same syntax as filename generation (see **sh**). It matches as many trailing pathname components as specified in *pattern*.

–ncpio *cpio-file*
Writes the file found to the target file *cpio-file* in **cpio –c** format. This is equivalent to:

```
find ... | cpio -oc >cpio-file
```

This primary matches if the command succeeds.

–newer *file*
Compares the modification date of the found file with that of the *file* given. This matches if someone has modified the found file more recently than *file*.

–nogroup
Matches if no defined group owns the file.

–none
Indicates that some action has been taken; thus **find** does not invoke the default **–print** action. If present, this primary always matches.

–nouser
Matches if no defined user owns the file.

–ok*command***;**
Is similar to **–exec**, but before **find** executes the command, it displays the command to confirm that you want to go ahead. **find** executes the command line only if your input matches the expression for "yes" (yes and no expressions are defined in **LC_MESSAGES**). If you type the expression for "no", the primary does not match. You must delimit the terminal semicolon with white space.

Note: The semicolon is a shell metacharacter. To use it in *expression*, you must quote it.

–perm[-] *mask*
By default, matches if the permissions on file are identical to the ones given in *mask*. You can specify *mask* in octal or in symbolic mode (see **chmod**). If you use symbolic mode, **find** assumes that you begin with no bits set in *mask*, and that the symbolic mode is a recipe for turning the bits you want on and off. A leading minus sign (–) is special. It means that a file matches if at least all the bits in *mask* are set. As a result, with symbolic mode, you cannot use a *mask* value that begins with a minus sign (–).

If you use octal mode, **find** uses only the bottom 12 bits of *mask*. With an initial minus sign (–), **find** again matches only if at least all the limits in *mask* are set in the file permissions lists.

–print
Displays the current filename. This primary always matches.

–prune
Stops searching deeper into the tree at this point. If present, this primary always matches. **–prune** has no effect if **–depth** is also specified.

–size *number***[**c**]**
Matches if the size of the file is *number* blocks long, where a block is 512 bytes. If you include the suffix *c*, the file size is *number* bytes.

–type *c*
Matches if the type of the file is the same as the type given by the character *c*. Possible values of the character are:

b — Block special file (not supported for OS/390 UNIX System Services)
c — Character special file
d — Directory
f — Regular file
l — Symbolic link
n — Network file
p — FIFO (named pipe)
s — Socket

–user *name*
Matches if the owner of the file is *name*. *name* can also be a user ID number.

–xdev
Does not cross device boundaries from the root of the tree search. If present, this primary always matches.

Examples

1. To find all files with a suffix of **.c** that have the audit mode set to **rwx** (read, write, execute), issue:

   ```
   find / –name *.c –audit rwx=sf
   ```

2. To find all files with a suffix of **.c** and audit mode bits set to 777 (**rwx**), issue:

   ```
   find / –name *.c –audit 777
   ```

3. To find all files with the extensions **.c** and **.h**, starting at the current point in the directory hierarchy:

   ```
   find . –name "*.[ch]"
   ```

4. To find all files that have the extension **.Z** and that have not been accessed in the last three days:

   ```
   find . –name "*.Z" –mtime +3
   ```

Localization

find uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Values

`0` Successful completion

`1` Failure due to any of the following:

- Not enough memory
- Incorrect character specified after **–type**
- Inability to get information on a file for **–newer**
- Incorrect permissions for **–perm**
- Inability to open a file for the **–cpio** option
- Unknown user or group name
- Unable to access the **PATH** variable
- Cannot run a command specified for **–exec** or **–ok**
- Syntax error
- Stack overflow caused by an expression that is too complex

`2` Failure due to one of the following:

- Incorrect command-line option
- Not enough arguments on the command line
- Missing option
- Argument list that is not properly ended

Messages

Possible error messages include:

bad number specification in *string*
You specified an option that takes a numeric value (for example, **–atime**, **–ctime**) but did not specify a valid number after the option.

cannot stat file *name* **for -newer**
You used a **–newer** option to compare one file with another; however, **find** could not obtain a modification time for the specified file. Typically, this happens because the file does not exist or you do not have appropriate permissions to obtain this information.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Most UNIX systems do not have a default action of **–print**. Therefore, they do not need the **–none** option. The **–a** operator is not documented on many UNIX systems.

The **–aaudit**, **–audit**, **–cpio**, **–follow**, **–level**, **–ncpio**, and **–none** primaries are extensions of the POSIX standard.

The **aaudit** and **audit** options are unique to the OS/390 shell.

Related Information

chaudit, **chmod**, **cpio**, **sh**

---

fold — Break lines into shorter lines

Format

fold [**–bs**] [**–w** *width*] [**–***width*] [*file*...]

Description

fold reads the standard input (**stdin**) or each file, if you specify any. Each input line is broken into lines no longer than *width* characters. If you do not specify *width* on the command line, the default line length is `80`. The output is sent to the standard output (**stdout**).

Options

–b Specifies *width* in bytes rather than in column positions; that is, **fold** does not interpret tab, backspace, and carriage return characters. If the last byte specified by *width* is part of a doublebyte character, **fold** does not break the character. Instead, the line is broken before the doublebyte character.

–s Breaks each line at the last blank within *width* column positions. If there is no blank that meets the requirement, **fold** breaks the line normally.

–w *width*
Specifies a maximum line length of *width* characters.

–width
Is identical in effect to **–w** *width*.

Localization

fold uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion
1 Failure because the input file could not be opened
2 Invalid command-line option or a missing *width* argument

Portability

POSIX.2, 4.2BSD.

The *–width* option is an extension of the POSIX standard.

Related Information

pr

---

functions — Display or assign attributes to functions

Format

functions [***tux**][*name ...*]

Note: * indicates a **+** or **–** character.

Description

functions is an alias for **typeset –f**.

See “typeset — Assign attributes and values to variables” on page 646 for more information.

Related Information

typeset, **sh**

---

fuser — List process IDs of processes with open files

Format

fuser [**–cfku**] *file ...*

Description

The **fuser** command writes to standard output the process IDs of all processes running on the local system that have one or more named files open. *file* is the pathname of the file for which information is to be reported, or, if the **–c** options is used, the pathname of a file on the file system for which information is to be reported.

The **fuser** command writes additional information to standard error, such as the user name of the process and a character indicating how the process is using the file. **fuser** only reports on local processes, not remote ones.

Option

–c **fuser** reports on all open files within the file system that the specified file is a member of

–f **fuser** reports on only the named files. This is the default.

–k **fuser** sends the SIGKILL signal to each local process (with the exception of the **fuser** process and parent processes of **fuser**). Only a superuser can terminate a process that belongs to another user. This option is an OS/390 extension.

–u **fuser** writes to standard error the user name associated with each process ID written to standard output.

Usage Notes

fuser will write the process ID for each process to standard output. **fuser** also writes the following to standard error:

- The pathname of each file specified on the command line.
- An indicator of how the process is using this file (written after the process ID):

  –c The process is using the file as its current directory.

  –r The process is using the file as its root directory.

  If no character follows the PID, this means that the process has the file open.
- When the **–u** option is specified, **fuser** writes the user name corresponding to the process' real user ID.

Examples

1. To list the process numbers of local processes using the **/etc/magic** file, enter:

   ```
   fuser /etc/magic
   ```

   which will give you the following output:

   ```
   /etc/magic:  67109274 144
   ```

2. To display the user names associated with the processes accessing the file **/etc/magic**:

   ```
   fuser -u /etc/magic
   ```

   Your output would be:

```
/etc/magic:  67109274(Steve) 144(Fred)
```

3. To terminate all of the processes using a given file system, enter either the mount point name or the name of a file in that file system:

   ```
   fuser -ku /u/home
   ```

   or

   ```
   fuser -kuc /u/home/code
   ```

   Your output would look like:

   ```
   /u/home/code:  111111c(Steve) 222222r(Erin) 333333(Garth)
   444444c(Steve) 555555r(Renata) 66666c(Angie)
   ```

   This command lists the process number and user name, and then sends a kill signal to each process that is using the **/u/home** file system. Only a superuser can terminate processes that belong to another user. You might want to use this command if you are trying to unmount the **/u/home** file system and a process that is accessing the file system prevents this.

Exit Values

0	Successful completion
1	An error

Related Information

kill, **ps**

---

gencat — Create or modify message catalogs

Format

gencat *CatalogFile MessageFile ...*

Description

gencat merges the message text source files *MessageFile* (usually ***.msg**) into a formatted message catalog *CatalogFile* (usually ***.cat**). The file *CatalogFile* is created if it does not already exist. If *CatalogFile* does exist, its messages are modified according to the directives in the *MessageFiles*. If set and message numbers are the same, the new message text defined in *MessageFile* replaces the message text defined in *CatalogFile*.

You can specify any number of *MessageFiles*. **gencat** processes the *MessageFiles* one after another, in the sequence specified. Each successive *MessageFile* modifies the *CatalogFile*.

If **–** is specified for *CatalogFile*, standard output (**stdout**) is used. If **–** is specified for *MessageFile*, standard input (**stdin**) is used.

gencat does not accept symbolic message identifiers. You must use **mkcatdefs** if you want to use symbolic message identifiers.

Extended Description

The format of a message text source file is defined as follows. All characters must be encoded as singlebyte characters except where noted. The fields of a message text source line are separated by a single blank character. Any other blank characters are considered as being a part of the subsequent field.

$set *n comment*
: Specifies the set identifier of the following messages until the next **$set** or end of file appears. The *n* denotes the set identifier, which is defined as a number in the range 1–NL_SETMAX. Set identifiers must be in ascending order within a single source file, but need not be contiguous. Any string following the set identifier is treated as a comment. If no **$set** directive is specified in a message text source file, all messages are located in default message set NL_SETD.

$delset *n comment*
: Deletes message set *n* from an existing message catalog. The *n* denotes the set number, 1–NL_SETMAX. Any string following the set number is treated as a comment.

$ *comment*
: A line beginning with **$** followed by a blank character is treated as a comment.

m message-text
: The *m* denotes the message identifier, which is defined as a number in the range 1–NL_MSGMAX. Message identifiers must be in ascending order within a single set, but need not be contiguous. The length of *message-text* must be in the range –NL_TEXTMAX. The message text is stored in the message catalog with the set identifier specified by the last **$set** directive, and with message identifier *m*. If the message text is empty, and a blank character field separator is present, an empty string is stored in the message catalog. If a message source line has a message number, but not a field separator or message text, the existing message with that number (if any) is deleted from the catalog. The message text can contain doublebyte characters.

$quote *c*
: Specifies an optional quote character *c*, which can be used to surround *message-text* so trailing spaces or null (empty) messages are visible in a message source line. By default, or if an empty **$quote** directive is supplied, no quoting of *message-text* is recognized. The quote character can be a doublebyte character.

$timestamp
: Specifies a time stamp that can be used to identity the subsequent **.cat** file as having come from this file. The timestamp can be up to 20 characters long and can be any format you wish. Usually it follows this format:

```
$timestamp 1994 137 19:09 UTC
```

The **mkcatdefs** command automatically generates a timestamp in the file it creates for input to **gencat.**

Empty lines in a message text source file are ignored. Lines starting with any character other than those defined above are considered invalid.

Text strings can contain the special characters and escape sequences defined in the following table:

Description	Sequence
Backspace	**\b**
backslash	\
Carriage-return	**\r**
Double quote	\"
Form-feed	**\f**
Horizontal tab	**\t**
Newline	**\n**
Octal bit pattern	**\ddd**
Vertical tab	**\v**

These sequences must be encoded as singlebyte characters.

The escape sequence **\ddd** consists of backslash followed by one, two, or three octal digits, which are taken to specify the value of the desired character. If the character following a backslash is not one of those specified, the backslash is ignored.

A backslash (\) followed by a newline character is also used to continue a string on the following line. Thus the following two lines describe a single message string:

```
1 This line continues \
to the next line
```

which is equivalent to:

```
1 This line continues to the next line
```

Portability of Message Catalogs

gencat works with the OS/390 C runtime library function **catgets()** to correctly display message text in the locale that the C program using **catgets()** is running in. **gencat** adds information to the *CatalogFile* about the codeset that was in effect when **gencat** processed the *CatalogFile*. **gencat** should be run with the same locale that the recipients of the messages will be using. This should be the same locale that was used to create the message text in **MessageFile**.

Message catalogs produced by **gencat** are binary-encoded, meaning that their portability cannot be guaranteed between systems. Thus, just as C programs need to be recompiled, so message catalogs must be re-created via **gencat** when moved to another system.

Example

To generate a **test.cat** catalog from the source file **test.msg**, enter:

```
gencat test.cat test.msg
```

Localization

gencat uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_SYNTAX**
- **NLSPATH**

Exit Values

0 Successful completion
1 An error occurred

Portability

X/Open Portability Guide.

---

getconf — Get configuration values

Format

getconf –a
getconf *system_var*
getconf [**–a**] *path_var pathname*

Description

getconf writes the value of a configuration variable to the standard output (**stdout**). You can specify the configuration variable using one of forms listed in the Format section. If you use the first form, **getconf** writes the value of the variable *system_var*. If you use the second form, **getconf** writes the value of the variable *path_var* for the pathname given by *pathname*. The **–a** option prompts **getconf** to display all current configuration variables, and their values, to **stdout**.

getconf writes numeric values in decimal format and nonnumeric values as simple strings. If the value is not defined, **getconf** writes the string `undefined` to **stdout**.

Options

–a Writes out all the configuration variables for the current system, and their values, to **stdout**. Path variables are written based on a pathname of dot (**.**).

Configuration Variables

You can use the second form of **getconf** to find the value of the following POSIX.1-1990 standard configuration variables for the specified pathname:

`LINK_MAX`
Specifies the maximum number of links that this file can have.

`MAX_CANON`
Specifies the maximum number of bytes in the workstation's canonical input queue (before line editing).

`MAX_INPUT`
Specifies the space available in the workstation's input queue.

`NAME_MAX`
Specifies the largest filename size.

`PATH` Specifies the standard **PATH** setting.

`_CS_PATH`
Specifies the standard **PATH** setting.

`PATH_MAX`
Specifies the maximum number of bytes in a pathname.

PIPE_BUF
: Specifies the largest atomic write to a pipe.

_POSIX_CHOWN_RESTRICTED
: Specifies the restrictions that apply to file ownership changes.

_POSIX_NO_TRUNC
: If set, it is an error for any pathname component to be longer than NAME_MAX bytes.

_POSIX_VDISABLE
: Specifies that processes are allowed to disable ending special characters.

You can use the first form of **getconf** to find the value of the following POSIX.1-1990 standard configuration variables:

ARG_MAX
: Specifies the maximum length of arguments for running a program, including environment data.

CHILD_MAX
: Specifies the maximum number of simultaneous processes allowed per real user.

CLK_TCK
: Specifies the number of intervals per second in the machine clock.

NGROUPS_MAX
: Specifies the number of simultaneous group IDs per process.

OPEN_MAX
: Specifies the maximum number of open files at any time per process.

STREAM_MAX
: Specifies the number of streams that one process can have open at one time.

TZNAME_MAX
: Specifies the maximum number of bytes supported for the name of a time zone (not of the **TZ** variable).

_POSIX_ARG_MAX
: Specifies the minimum conforming value for ARG_MAX.

_POSIX_CHILD_MAX
: Specifies the minimum conforming value for CHILD_MAX.

_POSIX_JOB_CONTROL
: Specifies the POSIX job control supported.

_POSIX_LINK_MAX
: Specifies the minimum conforming value for LINK_MAX.

_POSIX_MAX_CANON
: Specifies the minimum conforming value for MAX_CANON.

_POSIX_MAX_INPUT
: Specifies the minimum conforming value for MAX_INPUT.

_POSIX_NAME_MAX
: Specifies the minimum conforming value for NAME_MAX.

_POSIX_NGROUPS_MAX
: Specifies the minimum conforming value for NGROUPS_MAX.

_POSIX_OPEN_MAX
: Specifies the minimum conforming value for OPEN_MAX.

_POSIX_PATH_MAX
: Specifies the minimum conforming value for PATH_MAX.

_POSIX_PIPE_BUF
: Specifies the minimum conforming value for PIPE_BUF.

_POSIX_SAVED_IDS
: Specifies that processes have saved set-user-ID and saved set-group-ID bits set.

_POSIX_SSIZE_MAX
: Specifies the value that can be stored in an object of type *ssize_t*.

_POSIX_STREAM_MAX
: Specifies the minimum conforming value for STREAM_MAX.

_POSIX_TZNAME_MAX
: Specifies the minimum conforming value for TZNAME_MAX.

_POSIX_VERSION
: Specifies the version of POSIX adhered to in this release.

You can use the first form of **getconf** to find the value of the POSIX.2 standard configuration variables:

BC_BASE_MAX
: Specifies the maximum *ibase* and *obase* values for the **bc** command.

BC_DIM_MAX
: Specifies the maximum number of elements permitted in a **bc** array.

BC_SCALE_MAX
: Specifies the maximum *scale* size allowed in **bc**.

BC_STRING_MAX
: Specifies the maximum number of characters in a string in **bc**.

COLL_WEIGHTS_MAX
: Specifies the maximum number of weights assignable to an entry of the **LC_COLLATE order** keyword.

EXPR_NEST_MAX
: Specifies the maximum number of expressions that you can nest inside parentheses in an expression evaluated by **expr**.

LINE_MAX
: Specifies the maximum number of bytes that a utility can accept as an input line (either from the standard input or a text file) when the utility takes text files as input. This number includes the trailing <newline>.

RE_DUP_MAX
: Specifies the maximum number of repeated occurrences of a regular expression when using the interval notation \{*m,n*\}.

  (See Appendix C.)

POSIX2_C_BIND
: Indicates if the system supports the C Language Bindings Option.

POSIX2_C_DEV
: Indicates if the system supports the C Language Development Utilities Option.

POSIX2_FORT_DEV
: Indicates if the system supports the FORTRAN Development Utilities Option.

POSIX2_FORT_RUN
: Indicates if the system supports the FORTRAN Runtime Utilities Option.

POSIX2_LOCALEDEF
: Indicates if the system supports the creation of locales.

POSIX2_SW_DEV
: Indicates if the system supports the Software Development Utilities Option.

POSIX2_CHAR_TERM
: Indicates if the system supports at least one terminal type capable of all operations necessary for the User Portability Utilities Option. This parameter name is correct only on if POSIX2_UPE is on.

POSIX2_UPE
: Indicates if the system supports the User Portability Utilities Option.

POSIX2_VERSION
: Specifies the version of POSIX.2 adhered to in this release.

POSIX2_BC_BASE_MAX
: Specifies the minimum conforming value for BC_BASE_MAX.

POSIX2_BC_DIM_MAX
: Specifies the minimum conforming value for BC_DIM_MAX.

POSIX2_BC_SCALE_MAX
: Specifies the minimum conforming value for BC_SCALE_MAX.

POSIX2_BC_STRING_MAX
: Specifies the minimum conforming value for BC_STRING_MAX.

POSIX2_COLL_WEIGHTS_MAX
: Specifies the minimum conforming value for EQUIV_CLASS_MAX.

POSIX2_EXPR_NEST_MAX
: Specifies the minimum conforming value for EXPR_NEST_MAX.

POSIX2_LINE_MAX
: Specifies the minimum conforming value for LINE_MAX.

POSIX2_RE_DUP_MAX
: Specifies the minimum conforming value for RE_DUP_MAX.

This implementation of **getconf** also supports the following non-POSIX-conforming name:

_CS_SHELL
: Specifies the default shell (command interpreter).

Example

This example uses **getconf** to find the minimum conforming value for **PATH_MAX**, which is contained in the variable **_POSIX_PATH_MAX**. If you issue

```
getconf _POSIX_PATH_MAX
```

getconf displays

255

Localization

getconf uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix F for more information.

Exit Values

0 The specified *parameter_name* was valid and **getconf** displayed its value successfully.

>0
An error occurred.

Portability

POSIX.2, X/Open Portability Guide.

_CS_SHELL is an extension of the POSIX standard. Some symbols are supported only on systems that support POSIX.2.

Related Information

bc, **expr**, **sh**

See Appendix C for more information about **regexp**.

---

getopts — Parse utility options

Format

getopts *opstring name* [*arg ...*]

Description

getopts obtains options and their arguments from a list of parameters that follows the standard POSIX.2 option syntax (that is, single letters preceded by a hyphen (–) and possibly followed by an argument value). Typically, shell scripts use **getopts** to parse arguments passed to them. When you specify arguments with the *arg* argument on the **getopts** command line, **getopts** parses those arguments instead of the script command-line arguments (see **set**).

Options

opstring
Gives all the option letters that the script recognizes. For example, if the script recognizes **–a**, **–f**, and **–s**, *opstring* is afs. If you want an option letter to be followed by an argument value or group of values, put a colon after the letter, as in a:fs. This indicates that getopts expects the **–a** option to have the form **–a** *value*. Normally one or more blanks separate *value* from the option letter;

however, **getopts** also handles values that follow the letter immediately, as in **–a***value*. *opstring* cannot contain a question mark (?) character.

name
Specifies the name of a shell variable. Each time you invoke **getopts**, it obtains the next option from the positional parameters and places the option letter in the shell variable *name*.

getopts places a question mark (?) in *name* if it finds an option that does not appear in *opstring*, or if an option *value* is missing.

arg ...
Each option on the script command line has a numeric *index*. The first option found has an index of 1, the second has an index of 2, and so on. When **getopts** obtains an option from the script command line, it stores the index of the script in the shell variable **OPTIND**.

When an option letter has a following argument (indicated with a **:** in *opstring*), **getopts** stores the argument as a string in the shell variable **OPTARG**. If an option doesn't take an argument, or if **getopts** expects an argument but doesn't find one, **getopts** unsets **OPTARG**.

When **getopts** reaches the end of the options, it exits with a status value of 1. It also sets *name* to the character ? and sets **OPTIND** to the index of the first argument after the options. **getopts** recognizes the end of the options by any of the following:

- Finding an argument that doesn't start with **–**
- Finding the special argument **––**, marking the end of options
- Encountering an error (for example, an unrecognized option letter)

OPTIND and **OPTARG** are local to the shell script. If you want to export them, you must do so explicitly. If the script invoking **getopts** sets **OPTIND** to 1, it can call **getopts** again with a new set of parameters, either the current positional parameters or new *arg* values.

By default, **getopts** issues an error message if it finds an unrecognized option or some other error. If you do not want such messages printed, specify a colon as the first character in *opstring*.

Example

Following is an example of using getopts in a shell script:

```
# Example illustrating use of getopts builtin. This
# shell script would implement the paste command,
# using getopts to process options, if the underlying
# functionality was embedded in hypothetical utilities
# hpaste and vpaste, which perform horizontal and
# vertical pasting respectively.
#
paste=vpaste    # default is vertical pasting
seplist=" " # default separator is tab

while getopts d:s o
do      case "$o" in
        d)      seplist="$OPTARG";;
        s)      paste=hpaste;;
        [?])  print >&2 "Usage: $0 [-s] [-d seplist] file ..."
                exit 1;;
        esac
done
shift $OPTIND-1

# perform actual paste command
$paste -d "$seplist" "$@"
```

Environment Variables

getopts uses the following environment variables:

OPTARG Stores the value of the option argument found by **getopts**.
OPTIND Contains the index of the next argument to be processed.

Localization

getopts uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix F for more information.

Usage Note

getopts is a built-in shell command.

Exit Values

0 **getopts** found a script command line with the form of an option. This happens whether or not it recognizes the option.

1 **getopts** reached the end of the options, or an error occurred.

2 Failure because of an incorrect command-line option

Portability

POSIX.2, X/Open Portability Guide.

On UNIX systems, **getopts** is built in both the KornShell and Bourne shell.

Related Information

sh

---

grep — Search a file for a specified pattern

Format

grep [**–bcEFilnqsvx**] [**–e** *pattern*] ... [**–f** *patternfile*] ... [*pattern*] [*file ...*]
egrep [**–bcilnqsvx**] [**–e** *pattern*] ... [**–f** *patternfile*] ... [*pattern*] [*file ...*]
fgrep [**–bcilnqsvx**] [**–e** *pattern*] ... [**–f** *patternfile*] ... [*pattern*] [*file ...*]

Description

fgrep searches files for one or more *pattern* arguments. It does not use regular expressions; instead, it directs string comparisons to find matching lines of text in the input.

egrep works in a similar way, but uses extended regular expression matching. (For information about regular expression matching, see Appendix C.) If you include special characters in patterns typed on the command line, escape them by enclosing them in apostrophes to prevent inadvertent misinterpretation by the shell or command interpreter. To match a character that is special to **egrep**, put a back-slash (\) in front of the character. It is usually simpler to use **fgrep** when you don't need special pattern matching.

grep is a combination of **fgrep** and **egrep**. If you do not specify either **–E** or **–F**, **grep** behaves like **egrep**, but matches basic regular expressions instead of extended ones. You can specify a pattern to search with either the **–e** or **–f** option. If you do not specify either option, **grep** (or **egrep** or **fgrep**) takes the first non-option argument as the pattern for which to search. If **grep** finds a line that matches a pattern, it displays the entire line. If you specify multiple input files, the name of the current file precedes each output line.

Options

grep accepts all the following options while **egrep** and **fgrep** accept all but the **–E** and **–F** options.

–b Precedes each matched line with its file block number.

–c Displays only a count of the number of matched lines and not the lines themselves.

–E Matches using extended regular expressions (causes **grep** to behave like **egrep**).

–e *pattern*
Specifies one or more patterns separated by newlines for which **grep** is to search.

You can indicate each pattern with a separate **–e** option character, or with newlines within pattern. For example, the following two commands are equivalent:

```
grep –e pattern_one –epattern_two file
grep –e 'pattern_one
pattern_two' file
```

–F Matches using fixed strings (causes **grep** to behave like **fgrep**).

–f *patternfile*
Reads one or more patterns from *patternfile*. Patterns in *patternfile* are separated by newlines.

–i Ignores the case of the strings being matched.

–l Lists only the filenames that contain the matching lines.

–n Precedes each matched line with its fileline number.

–q Suppresses output and returns the appropriate return code.

–s Suppresses the display of any error messages for nonexistent or unreadable files.

–v Complements the sense of the match—that is, displays all lines *not* matching a pattern.

–x Requires a string to match an entire line.

Example

To display every line mentioning an astrological element:

```
egrep "earth|air|fire|water" astro.log
```

Localization

grep uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 At least one match for *pattern* was found.

1 No matches for *pattern* were found.

2 Failure due to any of the following:

- The **–e** option was missing *pattern*.
- The **–f** option was missing *patternfile*.
- Out of memory for input or to hold a pattern.
- *patternfile* could not be opened.
- Incorrect regular expression.
- Incorrect command-line option.

- The command line had too few arguments.
- The input file could not be opened.

If the program fails to open one input file, it tries to go on to look at any remaining input files, but it returns 2 even if it succeeds in finding matches in other input files.

Messages

Possible error messages include:

input lines truncated—result questionable
One or more input lines were longer than **grep** could handle; the line has been truncated or split into two lines. Shorten the line or lines, if possible. This message does not affect the exit status.

out of space for pattern *string*
grep did not have enough memory available to store the code needed to work with the given pattern (regular expression). The usual cause is that the pattern is very complex. Make the pattern simpler, or try to release memory so that **grep** has more space to work with.

Limits

The longest input record (line) is restricted by the system variable LINE_MAX. It is always at least 2048 bytes. **fgrep** may be able to handle lines longer than LINE_MAX. Longer lines are treated as two or more records.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Only the **grep** command is part of the POSIX and X/Open standards. The **egrep** and **fgrep** commands are extensions. The **–b** option is an extension of the POSIX standard.

Related Information

ed, **find**

See Appendix C for more information about **regexp**.

---

hash — Create a tracked alias

Format

hash [*name ...*]
hash -r

Description

hash creates one or more tracked aliases. Each *name* on the command line becomes an alias that is resolved to its full pathname; thus the shell avoids searching the **PATH** directories for the command whenever you issue it. A tracked alias is assigned its full pathname the first time that the alias is used. It is reassigned a pathname the first time that it is used after the variable **PATH** is changed or the shell command **cd** is used.

hash is a built-in alias defined with

```
alias hash='alias -t'
```

If you specify **hash** without any arguments on the command line, **hash** displays the current list of tracked aliases.

Option

–r Removes all current tracked aliases.

Usage Note

hash is a built-in shell command.

Localization

hash uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion
1 Failure because of an incorrect command-line option

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Information

alias, **sh**

---

head — Display the first part of a file

Format

head [**–bcklmn** *num*] [*file ...*]
head [*–num*] [*file ...*]

Description

By default, **head** displays the first 10 lines of each file given on the command line. If you do not specify *file*, **head** reads standard input (**stdin**).

Options

–b *num* Displays the first *num* blocks (a block is 512 bytes) of each file.
–c *num* Displays the first *num* bytes of each file.
–k *num* Displays the first *num* kilobytes (1024 bytes) of each file.
–l *num* Displays the first *num* lines of each file.
–m *num* Displays the first *num* megabytes of each file.
–n *num* Displays the first *num* lines of each file.
–*num* Displays the first *num* lines of each file.

Localization

head uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion
1 Failure due to any of the following:
- Inability to open an input file
- Read error on **stdin**
- Write error on **stdout**

2 Failure due to any of the following:
- Unknown command-line option
- Missing or incorrect *num* in an **–n** option

Messages

Possible error messages include:

Badly formed line or character count *num*
The value *num*, following a **–b**, **–c**, **–k**, **–l**, **–m**, or **–n** option, was not a valid number.

Portability

POSIX.2, X/Open Portability Guide.

This program originated with Berkeley Software Distribution (BSD) and is a frequent add-on to UNIX systems.

The POSIX.2 standard included only the **–n** *num* and **–***num* options, though it considers the latter obsolete.

Related Information

cat, **sed**, **tail**

history — Display a command history list

Format

history [*first*[*last*]]

tcsh shell:

history [**-hTr**] [*n*]

history -Sl-Ll-M [*filename*]

history -c

Description

history is an alias for **fc –l**. Like **fc –l**, **history** displays the list of commands that have been input to an interactive shell. This command does not edit or reenter the commands. If you omit *last*, **history** displays all commands from the one indicated by *first* through to the previous command entered. If you omit both *first* and *last* with this command, the default command range is the 16 most recently entered commands.

See "fc — Process a command history list" on page 250 for more information.

history in the tcsh shell

In the tcsh shell, **history**, used alone, prints the history event list. If *n* is given only the *n* most recent events are printed or saved.

Note: See "tcsh — Invoke a C shell" on page 556 for descriptions of the tcsh shell variables and commands indicated below.

The tcsh shell **history** built-in command uses the following options:

- With **-h**, the history list is printed without leading numbers.
- With **-T**, timestamps are printed also in comment form. (This can be used to produce files suitable for loading with **history -L** or **source -h**.)
- With **-r**, the order of printing is most recent first rather than oldest first.
- With **-S**, **history** saves the history list to *filename*. If the first word of the **savehist** shell variable is set to a number, at most that many lines are saved. If the second word of **savehist** is set to merge, the history list is merged with the existing history file instead of replacing it (if there is one) and sorted by time stamp. Merging is intended for an environment like the X Window System with several shells in simultaneous use. Currently it only succeeds when the shells quit one after another.
- With **-L**, the shell appends *filename*, which is presumably a history list saved by the **-S** option or the **savehist** mechanism, to the history list. **-M** is like **-L**, but the contents of *filename* are merged into the history list and sorted by timestamp. In either case, **histfile** is used if filename is not given and **~/.history** is used if histfile is unset. **history -L** is exactly like **source -h** except that it does not require a filename.
- With **-c**, clears the history list.

tcsh login shells do the equivalent of **history -L** on startup and, if **savehist** is set, **history -S** before exiting. Because only **~/.tcshrc** is normally sourced before **~/.history**, **histfile** should be set in **~/.tcshrc** rather than **~/.login**. If **histlit** is set, the first form (**history** [**-hTr**] [*n*]) and second form (**history -Sl-Ll-M** [*filename*]) print and save the literal (unexpanded) form of the history list.

Related Information

fc, **sh**, **tcsh**

---

iconv — Convert characters from one code set to another

Format

iconv [–sc] –f *oldset* **–t** *newset [file ...]*
iconv –l[–v]

Description

iconv converts characters in *file* (or from **stdin** if no file is specified) from one code page set to another. The converted text is written to **stdout**. See *OS/390 C/C++ Programming Guide* for more information about the code sets supported for this command.

If the input contains a character that is not valid in the source code set, **iconv** replaces it with the byte `0xff` and continues, unless the **–c** option is specified.

If the input contains a character that is not valid in the destination code set, behavior depends on the system's **iconv()** function. See *OS/390 C/C++ Run-Time Library Reference* for more information about the character used for converting incorrect characters.

Also, *OS/390 C/C++ Programming Guide* has a list of code pages supported by the OS/390 shell.

You can use **iconv** to convert singlebyte data or doublebyte data.

Options

–c Characters containing conversion errors are not written to the output. By default, characters not in the source character set are converted to the value `0xff` and written to the output.

–f *oldset*
oldset can be either the code set name or a pathname to a file containing an external code set.

–l Lists code sets in the internal table. This option is not supported.

–s Suppresses all error messages about faulty encodings.

–t *newset*
Specifies the destination code set for the output. *newset* can be either the code set name or a pathname to a file containing an external code set.

–v Specifies verbose output.

Localization

iconv uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Examples

1. To convert the file **words.txt** from the IBM-1047 standard code set to the ISO 8859-1 standard code set and store it in **converted**:

   ```
   iconv –f IBM-1047 –t ISO8859-1 words.txt > converted
   ```

   Also, for the exact conversion table names, refer to *OS/390 C/C++ Programming Guide*.

2. To convert the file **mbcsdata**, which is in code page IBM-932 (doublebyte ASCII), to code page IBM-939 and put the output in a file called **dbcsdata**:

   ```
   iconv –f IBM-932 –t IBM-939 mbcsdata > dbcsdata
   ```

Exit Values

0 Successful completion

1 Failure due to any of the following:

- Insufficient memory
- Inability to open the input file
- Incorrect or unknown option

2 Input contained a character sequence that is not permitted in the source code set

Portability

X/Open Portability Guide.

–v is an extension to the POSIX.2 standard. The **–c**, **–l**, and **–s** options are extensions to the XPG standard.

---

id — Return the user identity

Format

id [*user*]
id –G [**–n**] [*user*]
id –g [**–nr**] [*user*]
id –u [**–nr**] [*user*]

Description

id displays the user name and group affiliations of the user who issued the command. Specifying a *user* argument on the command line displays the same information for the given user instead of the person invoking **id**. In this case, you require appropriate permissions.

The output has the format:

`uid=`*`runum(username)`* `gid=`*`rgnum(groupname)`*

where *runum* is the user's real user ID (UID) number, *username* is the user's real user name, *rgnum* is the user's real group ID (GID) number, and *groupname* is the user's real group name.

A user's real and effective IDs may differ. In this case, there may be separate entries for effective user ID (UID) with the format:

`euid=`*`eunum(euname)`*

where *eunum* is the effective user ID number and *euname* is the effective user name. An entry for effective group ID has the format:

`egid=`*`egnum(egname)`*

where *egnum* is the effective group ID number and *egname* is the effective group name.

If a user is a member of other supplemental groups, these are listed at the end of the output, with this format:

`groups=`*`gnum(groupname)`*

where *gnum*is the user's supplemental group ID number and *groupname* is the user's supplemental group name.

Options

–G Displays all different group IDs (effective, real, and supplementary) as numbers separated by spaces.

–g Displays only the effective group ID number.

–n With **–G**, **–g**, or **–u**, displays the name rather than the number.

–r With **–g** or **–u**, displays the real ID rather than the effective one.

–u Displays only the effective user ID number.

Localization

id uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_NUMERIC**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion
1 You specified an incorrect user with the **–u** option
2 Failure due to an incorrect command-line argument

Portability

POSIX.2, X/Open Portability Guide, UNIX System V.

Related Information

logname

---

inetd daemon — Provide Internet Service Management

Format

inetd [**–d**] [*configuration file*]

Description

The **inetd** daemon provides service management for a network. For example, it starts the rlogind program whenever there is a remote login request from a workstation.

The rlogind program is the server for the remote login command **rlogin** commonly found on UNIX systems. It validates the remote login request and verifies the password of the target user. It starts an OS/390 shell for the user and handles translation between ASCII and EBCDIC code pages as data flows between the workstation and the shell.

When **inetd** is running and receives a request for a connection, it processes that request for the program associated for that socket. For example, if a user tries to log in from a remote system into the OS/390 shell while **inetd** is running, **inetd** processes the request for connection and then issues a **fork()** and **execl()** to the rlogin program to process the rlogin request. It then goes back to monitoring for further requests for those applications that can be found as defined in the **/etc/inetd.conf** file.

Options

–d Specifies that the **inetd** daemon be started in debug mode. All debug messages are written to **stderr**.

configuration file
Specifies that the **inetd** daemon be started with a configuration file other than the default **/etc/inetd.conf** file.

Signals

inetd recognizes the following signals:

SIGTERM Terminates **inetd** in an ordinary fashion and deletes **/etc/inetd.pid**. You can restart **inetd**, if you want.

SIGINT Same as SIGTERM.

SIGHUP Rereads the **inetd** configuration file. This can be used to start new services, or to restart services with a different port.

Other signals that normally end a process (such as SIGQUIT or SIGKILL) should not normally be sent to **inetd** because the program will not have a chance to remove **/etc/init.pid**.

Usage Notes

The configuration file is field-sensitive, but not column-sensitive. Fields should be arranged in the order shown in Figure 6. Continuation lines for an entry must begin with a space or tab. Each entry must contain all fields. The **inetd** daemon uses the configuration file entry to properly set up the environment expected by the server. Specifying an incorrect value for one or more of the parameters is likely to cause the server to fail.

Figure 6. Fields in the Configuration File (inetd daemon)	
service_name	A well-known service name such as **login** or **shell**. The name and protocol specified must match one of the server names defined in the MVS data set `/etc/services`. For more information on `/etc/services`, see *OS/390 SecureWay Communications Server: IP Configuration* and *OS/390 SecureWay Communications Server: IP Migration* .
socket_type	Stream or dgram
protocol	**tcp** or **udp**. The protocol is used to further qualify the service name. Both the service name and the protocol should match an entry in the data set, `/etc/services`, which is part of the user's TCP/IP configuration. For more information on `/etc/services`, see *OS/390 SecureWay Communications Server: IP Configuration* and *OS/390 SecureWay Communications Server: IP Migration* .
wait_flag [.max]	**Wait** or **nowait**. **Wait** indicates the daemon is single-threaded and another request will not be serviced until the first one completes. If **nowait** is specified, the **inet** daemon issues an accept when a connect request is received on a stream socket. If **wait** is specified, the **inet** daemon does not issue the accept. It is the responsibility of the server to issue the accept if this is a stream socket. **max** is the maximum number of users allowed to request service in a 60 second interval. Default is 40. If exceeded, the service's port is shut down.

login_name	User ID and group that the forked daemon is to execute under. **inetd** can run a program with a UID that is not 0. However, if the program that **inetd** runs needs to change the identity of the process to that of the user, then the login_name must have been defined to RACF via ADDUSER as a superuser with a UID of 0 (UID 0) and the login_name must have been defined to RACF. This will allow **inetd** to use special functions like **setgid()** and **setuid()**.
	
	If the program that will be invoked by **inetd** requires the use of special functions like **setuid()** and **seteuid()**, then it must be permitted to the BPX.DAEMON class as in the following example for login, which is a typical ADDUSER command.
	
	`ADDUSER rlogind omvs(uid(0) home(/)`
	
	A typical permit command is:
	
	`permit bpx.daemon class(facility)`
	`  id(rlogind) access(read)`
	
	How you set up security for daemons is the final determining factor. For more information, see *OS/390 UNIX System Services Planning*.

server_program	Full pathname of the service. For example: `/usr/sbin/rlogind` is the full pathname for the **rlogind** command.
server_arguments	Maximum of 20 arguments. The first argument is the server name.

Related Information

The **inetd** daemon creates a temporary file, **/etc/inetd.pid,** that contains the PID of the currently executing **inetd** daemon. This PID value is used to identify syslog records that originated from the **inetd** daemon process, and also to provide the PID value for commands such as **kill** that require you to specify a PID, and to provide a lock to prevent more than one **inetd** from being active at one time.

For more information on setting up the inetd configuration file and configuring daemons in general, see *OS/390 UNIX System Services Planning* or *OS/390 SecureWay Communications Server: IP Configuration* .

---

infocmp — Compare or print the terminal description

Format

infocmp [**–ducn**] [**–ILC**] [**–1Vv**] [**–s dlilllc**] [**–A** *directory*] [**–B** *directory*] [*term_names...*]

Description

infocmp compares terminfo database entries, or prints a terminfo database entry. Output is written to standard output (**stdout**).

The Curses application uses the terminfo database, which contains a list of terminal descriptions. This enables you to manipulate a terminal's display regardless of the terminal type. To create the terminfo database, use **tic**. For information on defining the terminfo database, see *OS/390 UNIX System Services Planning*.

For more information about curses, see *OS/390 C Curses*.

Options

d Prints the two terminal definitions showing the differences between the capabilities.

u Prints the differences between the two terminal definitions.

c Prints entries that are common to the two terminfo databases.

n Does not print entries in either terminfo database.

I Prints the current terminal description using capname. (*capname* is the short name for a capability specified in the terminfo source file.)

C Prints the current terminal description using termcap.

L Prints the current terminal description using variables (names that the curses functions can use when working with the terminfo database)

1 Single-column output.

V Prints the program version.

v Prints debugging information (verbose) to **stderr.**

s Changes sort order of the fields printed.

- **d** Sorts by database
- **i** Sorts by terminfo
- **c** Sorts by termcap

I Sorts by the variables (names that the curses function can use when working with the terminfo database)

A First terminfo database.

B Other terminfo database.

term_names
Names of entries to be processed.

Usage Notes

When displaying terminal database information for entries that are to be processed, **infocmp** operates as follows:

1. If you omit *term_names*, **infocmp** locates the terminal database information specified by the **TERM** environment variable and displays that as the entry's terminal database information.
2. If you specify a single *term_name*, **infocmp** displays terminal database information for that named entry.
3. If you specify more than one *term_name*, **infocmp** displays the results of a terminal database comparison between all of the specified *term_names*.

Examples

1. To print out the current terminal description using capname, issue:

   ```
   infocmp
   ```

   You will see:

   ```
   infocmp ibm3101
   #    Reconstructed via infocmp from file:/usr/share/lib/terminfo/i
   ibm3101|IBM 3101-10:\

   ibm3101|IBM 3101-10,
      am,
      cols#80, lines#24,
      bel=¬?, clear=¬.\322, cr=\r, cub1=\b, cud1=\n,
      cuf1=¬.\303,
   ```

2. To print out the current terminal description using the curses capability names, issue:

   ```
   infocmp –L
   ```

   You will get:

```
infocmp –L ibm3101
#    Reconstructed via infocmp from file:/usr/share/lib/terminfo/i
ibm3101|IBM 3101-10:\

Terminal type ibm3101
  ibm3101|IBM 3101-10
flags
  auto_right_margin,

numbers
  columns = 80, lines = 24,

strings
  bell = '¬?', carriage_return = '\r', clear_all_tabs = '¬.\310',
  clear_screen = '¬.\322', clr_eol = '¬.\311', clr_eos = '¬.\321',
```

3. To print out the current terminal description using capname, issue:

   ```
   infocmp –I
   ```

   You will get:

   ```
   infocmp –I ibm3101
   #    Reconstructed via infocmp from file:/usr/share/lib/terminfo/i
   ibm3101|IBM 3101-10:\

   ibm3101|IBM 3101-10,
      am,
      cols#80, lines#24,
      bel=¬?, clear=¬.\322, cr=\r, cub1=\b, cud1=\n,
      cuf1=¬.\303,
   ```

4. To print out the current terminal description using termcap, issue:

   ```
   infocmp –C
   ```

   You will get:

   ```
   infocmp –C ibm3101
   #    Reconstructed via infocmp from file:/usr/share/lib/terminfo/i
   ibm3101|IBM 3101-10:\
      :am:bs:\
      :.co#80:li#24:kn#2:\
      :.cd=\EJ:.ce=\EI:.cl=\EK:\
      :.c:.cm=\EY%p1%' '%+%c%p2%'
   '%+%c:.ct=\EH:.ho=\EH:\
      :.nd=\EC:.st=\E0:.up=\EA:
   ```

5. To print entries in single-column format, issue:

   ```
   infocmp –1
   ```

   You will get:

```
infocmp –C –1 ibm3101
#    Reconstructed via infocmp from file:/usr/share/lib/terminfo/i
ibm3101|IBM 3101-10:\
   :am:\
   :bs:\
   :co#80:\
   :li#24:\
   :kn#2:\
   :.cd=\EJ:\
   :.ce=\EI:\
   :.cl=\EK:\
```

6. To print the two terminal definitions showing the difference between the capabilities (F indicates False, entry not present; T indicates True, entry present):

   ```
   infocmp –d ibm3101 hft-c
   ```

   You will get:

   ```
   #    Reconstructed via infocmp from file:/usr/share/lib/terminfo/i
   ibm3101|IBM 3101-10:\
           comparing ibm3101 to hft-c.
              comparing booleans.
              bw:F:T.
              msgr:F:T.
              xon:F:T.
                comparing numbers.
              it:-1:8.
              lines:24:25.
                comparing strings.
              batt1:'NULL','\206\361'.
              batt2:'NULL','\206\361\224\204'.
   ```

   To print the capabilities that are different between the two terminal definitions. The values for the first terminal definitions are shown.

   ```
   infocmp –u ibm3101 hft-c
   ```

   You will get:

   ```
   #    Reconstructed via infocmp from file:/usr/share/lib/terminfo/i
   ibm3101|IBM 3101-10:\
           ibm3101|IBM 3101-10,
               bw@, msgr@, xon@,
               it@, lines#24,
               batt1@, batt2@, blink@, bold@, box1@, box2@,
               clear=¬.\322, colb0@, colb1@, colb2@, colb3@, colb4@,
               colb5@, colb6@, colb7@, colf0@, colf1@, colf2@,
               colf3@, colf4@, colf5@, colf6@, colf7@, cub@, cud@,
               cuf@, cuf1=¬.\303,
   ```

7. To print the capabilities that are the same in both terminal definitions, issue:

   ```
   infocmp –c ibm3101 hft-c
   ```

   You will get:

```
#    Reconstructed via infocmp from file:/usr/share/lib/terminfo/i
ibm3101|IBM 3101-10:\
     comparing ibm3101 to hft-c.
        comparing booleans.
           am= T.
        comparing numbers.
           cols= 80.
        comparing strings.
           bel= '¬?'.
           cr= '\r'.
           cub1= '\b'.
           cud1= '\n'.
```

8. To print the capabilities that are not found in either terminal definition, issue:

   ```
   infocmp –u ibm3101 hft-c
   ```

   You will get:

   ```
   #    Reconstructed via infocmp from file:/usr/share/lib/terminfo/i
   ibm3101|IBM 3101-10:\
      comparing ibm3101 to hft-c.
         comparing booleans.
            !bce.
            !ccc.
            !chts.
            !cpix.
            !crxm.
            !da.
            !daisy.
   ```

Environment Variables

infocmp uses the following environment variable:

TERMINFO Contains the pathname of the terminfo database.

TERM Contains the name of your terminal, that is, the current terminal definition.

Related Information

captoinfo, **tic**

---

integer — Mark each variable with an integer value

Format

integer [*number*]

Description

integer is an alias for **typeset –i**. Like **typeset –i**, **integer** marks each variable as having an integer value, thus making arithmetic faster. If *number* is given and is nonzero, the output base of each variable is *number*. The default is decimal.

See “typeset — Assign attributes and values to variables” on page 646 for more information.

Related Information

typeset, **sh**

---

ipcrm — Remove message queues, semaphore sets, or shared memory IDs

Format

ipcrm [**–m SharedMemoryID**] [**–M SharedMemoryKey**] [**–q QMessageID**] [**–Q MessageKey**] [**–s SemaphoreID**] [**–S SemaphoreKey**]

Description

ipcrm removes one or more message queues, semaphores set, or shared memory identifiers.

Options

–m SharedMemoryID
Removes the shared memory identifier **SharedMemoryID.** The shared memory segment and data structure associated with **SharedMemoryID** are also removed after the last detach operation.

–M SharedMemoryKey
Removes the shared memory identifier, created with the key **SharedMemoryKey.** The shared memory segment and data structure associated with it are also removed after the last detach operation.

–q MessageID
Removes the message queue identifier **MessageID** and the message queue and data structure associated with it.

–Q MessageKey
Removes the message queue identifier, created with the key **MessageKey,** and the message queue and data structure associated with it.

–s SemaphoreID
Removes the semaphore identifier **SemaphoreID** and the set of semaphores and data structure associated with it.

–S SemaphoreKey
Removes the semaphore identifier, created with the key **SemaphoreKey**, and the set of semaphores and data structure associated with it.

The msgctl, shmctl, and semctl subroutines provide details of the remove operations. You can use the **ipcs** command to find the identifiers and keys.

Examples

1. To remove the shared memory segment associated with **SharedMemoryID 18602**, enter:

   ```
   ipcrm –m 18602
   ```

2. To remove the message queue that was created with a key of 0xC1C2C3C3, enter:

   ```
   ipcrm –Q 0xC1C2C3C4
   ```

Exit Values

0 Successful completion
1 Incorrect command-line option

Related Information

ipcs

---

ipcs — Report status of the interprocess communication facility

Format

ipcs [**–m**] [**–q**] [**–s**] [**–a** | **–b c o p t**] [**–w**] [**–x**] [**–y**]

Description

ipcs writes to the standard output information about active interprocess communication facilities. If you do not specify any flags, **ipcs** writes information in a short form about currently active message queues, shared memory segments, and semaphores.

The column headings and the meaning of the columns in an **ipcs** command listing are listed in Figure 7. The letters in parentheses indicate the command flags that cause the corresponding heading to appear. *(all)* means that the heading is always displayed. These flags determine what information is provided for each facility. They do not determine which facilities are listed.

Figure 7 (Page 1 of 5). Explanation of the ipcs Command Listing

Column Heading	Meaning of the Column
T (all except y)	The type of facility: **q** Message queue **m** Shared memory segment **s** Semaphore
ID (all except x,w,y)	The identifier for the facility entry
KEY (all except y)	The key used as a parameter to the **msgget** subroutine, the **semget** subroutine, or the **shmget** subroutine to make the facility entry. (The key of a shared memory segment is changed to IPC_PRIVATE when the segment is removed until all processes attached to the segment detach it.)

Figure 7 (Page 2 of 5). Explanation of the ipcs Command Listing

Column Heading	Meaning of the Column
MODE (all except x,w,y)	The facility access modes and flags. The mode consists of 11 characters that are interpreted as follows: The first two characters can be the following: **R** If a process is waiting in a **msgrcv()** system call. **S** If a process is waiting in a **msgsnd()** system call. **D** If the associated shared memory segment has been removed. It disappears when the last process attached to the segment detaches it. The next nine characters are interpreted as three sets of three characters each. The first set refers to the owner's permissions; the next to permissions of others in the user group of the facility entry; and the last to all others. Within each set, the first character indicates permission to read, the second character indicates permission to write or alter the facility entry, and the last character is currently unused. The permissions are indicated as follows: **r** If read permission is granted **w** If write permission is granted **a** If alter permission is granted **–** If the indicated permission is not granted
OWNER (all)	The login name or user ID of the owner of the facility entry.
GROUP (all)	The name or group ID of the group that owns the facility entry.
CREATOR (a,c)	The login name or user ID of the creator of the facility entry.
CGROUP (a,c)	The group name or group ID of the creator of the facility entry.
CBYTES (a,x,o)	The number of bytes in messages currently outstanding on the associated message queue.
INFO (x)	Provides additional extended state information. Under this field will be returned one or more of the following codes (codes are not mutually exclusive unless noted): For shared memory output: **M** megaroo For semaphore output: **P** PLO in use. Mutually exclusive with **L**. **L** Latch in use. Mutually exclusive with **P**. **B** Binary semaphore. For message queue output: **P** PLO in use. Mutually exclusive with **L**. **L** Latch in use. Mutually exclusive with **P**. **R** RCV type PID. **S** Send type PID. **1** PLO1 flag on—IPC_PLO1 set on msgget() **2** PLO2 flag on—IPC_PLO2 set on msgget()
QNUM (a,o)	The number of messages currently outstanding on the associated message queue.

Figure 7 (Page 3 of 5). Explanation of the ipcs Command Listing

Column Heading	Meaning of the Column
QBYTES (a,b)	The maximum number of bytes allowed in messages outstanding on the associated message queue.
LSPID (p)	The ID of the last process that sent a message to the associated queue.
LRPID (p)	The ID of the last process that received a message from the associated queue.
STIME (a,t)	The time when the last message was sent to the associated queue.
RTIME (a,t)	The time when the last message was received from the associated queue.
CTIME (a,t)	The time when the associated entry was created or changed.
NATTCH (a,o)	The number of processes attached to the associated shared memory segment.
SEGSZPG (a,b)	The size in PAGES of the associated shared memory segment.
CPID (p)	The process ID of the creator of the shared memory entry.
LPID (p)	The process ID of the last process to attach or detach the shared memory segment.
ATIME (a,t)	The time when the last attach was completed to the associated shared memory segment.
DTIME (a,t)	The time the last detach was completed on the associated shared memory segment.
NSEMS (a,b)	The number of semaphores in the set associated with the semaphore entry.
OTIME (a,t)	The time the last semaphore operation was completed on the set associated with the semaphore entry.
RCVWAIT (x)	A count of **msgrcv()** waiters.
SNDWAIT (x)	A count of **msgsnd()** waiters.
MSGQPID (w)	For the message Q report, up to 10 lines of data will be shown under this heading.
MSGQTYPE (w)	For the message Q report, up to 10 lines of data will be shown under this heading.
RCVPID (w)	The process ID of a **msgrcv()** waiter. A maximum of 10 process IDs can be written.
RCVTYP (w)	The message type of a **msgrcv()** waiter associated with RCVPID. A maximum of 10 message type will be written. If the caller does not have read access, this field is not displayed.
SNDPID (w)	The process ID of a **msgsnd()** waiter. A maximum of 10 process IDs can be written
SNDLEN (w)	The message send length of a **msgsnd()** waiter associated with SNDPID. A maximum of 10 message send lengths can be written.
TERMA (x)	The number of times sem_val was changed during termination for semaphore adjustments.
CNADJ (x)	The current number of processes with semaphore adjustments.
SNCNT (x)	The number of waiters waiting for a sem_val greater than zero.

Figure 7 (Page 4 of 5). Explanation of the ipcs Command Listing

Column Heading	Meaning of the Column
SZCNT (x)	The number of waiters waiting for a sem_val equal to zero.
WTRPID (w)	The process IDs of a semop waiter. A maximum of 10 semop waiters are written.
WTRNM (w)	The semaphore number associated with WTRPID. A maximum of 10 semaphore numbers are written.
WTROP (w)	The semaphore operation value associated with WTRNM and WTRPID. A maximum of 10 semaphore operation values are written.
AJPID (w)	The process ID of a process with semaphore adjustments. A maximum of 10 process IDs are written.
AJNUM (w)	The semaphore number of the semaphore adjustment associated with AJPID. A maximum of 10 semaphore numbers are written.
AJPID (w)	The process ID of a process with semaphore adjustments. A maximum of 10 process IDs are written.
AJNUM (w)	The semaphore number of the semaphore adjustments associated with AJPID. A maximum of 10 semaphore numbers are written.
AJVAL (w)	The semaphore adjustment value associated with AJNUM and AJPID. A maximum of 10 semaphore adjustment values are written.
ATPID (x)	The process ID of a process that is attached to this shared memory segment. A maximum of 10 process IDs are written.
ATADDR (x)	The shared memory address where the process ATPID is attached to this segment. A maximum of 10 addresses are written.
MNIDS (y)	The system limit for maximum number of message queues, semaphores, or shared memory IDs.
HWIDS (y)	The most message queues, semaphores, or shared memory IDs created.
CIDSA (y)	The current number of message queues, semaphores, or shared memory IDs available.
CPRIV (y)	The current number of message queues, semaphores, or shared memory IDs created with IPC_PRIVATE
CKEY (y)	The current number of message queues, semaphores, or shared memory IDs created with IPC_PRIVATE.
GETEX (y)	The number of times **msgget**, **semget**, or **shmget** exceeded the maximum number of IDs MNID.
MAXQB (y)	The system limit for maximum number of bytes on a message queue.
QMNUM (y)	The system limit for maximum number messages on a message queue.
ENOMEM (y)	The number of times **msgsnd()** calls returned ENOMEM.
MNSEMS (y)	The system limit for maximum number of semaphores per set.
MNOPS (y)	The system limit for maximum number of operations per semop.

Figure 7 (Page 5 of 5). Explanation of the ipcs Command Listing	
Column Heading	**Meaning of the Column**
CSBYTES (y)	The current number of bytes used by the system for semaphores.
TPAGES (y)	The system limit for number of system-wide shared memory pages
SPAGES (y)	The system limit for number of pages per shared memory segment.
SEGPR (y)	The system limit for number of segments per process.
CPAGES (y)	The current number of system-wide shared memory pages
MAXSEG (y)	The largest number of shared memory pages allocated to a single shared memory segment.

Options

–a Uses the **–b**, **–c**, **–o**, and **–t** flags.

–b Writes the maximum number of bytes in messages on queue for message queues, the size of segments for shared memory, and the number of semaphores in each semaphores set.

–c Writes the login name and group name of the user that made the facility.

–m Writes information about active shared memory segments.

–o Writes the following usage information:

- Number of messages on queue
- Total number of bytes in messages in queue for message queues
- Number of processes attached to shared memory segments

–p Writes the following:

- Process number of the last process to receive a message on message queues
- Process number of the creating process
- Process number of last process to attach or detach on shared memory segments

–q Writes information about active message queues.

–s Writes information about active semaphore set.

–t Writes the following:

- Time of the last control operation that changed the access permissions for all facilities
- Time of the last **msgsnd()** and **msgrcv()** on message queues
- Time of the last shmat and shmdt on shared memory
- Time of the last semop on semaphore sets

–x Writes extended status in these fields:

ATADDR	INFO	RCVWAIT	SZCNT
ATPID	KEY	SEGSZPG	T
CNADJ	OWNER	SNCNT	TERMA
GROUP	QCBYTES	SNDWAIT	

–w Writes message queue wait status and semaphore adjustment status in these fields:

AJNUM	KEY	RCVPID	T
AJPID	MSGQPID	RCVTYP	WTRNM
AJVAL	MSGQTYP	SNDLEN	WTROP
GROUP	OWNER	SNDPID	WTRPID

–y Writes summary and system limit status in these fields:

CIDSA	ENOMEM	MAXSEG	SPAGES
CKEY	GETEX	MNIDS	TPAGES
CPAGES	HWIDS	MNOPS	
CPRIV	QMNUM	MNSEMS	
CSBYTES	MAXQB	SEGPR	

Example

Following is a sample output from entering **ipcs** without flags:

```
IPC status as of Wed Apr  6 14:56:22 EDT 1994
Message Queues:
T         ID     KEY        MODE       OWNER    GROUP
q 1234567890 0x4107001c -Rrw-rw----     root   printq
Shared Memory:
T         ID     KEY        MODE       OWNER    GROUP
m          0 0x0d07021e --rw-------     root   system
m          1 0x0d08c984 --rw-rw-rw-     root   system
Semaphores:
T         ID     KEY        MODE       OWNER    GROUP
s       4096 0x0108c86e --ra-------     root   system
s          1 0x6208c8ef --ra-r--r--     root   system
s          2 0x4d0b00a6 --ra-ra----     root   system
s      24579 0x00bc614e --ra-ra-ra-     xlin   vendor
s     176132 0x00000058 --ra-ra-ra-     xlin   vendor
```

Exit Values

0 Successful completion
1 Failure due to incorrect command-line option

Related Information

ipcrm

---

jobs — Return the status of jobs in the current session

Format

jobs [**–l**|**–p**] [*job-identifier...*]

tcsh shell: **jobs** [**-l**]

Description

jobs produces a list of the processes in the current session. Each such process is numbered for easy identification by **fg** or **kill**, and is described by a line of information:

[job-identifier] default state shell_command

job-identifier
: Is a decimal number that identifies the process for such commands as **fg** and **kill** (preface *job-identifier* with **%** when used with these commands).

default
: Identifies the process that would be the default for the **fg** and **bg** commands (that is, the most recently suspended process). If *default* is a **+**, this process is the default job. If *default* is a **–**, this job becomes the default when the current default job exits. There is at most one **+** job and one **–** job.

state Shows a job as:

`Running`	If it is not suspended and has not exited
`Done`	If it exited successfully
`Done(exit status)`	If it exited with a nonzero exit status
`Stopped` **(***signal***)**	If it is suspended; *signal* is the signal that suspended the job

shell_command
: Is the associated shell command that created the process.

jobs in the tcsh shell

In the tcsh shell, **jobs** lists the active jobs. With **-l**, lists process IDs in addition to the normal information. See “tcsh — Invoke a C shell” on page 556.

Options

–l Displays the process group ID of a job (before *state*).
–p Displays the process IDs of all processes.

The **–l** and **–p** options are mutually exclusive.

Localization

jobs uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Usage Note

jobs is a built-in shell command.

Exit Values

0 Successful completion
2 Failure due to an incorrect command-line argument

Portability

POSIX.2 User Portability Extension.

Related Information

bg, **fg**, **kill**, **ps**, **wait**, **tcsh**

---

join — Join two sorted textual relational databases

Format

join [**–a** *n*] [**–e** *s*] [**–o** *list*] [**–t** *c*] [**–v** *n*] [**–1** *n*] [**–2** *n*] *file1 file2*
join [**–a** *n*] [**–e** *s*] [**–j**[*n*] *m*] [**–o** *list*] [**–t** *c*] *file1 file2*

Description

join joins two databases. It assumes that both *file1* and *file2* contain textual databases in which each input line is a record and that the input records are sorted in ascending order on a particular join key field (by default the first field in each file). If you specify **–** in place of *file1* or *file2*, **join** uses the standard input (**stdin**) for that file. If you specify **– –** in place of both *file1* and *file2*, the output is undefined.

Conceptually, **join** computes the Cartesian product of records from both files. By default, spaces or tabs separate input fields and **join** discards any leading or trailing white space. (There can be no white-space-delimited empty input fields.) It then generates output for those combined records in which the join key field (the first field by default) matches in each file. The default output for **join** is the common join key field, followed by all the other fields in *file1*, and then all the other fields in *file2*. The other fields from each file appear in the same order they appeared in the original file. The default output field separator is a space character.

Options

–a *n*
Produces an output line for lines that do not match in addition to one for a pair of records that does match. If you specify *n* as one of 1 or 2, **join** produces unpaired records from only that file. If you specify both **–a** 1 and **–a** 2, it produces unpaired records from both files.

–e *string*
Replaces an empty field with *string* on output. In a doublebyte locale, *string* can contain doublebyte characters.

–j[*n***]** *m*
Uses field number *m* as the join key field. By default, the join key field is the first field in each input line. As with the **–a** option, if *n* is present, this option specifies the key field just for that file; otherwise, it specifies it for both files.

–o *list*
Specifies the fields to be output. You can specify each element in *list* as either *n.m*, where *n* is a file number (1 or 2) and *m* is a field number, or as 0 (zero), which represents the join field. You can specify any number of output fields by separating them with blanks or commas. The POSIX-compatible version of this command (first form in the syntax) requires multiple output fields to be specified as a single argument; therefore, shell quoting may be necessary. **join** outputs the fields in the order you list them.

–t *c* Sets the field separator to the character *c*. Each instance of *c* introduces a new field, making empty fields possible. In a doublebyte locale, *c* can be a doublebyte character.

–v *n*
Suppresses matching lines. If you specify *n* as one of 1 or 2, **join** produces unpaired records from only that file. If you specify both **–v** 1 and **–v** 2, it produces unpaired records from both files. This does not suppress any lines produced using the **–a** option.

–1 *n*
Uses the *n*th field of *file1* as the join key field.

–2 *n*
Uses the *n*th field of *file2* as the join key field.

Examples

1. The following script produces a report about files in the working directory containing filename, file mode, and an estimate at what the file contains:

```
file * | tr –s ':' ';' >temp1
ls –l | tr –s ' ' ' ';' >temp2
join –t';' –j2 9 –o 1.1 2.1 1.2 ---
temp1 temp2
rm temp[12]
```

2. This example uses the historical implementation of the **join** command. The third line in the POSIX-compatible script could be:

```
join –t';' –2 9 –o 1.1,2.1,1.2 -- temp1 temp2
```

Localization

join uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion
1 Failure due to any of the following:
- Incorrect syntax
- The wrong number of command-line arguments
- Inability to open the input file
- Badly constructed output list
- Too many **–o** options on the command line

2 Failure due to an incorrect command-line argument

Messages

Most diagnostics deal with argument syntax and are self-explanatory. For example:

Badly constructed output list at *list*
Indicates that the list for a **–o** option did not have the proper syntax.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

POSIX considers the **–j** option to be obsolete.

Related Information

awk, **comm**, **cut**, **paste**, **sort**

---

kill — End a process or job, or send it a signal

Format

kill –l [*exit_status*]
kill [**–s** *signal_name*] [*pid ...*] [*job-identifier ...*]
kill [**–***signal_name*] [*pid ...*] [*job-identifier ...*]
kill [**–***signal_number*] [*pid ...*] [*job-identifier ...*]

tcsh shell:

kill [*-signal*] *%job|pid ...*

kill -l

Description

kill ends a process by sending it a signal. The default signal is **SIGTERM**.

kill in the tcsh shell

In the tcsh shell, **kill** [*-signal*] *%job|pid ...* sends the specified *signal* (or if none is given, the TERM (terminate) signal) to the specified jobs or processes. *job* may be a number, a string, '', %, + or - . Signals are either given by number or by name. When using the tcsh **kill** command, do not use the first three characters (*SIG*) of the *signal_name*. Enter the *signal_name* with uppercase characters. For example, if you want to send the **SIGTERM** signal, you would enter **kill -TERM** *pid* not **kill -SIGTERM** *pid*.

There is no default *job*. Specifying **kill** alone does not send a signal to the current job. If the signal being sent is TERM or HUP (hangup), then the job or process is sent a CONT (continue) signal as well.

kill -l lists the signal names. See "tcsh — Invoke a C shell" on page 556.

The *signal_numbers* and *signal_names* described in "Options" are also used with the tcsh **kill** command.

Options

–l Displays the names of all supported signals. If you specify *exit_status*, and it is the exit code of a ended process, **kill** displays the ending signal of that process.

–s *signal_name*
Sends the signal *signal_name* to the process instead of the **SIGTERM** signal. When using the **kill** command, do not use the first three characters (*SIG*) of the *signal_name*. Enter the *signal_name* with uppercase characters. For example, if you want to send the **SIGABRT** signal, enter:

```
kill –s ABRT pid
```

–signal_name
(Obsolete.) Same as **–s** *signal_name*.

–signal_number
(Obsolete.) A non-negative integer representing the signal to be sent to the process, instead of **SIGTERM**.

The *signal_number* represents the *signal_name* shown below:

signal_number	***signal_name***
0	**SIGNULL**
1	**SIGHUP**
2	**SIGINT**
3	**SIGQUIT**
4	**SIGILL**
5	**SIGPOLL**
6	**SIGABRT**
7	**SIGSTOP**
8	**SIGFPE**
9	**SIGKILL**
10	**SIGBUS**
11	**SIGSEGV**
12	**SIGSYS**
13	**SIGPIPE**
14	**SIGALRM**
15	**SIGTERM**
16	**SIGUSR1**
17	**SIGUSR2**
18	**SIGABND**
19	**SIGCONT**
20	**SIGCHLD**
21	**SIGTTIN**
22	**SIGTTOU**
23	**SIGIO**
24	**SIGQUIT**

25	**SIGTSTP**
26	**SIGTRAP**
27	**SIGIOERR**
28	**SIGWINCH**
29	**SIGXCPU**
30	**SIGXFSZ**
31	**SIGVTALRM**
32	**SIGPROF**
38	**SIGDCE**
39	**SIGDUMP**

Note: The *signal_number*s (3 and 6) associated with **SIGQUIT** and **SIGABRT**, respectively, differ from the values of **SIGQUIT** and **SIGABRT** used by the OS/390 kernel, but they are supported for compatibility with other UNIX platforms. (The **kill** command will send the OS/390 **SIGQUIT** or **SIGABRT** to the process.) (This note is also true for **kill** in the tcsh shell.)

Options

job-identifier
Is the job identifier reported by the shell when a process is started with **&**. It is one way to identify a process. It is also reported by the **jobs** command. When using the job identifier with the **kill** command, the job identifier must be prefaced with a percent (%) sign. For example, if the job identifier is 2, the **kill** command would be entered as follows:

```
kill -s KILL %2
```

pid Is the process ID that the shell reports when a process is started with **&**. You can also find it using the **ps** command. The *pid* argument is a number that may be specified as octal, decimal, or hex. Process IDs are reported in decimal. **kill** supports negative values for *pid*.

If *pid* is negative but not -1, the signal is sent to all processes whose process group ID is equal to the absolute value of *pid*. The negative *pid* is specified in this way:

```
kill -KILL -- -nn
```

where *nn* is the process group ID and may have a range of 2 to 7 digits (*nn* to *nnnnnnn*).

```
kill -s KILL -- -9812753
```

The format must include the -- before the *-nn* in order to specify the process group ID.

If *pid* is 0, the signal is sent to all processes in the process group of the invoker.

The process to be killed must belong to the current user, unless he or she is the superuser.

Localization

kill uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

- **NLSPATH**

See Appendix F for more information.

Usage Notes

kill is a built-in shell command.

Exit Values

`0` Successful completion

`1` Failure due to one of the following:

- The job or process did not exist
- There was an error in command-line syntax

`2` Failure due to one of the following:

- Two jobs or processes did not exist
- Incorrect command-line argument
- Incorrect signal

`>2`
Tells the number of processes that could not be killed

Messages

Possible error messages include:

job-identifier **is not a job**
You specified an incorrect ID.

signal_name **is not a valid signal**
You specified a noninteger signal for **kill**, or you specified a signal that is outside the range of valid signal numbers.

Portability

POSIX.2, X/Open Portability Guide.

Related Information

jobs, **ps**, **sh**, **tcsh**

[(left bracket) — Test for a condition

See the **test** command.

Note: When working in the shell, to view man page information about **[** (left bracket), type: `man left`.

---

let — Evaluate an arithmetic expression

Format

let *expression ...*
((*expression***))**

Description

let evaluates each arithmetic *expression* from left to right, with normal algebraic precedence (multiplication before addition, for example). **let** uses long integer arithmetic with no checks for overflow. No output is generated; the exit status is `0` if the last *expression* argument has a nonzero value, and `1` otherwise.

The following two lines are equivalent: the second form avoids quoting and enhances readability. These two forms are extensions to the POSIX standard.

```
let "expression"
((expression))
```

The POSIX version of this command is as follows:

```
$((expression))
```

Expressions consist of named variables, numeric constants, and operators. Characters in the names of named variables must come from the POSIX portable character set.

See “Arithmetic Substitution” on page 499

Examples

Examples of the three forms of the **let** command are as follows:

1. The example

   ```
   let a=7
   echo $a
   ```

   produces:

   ```
   7
   ```

2. The example

   ```
   echo $((a=7*9))
   ```

   produces:

   ```
   63
   ```

3. The example

   ```
   ((a=3*4))
   echo $a
   ```

   produces:

   ```
   12
   ```

Usage Note

let is a built-in shell command.

Localization

let uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 The last argument evaluated to a nonzero value

1 The last argument evaluated to a zero value, or the expression contained a syntax error or tried to divide by zero

Portability

POSIX.2. **let** and *((expression))* are extensions to the POSIX.2 standard. The POSIX.2 portable facility for arithmetic expression evaluation is *$((expression))*. See "Arithmetic Substitution" on page 499 for more information.

The (( )) syntax only works if the **set –o korn** option is in effect.

Related Information

expr, **sh**, **test**

---

lex — Generate a program for lexical tasks

Format

lex [**–achlntTv**] [**–o** *file.c*] [**–P** *proto*] [**–p** *prefix*] [*file.l ...*]

Description

lex reads a description of a lexical syntax, in the form of regular expressions and actions, from *file.l*. If you do not provide *file.l*, or if the file is named **–**, **lex** reads the description from standard input (**stdin**). It produces a set of tables that, together with additional prototype code from **/etc/yylex.c**, constitute a lexical analyzer to scan those expressions. The resulting recognizer is suitable for use with **yacc**. You can find detailed information regarding the use of **lex** in *OS/390 UNIX System Services Programming Tools*.

For a description of the typedefs, constants, variables, macros, and functions in the table file, which can be used to access the lexical analyzer's variables or to control its operations, see *OS/390 UNIX System Services Programming Tools*.

Options

–a Generates 8-bit tables instead of 7-bit tables. On systems with 8-bit character sets (such as this one), this option is always enabled.

–c Generates C code. Because this is the default, this option is provided only for compatibility with other implementations.

–h Prints a brief list of the options and quits.

–l Suppresses **#line** directives in the generated code.

–n Suppresses the display of table sizes by the **–v** option. If you did not specify **–v** and their are no table sizes specified in *file.1*, **lex** behaves as though you specified **–n**.

–o *file.c*
Writes the lexical analyzer (internal state tables) onto the named output file, instead of the default file **lex.yy.c**.

–P *proto*
Uses the named code file, instead of the default prototype file **/etc/yylex.c**.

–p *prefix*
Uses the given prefix instead of the prefix **yy** in the generated code.

–T Writes a description of the analyzer onto the file **l.output**.

–t Writes the lexical analyzer onto standard output (**stdout**) instead of the file **lex.yy.c**.

–v Displays the space used by the various internal tables. Normally **lex** displays these statistics on **stdout**, but if you also specified the **–t** option, it displays them on **stderr**. If you did not choose this option and *file.1* specifies table sizes, **lex** still displays these statistics unless you specified the **–n** option.

The **lex** library contains a number of functions essential for use with **lex**. These functions are described in *OS/390 UNIX System Services Programming Tools* . The actual library to use depends on your system and compiler. For OS/390 programs, you should use **–ll**.

Some **lex** programs can cause one or more tables within **lex** to overflow. These tables are the NFA, DFA, and move tables; **lex** displays an appropriate message if an overflow occurs. You can change table sizes by inserting the appropriate line into the *definition* section of the **lex** input, with the number *size* giving the number of entries to use. This is shown in Figure 8.

Figure 8. Internal Table Sizes (lex command)

Line	Table Size Affected	Default
%e*size*	Number of NFA entries	1000
%n*size*	Number of DFA entries	500
%p*size*	Number of move entries	2500

You can often reduce the NFA and DFA space to make room for more move entries.

Locale

A *locale* is the subset of a user's environment that depends on language and cultural conventions. A locale defines such things as the definition of characters, and the collation sequence of those characters. POSIX.2 defines a POSIX locale, which is essentially USASCII.

Since **lex** generates code that is then compiled before being executed, it is difficult for **lex** to act properly on collation information. The POSIX.2 standard therefore does not require **lex** to accept any locales other than the POSIX locale. **lex** accepts regular expressions in this locale only.

Files

lex uses the following files:

l.output	Scanner machine description
lex.yy.c	Tables and action routines
/etc/yylex.c	The prototype **lex** scanner
/usr/lib/libl.a	**lex** function library

Localization

lex uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Values

`0` Successful completion

`1` Failure due to any of the following:

- Inability to create an output file
- Inability to open the file
- Missing output filename after **–o**
- Missing prefix after **–p**
- No **lex** rules
- No memory for DFA moves
- Out of NFA state space
- Out of DFA move space
- Out of DFA state space
- Push-back buffer overflow
- Read error on file
- Table too large for machine
- Too many character classes
- Too many translations
- Unknown option
- Write error on file
- Incomplete `%{` declaration

- Token buffer overflow

Limits

The parser stack depth is limited to 150 levels. Attempting to process extremely complicated syntaxes may result in an overflow, causing an error.

Portability

POSIX.2, POSIX.2 C-Language Development Utilities Option, UNIX systems.

The **–a**, **–h**, **–l**, **–o**, **–p**, **–P**, and **–T** options are extensions of the POSIX standard.

Related Information

yacc

For more information, see *OS/390 UNIX System Services Programming Tools*

---

line — Copy one line of standard input

Format

line

Description

line copies one input line from its standard input (**stdin**) to its standard output (**stdout**). The end of the line is the first newline encountered. This is useful in shell files that need small amounts of input (for example, responses to prompts).

Examples

```
echo "Enter name:\c"
NAME= line
```

Localization

line uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 A line was read successfully
1 **line** reached end-of-file before finding a newline character

Portability

X/Open Portability Guide, UNIX System V.

Related Information

cat, **head**, **read**, **sh**, **tail**

---

link — Create a hard link to a file

Format

link *oldfile newfile*

Description

link creates a hard link to an existing file. A link is a new directory entry that refers to the same file. This entry can be in the same directory that currently contains the file or in a different directory. The result is that you get a new pathname that refers to the file. You can access the file under the old or new pathname since both pathnames are of equal importance. If you use **rm** to remove one pathname, the other remains and the file contents are still available under that name. The contents of the file do not disappear until the last remaining link associated with the file is removed.

Following the format, *new* becomes a new pathname for the existing file *old*. If *old* names a symbolic link, **link** creates a hard link to the file that results from resolving the pathname contained in the symbolic link.

Links are allowed to files only, not to directories. A file can have any number of links to it. Thus, you can establish any number of different pathnames for any file.

link is implemented as a shell built-in.

Localization

link uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Values

`0` Successful completion

`1` Failure due to any of the following:

- A file specified could not be found
- No write permission on the directory intended to contain the link
- No search permission on a pathname component of old or new
- No permission to access old

- The pathname of one of the arguments is a directory
- The new link file already exists

2 Failure due to incorrect number of arguments

Related Information

link, **ln**, **rm**

---

lm — Starting login monitor for OCS support

Format

lm

Description

lm starts the login monitor daemon. The login monitor is a user-level daemon that starts the login process in the OS/390 shell for logins initiated by Outboard Communications Server (OCS). OCS nodes then forward service requests to the login monitor, which listens for the requests on a port number assigned to the **lm** service.

After receiving a request from one of its OCS server nodes to start the login process, the login monitor determines the name of the character special remote-tty (rty) file in the HFS that corresponds to the requesting terminal or pseudoterminal on the originating OCS node.

The login monitor initializes the terminal characteristics, performs user authentication, sets up the user's environment, and **exec()**s to the specified initial program (typically a login shell).

You have to be a superuser to use the **lm** command.

Localization

lm uses the following localization environment variable:

- **NLSPATH**

See Appendix F for more information.

Files

/etc/banner
Contains the herald message that is displayed prior to the login prompt.

TCPIP.ETC.SERVICES, /etc/services
Defines the port for the **lm** service.

Usage Notes

1. All messages issued by the login monitor are directed to **stderr**.
2. The login monitor is normally started during kernel initialization if an entry in the **/etc/rc** file on the system is included. For example, an entry to invoke **/etc/lm.rc** can be included in **/etc/rc**, where **/etc/lm.rc** is created as follows:

```
export _BPX_JOBNAME=LM
/usr/sbin/lm 2> /etc/lm.log &
```

The statements perform the following functions:

- The first statement sets the jobname for the logon monitor address space to **LM**, instead of the default name of **ETCINIT ETCRC.**
- The second statement starts the login monitor. Also, all login monitor error messages written to **stderr** are redirected to **/etc/lm.log**.

  Note that the login monitor initializes itself as a daemon process; there is no need to start the login monitor as a background job.

The login monitor can also be started as a cataloged procedure using BPXBATCH. See *OS/390 UNIX System Services Planning* for more details.

3. The login monitor must be started after at least one OCS node is configured with the **ocsconfig** command.
4. During the initialization, if the login monitor cannot determine the port assigned to the **lm** service, it uses a default port of 801.
5. As a part of the user login process, the herald message that is specified in **/etc/banner** is displayed at the user terminal. This message is displayed prior to user authentication.

   The herald message is located in **/etc/banner**. If a herald message is used, it must contain information coded in the codeset IBM-1047.
6. After the OCS system configuration is changed by adding, deleting, or reconfiguring one or more OCS nodes, issue **kill —HUP** against the PID (process identifier) for the login monitor. This signals the login monitor to reread the mapping of the OS/390 UNIX and OCS device names and device numbers from the OCS administrative pseudodevice driver.

Exit Values

0 Successful completion
1 Invoker does not have superuser authority
2 Unable to **open()** the OCSADMIN device driver **/dev/ocsadmin**
3 Socket **open()**, **bind()** or **listen()** failure
4 The TCP port for the **lm** service could not be determined
5 A failure occurred in an **iconv** service
6 Could not obtain enough storage to initialize
7 Communication between the host and AIX server failed

Related Information

kill, **ocsconfig**

---

ln — Create a link to a file

Format

ln [**–fiRrse**] *old new*
ln [**–fiRrse**] *old old ... dir*

Description

ln creates a link to an existing file or set of files. A *link* is a new directory entry that refers to the same file. This entry can be in the same directory that currently contains the file or in a different directory. The result is that you get a new pathname that refers to the file. You can access the file under the old pathname or the new one. Both pathnames are of equal importance. If you use **rm** to remove either name, the other one still remains and the file contents are still available under that name. The contents of the file do not disappear until you remove the last link.

A file can have any number of links to it. Thus you can establish any number of different pathnames for any file.

In the first form given in the syntax, *new* becomes a new pathname for the existing file *old*. In the second form, **ln** creates entries for all the *old* files under the directory *dir*. For example:

```
ln yourdir/* mydir
```

creates links under **mydir** to all the files under **yourdir**. The files have the same names under **mydir** that they had under **yourdir**. **ln** always assumes this directory form when the last operand on the command line is the name of a directory. In this case, none of the *old* names can be a directory, unless —r or —R is specified.

There could already be a file with the same name as the link you are trying to set up: a *conflicting* pathname. To deal with a conflicting pathname, **ln** follows these steps.

- If you have specified **–i**, **ln** writes a prompt to **stderr** to ask if you want to get rid of the conflicting pathname. If you answer affirmatively, **ln** attempts to remove it.
- Otherwise, if you have specified **–f**, **ln** attempts to remove the existing file without a warning.
- Otherwise, **ln** prints a diagnostic message.
- **ln** gets to this point if it is going to get rid of the conflicting pathname. It therefore attempts to get rid of the conflicting pathname in the same way that **rm** does. **ln** deletes the file associated with the pathname if this pathname is the last link to the file. If **ln** can't get rid of the conflicting pathname, it does not attempt to establish the new link; it simply prints an error message on **stderr** and goes on to process any other files.
- If **ln** successfully gets rid of the conflicting pathname, it then establishes the link.

Options

–e Specifies that the link created by **ln** be an external link. One purpose for creating an external link is to create a mount point that an NFS client can use to access a data set through the DFSMS/MVS Network File System feature. The normal content of an external link is a name that refers to an object outside the hierarchical file system, such as a data set. The data set that the DFSMS/MVS Network File System feature uses can be any type of MVS data set. For a partitioned data set, however, you specify a fully qualified name in all caps. For example:

```
ln -e NOLL.PLIB.PGMA /u/noll/plib/pgma
```

External links can also be used to map an HFS file name to a PDS or PDSE member name for an executable load module. An example of how you would define the external link is:

```
ln -e MYPGM /u/smorg/mylongpgmname
```

If an application attempts to access **/u/smorg/mylongpgmname** as an executable file, the kernel will attempt to load MYPGM from the current MVS search order (Job Pack Queue, STEPLIB/JOBLIB, LPA, LINK LIST). The kernel services which behave this way for external links are:

- **exec()** (all flavors)
- **spawn()** (including _spawn2, spawnp, _spawnp2)
- **loadhfs** which is used for all DLL processing and locales

Note: For OS/390 releases prior to Release 6, an external link name cannot be specified as a shell command. Starting in Release 6, an external link can be used as a shell command to invoke a program in the current MVS search order.

–f Gets rid of any conflicting pathnames without asking you for confirmation.

–i Checks with you before getting rid of conflicting pathnames. You must not specify both **–f** and **–i**.

–R Links files recursively. That is, you can link an entire hierarchy of subdirectories at once.

–r Is identical to **–R**.

–s Creates a symbolic link.

Note, for a symbolic link,*old* refers to the file you want to create the link to (this file does not have to exist). *new* is the name of the symlink you are creating. For example, if you have a file called **f1** and you want to create a symlink to it called **my_sym**, you input the following:

```
ln -s f1 my_sym
```

The locale settings for **LC_COLLATE**, **LC_CTYPE**, and **LC_MESSAGES** affect the program's interpretation of what constitutes a “yes” answer when **ln** asks if you want to get rid of a conflicting pathname.

Examples

If you define **/u/user1/name1** as a symbolic link to **/u/user1/name2**, and then invoke *name1*:

1. The shell will spawn *name1*.
2. **spawn()** will access the file for *name1* unaware that there is a symbolic link already established. It will access the *name2* file by its underlying vnode, not the *name2* handle.
3. If the sticky bit is on for the *name2* file, **spawn()** will do the MVS search for *name1* (the only name it has to work with).

Symbolic and external links with a sticky bit:

Note: DLLs, and all flavors of **spawn()** and **exec()**, follow the same processing as described below. Where it says **exec()**, it covers all forms of module loading.

1. External links:

   exec() does a **stat()** on the passed filename. **stat()** does the search, not **exec()**. If the filename is an external link, then **stat()** fails with a unique reason code which causes **exec()** to read the external link. If the external link name is a valid PDS member name (1–8 alphanumeric/special characters), then **exec()** will attempt to locate the module in the MVS search order. If it cannot be found, **exec()** fails.

   The external link is normally used when you want to set the sticky bit on for a file name which is longer than 8 characters or contains characters unacceptable for a PDS member name.

2. Symbolic links:

   If the filename you specify is a symbolic link, and **exec()** sees the sticky bit on, then it will truncate any dot qualifiers. So, as long as the base filename is an acceptable PDS member name, the need to set up links in order to get **exec()** to go to the MVS search order should not be an issue.

   For example, if you have a file named java.jll, when you put the sticky bit on, **exec()** will attempt to load JAVA. If **exec()** cannot find JAVA, it will revert to using the java.jll file in the file system.

   The important thing to understand is that **exec()** never sees the name that the symbolic link resolves to even though it can see the **stat()** data for the final file.

Localization

ln uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Values

`0` All requested links were established successfully.

`1` Failure due to any of the following:

- An argument had a trailing / but was not the name of a directory.
- A file could not be found.
- An input file could not be opened for reading.
- An output file could not be created or opened for output.
- The new link file already exists.
- A link could not be established.
- A read error occurred on an input file.
- A write error occurred on an output file.
- The input and output files were the same file.
- Inability to access a file when using **–r**.
- Inability to read a directory when using **–r**.
- Inability to create a directory when using **–r**.
- A target is not a directory when using **–r**.
- Source and destination directory are the same when using **–r**.

2 Failure due to any of the following:

- Incorrect command-line option.
- Too few arguments on the command line.
- A target that should be a directory but isn't.
- No space left on target device.
- Out of memory to hold the data to be copied.
- Inability to create a directory to hold a target file.

Messages

Possible error messages include:

link to target *name* **failed**
ln could not establish the link to the given file or directory. This may be because you do not have appropriate permissions, or because the target did not exist.

source *name* **and target** *name* **are identical**
The source and the target are actually the same file (for example, because of links, on UNIX systems). In this case, **ln** does nothing.

target directory *name* **on different file system than source** *name*
You cannot establish a normal link between files that are two different file systems.

target *name* **must be a directory**
The target name must be a directory

cannot find file *name*
The filename could not be found.

target file *name* **already exists**
The target filename already exists.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Only the **–f** option is part of the POSIX standard.

Related Information

cp, **locale**, **mv**, **rm**

---

locale — Get locale-specific information

Format

locale [–a|–m]
locale [–ck] *name* ...

Description

locale displays information about the current locale and all locales accessible to the current application. **locale** searches directory **/usr/lib/nls/locale** for all the compiled locales.

Invoking **locale** with no options or operands displays the values of the **LANG** and **LC_*** environment variables. If a **LC_*** variable is not set or is overridden by **LC_ALL**, **locale** displays its implied value in double quotes.

The operand *name* can be a category name, keyword name, or the reserved name `charmap`. If it is a category name, **locale** selects the given category and all keywords within it for output. If *name* is a keyword name, **locale** selects the given keyword and its category for output. If *name* is `charmap`, **locale** displays the name of the charmap used with the **localedef –f** option when the locale was created.

Options

–a Displays information about all accessible locales including **POSIX**, the POSIX locale.

–c Displays the names of selected categories.

–k Displays the names of selected keywords. If you do not specify the **–k** option, **locale** displays the values of selected keywords but not their names. With **–k**, strings are written in an unambiguous form using the `escape` character from the current locale.

–m Displays a list of all available character maps.

The following list contains valid **locale** keywords:

lower	Lower-case alphabet
upper	Upper-case alphabet
alpha	All alphabetic characters (upper and lower case)
digit	All numeric characters
space	How white space is represented
cntrl	Control characters
punct	Punctuation characters
graph	Graphic characters
print	Printable characters
xdigit	Hex digits
blank	How a blank is represented
tolower	Upper to lower case conversion
toupper	Lower to upper case conversion
character-collation	The collating sequence
d_t_fmt	Date and time format
d_fmt	Date format
t_fmt	Time format

t_fmt_ampm	Long date format
am_pm	AM and PM string
day	Full day names
abday	Abbreviated day names
mon	Full month names
abmon	Abbreviated month names
decimal_point	Decimal-point characters
thousands_sep	Character used to separate groups of digits to the left of the decimal-point character in formatted nonmonetary quantities
grouping	String indicating the size of each group of digits in formatted nonmonetary quantities
int_curr_symbol	International currency symbol for the current locale
currency_symbol	Local currency symbol of the current locale
mon_decimal_point	Decimal-point character used to format monetary quantities
mon_thousands_sep	Separator for digits in formatted monetary quantities
mon_grouping	String indicating the size of each group of digits in formatted monetary quantities
positive_sign	String indicating the positive sign used in monetary quantities
negative_sign	String indicating the negative sign used in monetary quantities
int_frac_digits	The number of displayed digits to the right of the decimal place for internationally formatted monetary quantities
frac_digits	Number of digits to the right of the decimal place in monetary quantities
p_cs_precedes	1 if the `currency_symbol` precedes the value for a nonnegative formatted monetary quantity; 0 if it does not
p_sep_by_space	1 if the `currency_symbol` is separated by a space from the value of a nonnegative formatted monetary quantity; 0 if it does not; 2 if a space separates the symbol and the string–if adjacent
n_cs_precedes	1 if the `currency_symbol` precedes the value for a negative formatted monetary quantity; 0 if it does not
n_sep_by_space	1 if the `currency_symbol` is separated by a space from the value of a negative formatted monetary quantity; 0 if it does not; 2 if a space separates the symbol and the sign string–if adjacent
p_sign_posn	Value indicating the position of the `positive_sign` for a nonnegative formatted monetary quantity
n_sign_posn	Value indicating the position of the `negative_sign` for a negative formatted monetary quantity
yesexpr	Expression for affirmative

noexpr	Expression for negative
charmap	Mapping of character symbols to actual character encodings
code_set_name	Name of the coded character set used
mb_cur_max	Maximum number of bytes used to represent a character
codeset	Same as **code_set_name**
backslash	Encoding of \
right_bracket	Encoding of]
left_bracket	Encoding of [
right_brace	Encoding of }
left_brace	Encoding of {
circumflex	Encoding of ^
tilde	Encoding of ~
exclamation_mark	Encoding of !
number_sign	Encoding of #
vertical_line	Encoding of |
dollar_sign	Encoding of $
commercial_at	Encoding of @
grave_accent	Encoding of `

Examples

In the following examples, let's assume that locale environment variables are set as follows:

```
LANG=locale_x
LC_COLLATE=locale_y
```

1. The command:

   ```
   locale
   ```

   produces the following output:

   ```
   LANG=locale_x
   LC_CTYPE="locale_x"
   LC_COLLATE=locale_y
   LC_TIME="locale_x"
   LC_NUMERIC="locale_x"
   LC_MONETARY="locale_x"
   LC_MESSAGES="locale_x"
   LC_ALL=
   ```

2. The command:

   ```
   LC_ALL=POSIX locale -ck decimal_point
   ```

   produces:

   ```
   LC_NUMERIC
   decimal_point="."
   ```

3. The following command shows an application of **locale** to determine whether a user supplied response is affirmative:

```
if printf "s%\n" "$response" | grep -Eq "$(locale yesexpr)"
then
        affirmative processing goes here
else
        nonaffirmative processing goes here
fi
```

Localization

locale uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion
1 An error occurred
2 A usage message was printed

Portability

POSIX.2, UNIX System V.

Related Information

localedef

---

localedef — Define the locale environment

Format

localedef [**–c**] [**–f** *charmap*] [**–i** *sourcefile*] *name*

Description

localedef converts source definitions for locale categories into a format usable by functions and utilities.

localedef is provided by IBM C/C++ for MVS and is shipped with the compiler. This command requires the installation of Language Environment for MVS and VM. It also requires the installation of the Shell and Utilities and the C compiler so the **c89** command can be used. For more information, refer to *OS/390 UNIX System Services Planning*.

A TSO/E command called LOCALDEF is also provided with C/C++. It is not supported by the OS/390 shell. See *OS/390 C/C++ User's Guide* as well as information about the charmap file and locale definition source file formats.

Options

–c Creates permanent output even if there were warning messages. Normally, **localedef** does not create permanent output when it has issued warning messages.

–f *charmap*
Specifies a *charmap* file that contains a mapping of character symbols and collating element symbols to actual character encodings.

–i *sourcefile*
Specifies the file that contains the source definitions. If there is no **–i**, **localedef** reads the source definitions from the standard input.

name
Is the target locale. If it contains no slashes, the locale is public and **localedef** converts *name* to a full pathname using the **NLSPATH** environment variable. If *name* contains one or more slashes, **localedef** interprets it as a full pathname of where to store the created definition.

See "locale — Get locale-specific information" on page 316 for related information.

Localization

localedef uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

The **LC_COLLATE** and **LC_CTYPE** environment variables have no effect on **localedef**. **localedef** always behaves as though these variables were set to the POSIX locale.

See Appendix F for more information.

Exit Values

`0` No errors occurred; the locale was successfully created.

`1` Warnings occurred; the locale was successfully created.

`2` The locale specification exceeded implementation limits, or the coded character set used was not supported by implementation. No locale was created.

`3` The capability to create new locales is not supported by the implementation. (**POSIX2_LOCALEDEF** is not defined.)

`>3`
Warnings or errors occurred; no output was created.

localedef issues warnings when:

- The **LC_CTYPE** or **LC_COLLATE** category description uses a symbolic name not found in the *charmap* file.
- The number of operands to the **order** keyword exceeds the `COLL_WEIGHTS_MAX` limit.

Portability

POSIX.2; UNIX System V.

Related Information

locale

---

logger — Log messages

Format

logger [**–lisTu**] [**–d** *dest*] [**–f** *filename*] [**–p** *priority*] [**–t** *tag*] [**–a** *tag2*] *string ...*

Description

logger saves a message in the console log; the message consists of the *string* operand on the command line. Some options of **logger** may be in effect by default; if they are on by default, they cannot be disabled.

The **–u** and **–i** options are in effect by default, so all messages from **logger** are prefixed by process ID and user login user name.

If there is no message specified on the command line, the standard input is read; each line of standard input is treated as a log message. If **–f** *filename* is specified, the file is read instead of the standard input.

Options

–f *filename*
Reads log messages from the file *filename* rather than from the standard input.

–l Adds the parent process ID (PPID) of **logger** to the message.

–i Adds the process ID (PID) of **logger** to the message. This option is in effect by default, so all messages from **logger** are prefixed by the PID.

–s Overrides any destination options and causes logging to the standard error output.

–T Adds a time stamp (`%x %X` format, per date) to the message. This time stamp is always in the POSIX locale, no matter the locale of the message.

–u Adds the login name of the controlling terminal to the message. This option is in effect by default, so all messages from **logger** are prefixed by the login name.

Note: The following options work on OS/390. However, because they are system-specific, they may or may not actually work on another system.

–d *destination*
Must be a list of numbers, separated by spaces, tabs, or commas, in the range of 1 to 128, and represents a bit in the routing code number (that is, ROUTCDE=) in the WTO macro. The default destination value is `0` (no bits set in the routing code number).

If you use **d1**, the message goes to the system console.

–p *priority*
Must be a list of numbers, separated by spaces, tabs, or commas, in the range of 1 to 16 and represents a bit in the message descriptor code (that is, DESC=) in the WTO macro (WTO == write to operator). The default priority value is 0 (that is, no bits set).

–t *tag*
Adds *tag* to the start of the message.

–a *tag2*
Adds *tag2* in front of all the options and the message.

For more information on the *destination* and *priority* options, refer to *OS/390 MVS JCL Reference*.

Examples

1. If you issue:

   ```
   logger -d1 This is a message.
   ```

   You will see:

   ```
   +WELLIE4: 2097152017: This is a message.
   ```

2. If you issue:

   ```
   logger -dl -a TheTag A message.
   ```

   You will see:

   ```
   +TheTag: WELLIE4: 213076449: A message.
   ```

Localization

logger uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

`0` Successful completion

`>0`
An error occurred

Messages

Possible error messages include:

-f *filename* **invalid if message given**
Both a filename and message was specified; only one is allowed.

file *filename*: **system error**
The file specified by **–f** *filename* could not be opened.

Formatted log message too long -- limit LINE_MAX (*number***)**
The log message specified was longer than the limit specified by **LINE_MAX**.

Unknown option *option*
You specified an incorrect option to **logger**.

Portability

POSIX.2, X/Open Portability Guide.

All the options are extensions of the POSIX standard.

---

logname — Return a user's login name

Format

logname

Description

logname displays the login name of the person who issued the command. It obtains this through the **getlogin()** function defined in the POSIX.1. standard. The login name is displayed as all uppercase letters, regardless of how it was entered.

Environment Variables

logname uses the following environment variable:

LOGNAME
Contains your user name.

Localization

logname uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_TYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion
1 **logname** could not determine the login name

Portability

POSIX.2, X/Open Portability Guide, UNIX System V.

Related Information

env, **id**

lp — Send a file to a printer

Format

lp [**–cmsw**] [**–d** *dest*] [**–n** *number*] [**–o** *printer-option*] [**–t** *title*] [*file ...*]

Description

lp prints one or more input files on a printer. If you do not specify any files on the command line, or if you specify a filename of **–lp** reads and prints the standard input. The files are printed in the same order that they are specified on the command line.

Note: If you are using the OS/390 Print Server Feature, your system automatically uses that version of the **lp** command. For more information about the OS/390 Print Server commands, see *OS/390 Infoprint Server User's Guide*.

Options

–c Immediately copies the files to be printed. This ensures that the version of the file that exists when the print request is made is the version printed.

–d *dest*
Specifies *dest* as the output device. **–d** takes precedence over the **LPDEST** environment variable, which in turn takes precedence over the **PRINTER** environment variable.

dest is a comma-separated list of arguments that is passed to JES. The first item must be the “destination_name”. The destination name can take the form *NODE.USER*. The second item must be the “class”. The third item must be the “forms”. Not all items must be specified, but the items must be specified in the proper order. The definition of “destination_name”, “class”, and “forms” is defined by JES.

For more information on the *dest* option, see *OS/390 MVS JCL Reference*.

–m This option is not implemented.

–n *number*
Prints *number* copies of each input file (the default is 1 copy).

–o *printer-option*
This option is not implemented.

–s This option is not implemented.

–t This option is not implemented.

–w This option is not implemented.

Examples

1. To send a previously formatted file to a JES printer:

   ```
   lp filename
   ```

   You can specify more than one filename with the command.

2. The following prints the file **temp.prt** using the default printer destination and specifying class `c` (where `c` is the locally designated class for confidential information):

```
lp –d ,c temp.prt

lp –d,c temp.prt
```

The parameters on the **-d** option are positional, so if you omit a destination, you must still include the comma.

Environment Variables

lp uses the following environment variables:

LPDEST Names the output device. This variable takes precedence over **PRINTER**.

PRINTER Names the output device if **LPDEST** is not defined.

Localization

lp uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion

>0
An error occurred

Portability

POSIX.2, X/Open Portability Guide.

The **–m**, **–o**, **–s**, **–t**, and **–w** options are extensions to the POSIX standard.

---

lpstat — Show status of print queues (stub command)

Format

lpstat [**–drst**] [**–a** [*list*]] [**–c** [*list*]] [**–o** [*list*]] [**–p** [*list*]] [**–u** [*list*]] [**–v**] [*list*] [*queue_id ...*]

Description

lpstat shows the status of print queue or queues, specified by *queue_id*. If no *queue_id* is given, **lpstat** lists information for all of the printers on the system.

lpstat is recognized, but its functions are not supported.

If you are using the OS/390 Print Server Feature, your system automatically uses that version of the **lpstat** command. For more information about the OS/390 Print Server commands, see *OS/390 Infoprint Server User's Guide*.

ls — List file and directory names and attributes

Format

ls [**–AabCcdEFfgHiLlmnopqRrstuWx1**] [*pathname ...*]

Description

ls lists files and directories. If the *pathname* is a file, **ls** displays information on the file according to the requested options. If it is a directory, **ls** displays information on the files and subdirectories therein. You can get information on a directory itself using the **–d** option.

If you do not specify any options, **ls** displays only the filenames. When **ls** sends output to a pipe or a file, it writes one name per line; when it sends output to the terminal, it uses the **–C** (multicolumn) format.

Options

ls displays at least the filename; you can request more information with the following options:

–A Lists all entries including those starting with periods (**.**); but excluding any **.** or **..** entries.

–a Lists all entries including those starting with a period (**.**).

–b Displays nonprintable characters as octal bytes with the form *ooo*.

–C Puts output into columns, sorted vertically; this is the default output format to the terminal.

–c Uses the time of the last change of the file's attributes for sorting (**–t**) or displaying (**–l**) .

–d Does not display the contents of named directories, but information on the directories themselves.

–D Displays from directories.

–E Displays extended attributes for regular files:

- **a** Program runs APF-authorized if linked AC=1
- **p** Program is considered program-controlled
- **s** Program runs in a shared address space
- **l** Program is loaded from the shared library region
- **—** Attribute not set

See “Long Output Format” on page 329.

–H Displays file formats for regular files:

———	not specified
bin	binary data
nl	new line
cr	carriage return
lf	line feed
crlf	carriage return followed by line feed
lfcr	line feed followed by carriage return

crnl carriage return followed by new line

See “Long Output Format” on page 329.

–F Puts a **/** after each directory name, a ***** after every executable file, a **|** after every FIFO file, a **@** after every symbolic link, and a **=** after every socket. It also puts an & character after an external link name.

–f Forces the *pathname* argument to be a directory; turns off sorting. **ls** gives the ordered list of filenames in a directory file. The directory file is read and the filenames are listed in the same order as they are returned. The contents of a directory file are shown.

–g Same as **–l** except that it does not display owner.

–i Displays file serial (inode) numbers along with filenames.

–L Follows symbolic links.

–l Displays permissions, links, owner, group, size, time, name;

–m Displays names in a single line, with commas separating names.

–n Displays UID number and GID number.

–o Same as **–l** except that it does not display group.

–p Puts / after directory names.

–q Displays nonprintable characters as ?.

–R Lists subdirectories recursively.

–r Sorts in reverse of usual order; you can combine this with other options that sort the list.

–s Displays size in blocks, after the file serial (inode) number, but before other information.

–t Sorts by time. By default, this option sorts the output by the modification times of files. You can change this with the **–c** and **–u** options.

–u Uses the last access time for sorting (**–t**) or displaying (**–l**) .

–W Enables the audit bits to be displayed (see “chaudit” for information on audit bits). This option turns on the **–l** option.

These bits are printed in a 6-character field directly after the field displaying the file permission bits. These 6 characters are really two groups of 3 bits each. The first group of 3 describes the user-requested audit information. The second group of 3 describes the auditor-requested audit information. Each 3 characters displayed are the read, write, and execute (or search) audit options. Each character indicates the audit option as:

s (Audit successful audit attempts)
f (Audit failed access attempts)
a (Audit all accesses)
– (No audit)

–x Puts output into sorted columns, with output going across the rows.

–1 Forces output to be one entry per line.

Note: When you specify options that are mutually exclusive (for example, **–c** and **–u**), the option that appears last on the command line is used.

Long Output Format

The output from **ls –l** summarizes all the most important information about the file on a single line. If the specified *pathname* is a directory, **ls** displays information on every file in that directory (one file per line). It precedes this list with a status line that indicates the total number of file system blocks occupied by files in the directory (in 512-byte chunks). Here is a sample of the output along with an explanation:

```
total 11
drwxr-xr-x       3 ROOT   SYS1    0 Mar 12 19:32 tmp
drwxrwxrwx       4 ROOT   SYS1    0 Mar 12 19:32 usr
drwxr-xr-x       2 ROOT   SYS1    0 Mar 12 19:32 bin
-rwxr--r--       1 ROOT   SYS1  572 Mar 12 19:32 foo
-rwxr--r--       1 ROOT   SYS1  640 Mar 12 19:33 abc
```

The first character identifies the file type:

- `–` Regular file
- `b` Block special file (not supported for uss;)
- `c` Character special file
- `d` Directory
- `e` External link
- `l` Symbolic link
- `p` FIFO
- `s` Socket file type

The next 9 characters are in three groups of 3; they describe the permissions on the file. The first group of 3 describes owner permissions; the second describes group permissions; the third describes other (or “world”) permissions. Characters that may appear are:

- `r` Permission to read the file
- `w` Permission to write on the file
- `x` Permission to execute the file

The following characters appear only in the execute permission (`x`) position of the output.

- `S` Same as `s`, except that the execute bit is turned off.
- `s` If in owner permissions section, the set-user-ID bit is on; if in group permissions section, the set-group-ID bit is on.
- `T` Same as `t`, except that the execute bit is turned off.
- `t` The sticky bit is on.

You can set permissions with the **chmod** command.

After the permissions are set, **ls** displays the following (using the preceding example), in order:

- The number of links to the file.
- The name of the owner of the file or directory.
- The name of the group that owns the file or directory.
- The size of the file, expressed in bytes.

- For a file, the date and time the file was last changed; for a directory, when it was created. The **–c** and **–u** options can change which time value is used. If the date is more than 6 months old or if the date is in the future, the year is shown instead of the time.
- The name of the file or directory.

Note: When files owned by user ID 0 (**UID**=0) are transferred from any UNIX-type system across an NFS connection to another UNIX-type system, the **UID** changes to —2 (**UID**=–2). Because —2 is not a valid UID on an OS/390 System, **ls** displays a **—2** in place of the user name.

If **ls–E** is issued, an additional four characters follow the original 10 characters:

```
total 11
-rwxr-xr-x  -ps-   1 ROOT   SYS1  101 Mar 12 19:32 her
-rwxrwxrwx  a-s-   1 ROOT   SYS1  654 Mar 12 19:32 test
-rwxr-xr-x  a--    1 ROOT   SYS1   40 Mar 12 19:32 temp
-rwxr--r--  ap-l   1 ROOT   SYS1  572 Mar 12 19:32 foo
-rwxr--r--  --sl   1 ROOT   SYS1  640 Mar 12 19:33 abc
```

If **ls –H** is issued, an additional four characters follow the original 10 characters:

```
total 32
-rwxr-rx-x   ----   1 ROOT   SYS1     0 Mar 26 08:47 tmp
-drwxr-xr-x         2 ROOT   SYS1  8192 Mar 26 08:50 usr
-rwxr--r--   cr     1 ROOT   SYS1    40 Mar 26 08:55 abc
```

If **ls –E** is used in conjunction with **–H**, then the four characters will follow the four characters normally associated with **ls –E**:

```
ls -EH
-rwxr-rx-x  ap-l bin   1 ROOT   SYS1   101 Mar 12 19:21 foo
```

If **ls –W** is issued, an additional 6 characters, in two groups of 3, follow the original 10 characters. The first group of 3 describes the user-requested audit information; the second group describes auditor-requested audit information.

```
total 11
drwxr-xr-x  fff---     3 ROOT   SYS1    0 Mar 12 19:32 tmp
drwxrwxrwx  fff---     4 ROOT   SYS1    0 Mar 12 19:32 usr
drwxr-xr-x  fff---     2 ROOT   SYS1    0 Mar 12 19:32 bin
-rwxr--r--  fff---     1 ROOT   SYS1  572 Mar 12 19:32 foo
-rwxr--r--  fff---     1 ROOT   SYS1  640 Mar 12 19:33 abc
```

Usage Note

To display information about a directory from a symbolic link to the directory, either add a trailing slash to the symbolic link name, or use the **-L** option. For example, if the **/etc** directory has been converted into a symbolic link, issuing an **ls** on **/etc** *without* a trailing slash will give you the following information:

```
> ls -l /etc
lrwxrwxrwx   1 BPXROOT  BIN           12 Oct 18 19:46 /etc -> $SYSNAME/etc
```

However, if you add the trailing slash, the following information about **/etc** will be displayed:

```
> ls /etc/
```

IBM	**cmx**	**init.options**	**profile**	**utmpx**
NetQ	**csh.cshrc**	**ioepdcf**	**rc**	**yylex.c**
Printsrv	**csh.login**	**ldap**	**recover**	**yyparse.c**
TextTools	**dce**	**log**	**security**	**zoneinfo**
booksrv	**dfs**	**magic**	**socks.conf**	
bpe	**imoisinf**	**mailx.rc**	**startup.mk**	

The same information is displayed when the **-L** option is used:

```
ls -L /etc
```

IBM	**cmx**	**init.options**	**profile**	**utmpx**
NetQ	**csh.cshrc**	**ioepdcf**	**rc**	**yylex.c**
Printsrv	**csh.login**	**ldap**	**recover**	**yyparse.c**
TextTools	**dce**	**log**	**security**	**zoneinfo**
booksrv	**dfs**	**magic**	**socks.conf**	
bpe	**imoisinf**	**mailx.rc**	**startup.mk**	

Environment Variables

ls uses the following environment variables:

COLUMNS Contains the terminal width in columns. **ls** uses this value to determine the number of output columns to write using the **–C** option.

TZ Contains the time zone to be used when displaying date and time strings.

Localization

ls uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_TIME**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion
1 Failure due to any of the following:
- Out of memory
- Inability to find a file's information
- Too many directories
- File or directory not found
- Specified on the command line

2 Incorrect command-line option

Messages

Possible error messages include:

File or directory *name* **is not found**
The requested file or directory does not exist.

Cannot allocate memory for sorting
To sort its output, **ls** needs to allocate memory; this message says that there was not enough memory for the sorting operation.

Too many directory entries in *dir*
This message appears only when **ls** runs out of dynamically allocated memory.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The **–A**, **–b**, **–f**, **–g**, **–L**, **–m**, **–n**, **–o**, **–p**, **–s**, **–W**, and **–x** options are extensions of the POSIX standard.

Related Information

Appendix I explains how to set the local time zone with the **TZ** environment variable.

ls-f, **sh**, **tcsh**

---

mail — Read and send mail messages

Format

mail [**–e** | **–p**] [**–qr**] [**–f** *file...*]
mail [**–t**] *name...*

Description

mail lets you read mail sent to you and sends mail to other users. It has two modes of operation, one for reading mail and one for sending mail. If you start **mail** without any arguments, it checks for mail to be read and then presents the messages in read mode. If you start it with an argument or arguments, it assumes you are sending a message to the address named as the argument and enters send mode. The text of the message is taken from standard input until **mail** encounters either EOF or a line consisting of only a single dot (.).

For example, to read mail, enter:

```
mail
```

To send a mail message to the users Chris and Lee, enter:

```
mail chris lee
```

Options

The **–t** option is used only when sending mail; the others only when reading mail.

–e Tests for the existence of mail and exits. If there is mail in the system mailbox, the return status is successful.

–f Reads mail from *file* instead of the system mailbox. This option is often used to read mail saved in other files.

–p Prints all mail to standard output without querying.

–q Quits the **mail** session after an interrupt signal; normally, an interrupt ends only the message being written.

–r Saves messages in first-in, first-out order, the reverse of the default. Normally, the most recently received message is written first.

–t Lists the recipients at the beginning of the message (default).

Reading Mail

When you start **mail** without arguments, **mail** checks your system mailbox for mail. If there is no mail, **mail** exits with a return code of 1; if there is waiting mail, **mail** displays the first message. (If you specify **–p** on the command line, it displays all messages.)

Commands within **mail** control how messages are handled. The following commands are available:

d Deletes the current message.

m[*name...***]**
Sends the current message to the specified user. If a user is not specified, the mail is sent to you.

p Prints the message on the screen again.

q Quits mail, storing any undeleted messages in the file **$HOME/mbox**.

s[*file***]**
Saves the message in the specified file. If a file is not specified, **mail** saves the message in **mbox** in your home directory.

w[*file***]**
Saves the message (same as **s**), but without header lines.

x Exits mail without changing the mailbox file.

ENTER (or newline)
Displays the next message.

!*command*
Runs *command* using the shell.

\+ Displays the next message (same as ENTER or newline).

— Displays the previous message.

***** Displays a summary of internal commands.

Because the commands are read from standard input, you can create **mail** command files and use input redirection to have **mail** execute them.

Sending Mail

To send mail, start **mail** with a list of addresses as arguments. Enter the text of the message, and end the message with either EOF or with a single dot (.) on a line followed by a <newline>.

The **–t** option inserts at the beginning of the message a list of the addresses; a pathname beginning with a slash (/) is recognized as a valid address (assuming you have the correct permissions).

If the address is not valid or recognized, or if the message is interrupted (see the **–q** option), **mail** stores the message in the file **dead.letter** in the current directory. If it can't create **dead.letter** in the current directory, it creates the file in your home directory. If **dead.letter** already exists, the new contents overwrite the old.

The **mail** program modifies the message text slightly; because lines beginning with `From` (including the trailing space) are used to separate files in the mailbox, **mail** changes any lines in the message that begin with `From` to read `>From`.

Example

To send the file **how2mail** to user Chris, enter:

```
mail chris < how2mail
```

Usage Notes

1. Wherever the POSIX standard doesn't define the behavior of **mail**, this implementation resembles **mailx**.
2. **mail** doesn't require a delivery path or mechanism to the destination, though for most uses, this is preferable.

Environment Variables

mail uses the following environment variables:

MAKEFLAGS
Specifies your home directory; used to locate the **mbox** and **dead.letter** files.

TZ Specifies the time zone to be used in date and time strings.

Localization

mail uses the following localization environment variables:

- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_TIME**
- **NLSPATH**

See Appendix F for more information.

The ability of **mail** to handle doublebyte characters (or even 8-bit ASCII depends on the underlying mail transport mechanism. You should restrict all messages to the POSIX portable character set. To send messages containing doublebyte characters or even binary files, encode them first with **uuencode**.

Files

mail uses the following files:

dead.letter
The most recently canceled message.

mbox
The default file for saving read mail, stored in the directory specified by **HOME**.

Exit Values

0 The session was successfully completed; if reading, there was mail.

1 There was no mail, or the session could not be started.

2 An error occurred after starting the session, or you supplied an invalid option, resulting in a usage message.

Portability

POSIX.2

Because this utility is due to be withdrawn from POSIX, you may want to use **mailx** for portable applications. The ability to write directly to a file is an extension to POSIX.

Limits

Any individual line is limited to **LINE_MAX** bytes; of course, transport mechanisms between systems may impose shorter limits.

Related Information

mailx, **uudecode**, **uuencode**

Appendix I also explains how to set the local time zone with the **TZ** environment variable.

---

mailx — Send or receive electronic mail

Format

mailx [**–efHiNn**] [**–u** *user*] [*filename*]
mailx [**–FinU**] [**–h** *number*] [**–r** *address*] [**–s** *subjecf*] *user ...*

Description

mailx helps you read electronic mail messages. It can also send messages to users on your system, but it has no built-in facilities for sending messages to other systems.

The command line:

```
mailx [options] user user user ...
```

sends a mail message to the given users. If you do not specify any users on the command line, **mailx** lets you read incoming mail interactively. However, see the environment variable **sendmail**. It is described on page 353.

In a doublebyte locale, aliases, variables, and addresses can contain doublebyte characters.

This description of **mailx** is divided into several sections:

- Options
- General overview
- Command-mode subcommands
- Input-mode subcommands
- Startup files
- Example
- Environment variables
- Files
- Exit values
- Portability
- Related Information

Options

You can use the following options when reading messages:

–e Checks to see if you have any messages waiting to be read. With this option, nothing is displayed. If you have waiting messages, **mailx** exits with a successful status return; otherwise, **mailx** exits with a failure return.

–f *filename*
Looks for messages in the specified file instead of in your current mailbox. If you do not specify *filename*, **mailx** reads messages from **$HOME/mbox**.

–H Displays only the header summary of a message.

–N Does not display the header summary of messages.

–u *user*
Looks for messages in the system mailbox of the specified user. This works only if you have read permission on the user's system mailbox.

You can use the following options only when sending messages:

–F
Records your message in a file with the same name as the first user specified on the command line. This option overrides the **record** variable, if it has been set.

–h *number*
Indicates how many “hops” a message has already made from one machine to another (in a network of machines). This option is not intended for most users; network mail software uses the option to prevent infinite loops (the same message cycling through a sequence of machines without ever getting to its intended destination).

–r *address*
Passes the given address to network mail software. If this option is present, it disables all input mode commands. Again, this option is not intended for most users.

–s *subject*
Uses the given *subject* string in the `Subject` heading line of the message. If the subject contains spaces or tab characters, the string should be enclosed in double quotes or single quotes. If you specify this option on the command line, **mailx** does not prompt you to enter a subject line when you type in the text of the message.

–U
Converts the address from UNIX-to-UNIX Copy Program (UUCP) style to Internet Protocol standards. This option overrides the effect of the **conv** variable.

This option is not supported.

You can use these options ***for both sending and reading messages:***

–i Ignores interrupts (for example, from pressing <Break> or <Ctrl-c>).

–n Does not initialize your **mailx** session from the system's **/etc/mailx.rc** file.

General Overview

This section describes the default behavior of **mailx**.

The simplest command to send a message is:

```
mailx address address address ...
```

where each `address` names someone who is to receive the message. The simplest kind of address is the *login name* of someone else who uses your shell.

You can also send messages as input to commands. To do this, use an address that consists of a pipe symbol (|) followed by a command line that invokes the appropriate command; enclose this whole address in single quotes. For example:

```
mailx ROBIN '|cat >save'
```

mails a message to `ROBIN` and also copies the message into a file called **save**.

After you type in the command to send a message, **mailx** asks you to enter the subject of the message (a brief description of what the message is about), and then lets you type in the text of the message. This brief description can be up to 256 characters long. Your message can consist of any number of lines, and may include blank lines. When you finish entering the message, type a line consisting only>of a tilde (~), followed by a period (.); then press <Enter>. This tells **mailx** that the message is ready to be sent.

mailx puts the completed message into a file called the recipient's system mailbox. The message stays in the system mailbox until the recipient asks to read the message. At that point, the message is obtained from the system mailbox and displayed on the recipient's workstation. The message is then saved in the recipient's personal mailbox. Since this is usually a file named **mbox** in the recipient's home directory, we use the name *mbox* to represent the personal mailbox and *mailbox* for a system mailbox.

The simplest way to read incoming messages is to type the command **mailx** (with no addresses on the command line). This starts an *interactive session* in which **mailx** lets you read your mail and perform other operations. For example, you can display new messages, delete old ones, reply to messages, or forward them to

someone else, and so on. When you are performing operations in this way, you are in *command mode.* When you are typing in the text of a message, you are in *input mode*.

A message consists of a sequence of *header lines,* followed by the body of the message. The header lines tell who sent the message, the time and date that the message was sent, the subject of the message, and so on. **mailx** automatically creates header lines. Some of the common header lines are:

`Cc:` *name name* **...**
: Stands for "carbon copies". This indicates that copies of this message are to be sent to the specified recipients. The names of these recipients appear in the header lines of everyone receiving the message.

`Bcc:` *name name* **...**
: Stands for "blind carbon copies." This is similar to `Cc:`, but the names of people receiving carbon copies do not appear in the header lines of the message. Recipients do not know that these people received a copy of the message.

`Subject:` *text*
: Gives the subject of the message.

`To:` *name name* **...**
: Gives the names of people who were sent the message directly.

All messages are in one of the following states:

deleted
: You used a **delete**, **dp**, or **dt** command to delete the message, or you saved it using a **Save** or **save** command and the variable **keepsave** was not set. When **mailx** quits, messages in this state are deleted.

new The message is in the system mailbox and you have not yet read it or otherwise changed its state. When **mailx** quits, messages in this state are kept in your system mailbox.

preserved
: You used a **preserve** command on the message. When **mailx** quits, messages in this state are kept in their current locations.

read
: You used one of the following commands on the message:

```
~F          copy        Print       type
~f          mbox        print       undelete
~M          next        top
~m          pipe        Type
```

or you used **delete** , **dp**, or **dt** on the preceding message and the **autoprint** variable was set. When **mailx** quits and you are in your system mailbox, **read** messages are kept in your personal mailbox—unless the **hold** variable is set, in which case, **read** messages are kept in your system mailbox. If you are in your personal or a secondary mailbox when **mailx** quits, **read** messages are kept in their current location.

unread
: You have run more than one **mailx** session with the message in the system mailbox and you have not read it or otherwise changed its state. When **mailx** quits, messages in this state are kept in your system mailbox.

Command-Mode Subcommands

The standard format of a command-mode subcommand is:

[**subcommand**] [*refs*] [*arguments*]

If no **subcommand** is specified, **p[rint]** is assumed.

The *refs* argument indicates the messages to which you want to apply the **subcommand**. **mailx** numbers incoming messages sequentially as they are received. The easiest way to refer to a message is to give its number. For example, the subcommand:

```
p 3
```

displays message number 3. At any point in a **mailx** session, there is one message that is considered the *current message.* This is the message you most recently did something with (for example, the one you most recently read). If you omit the *refs* argument in a subcommand that uses *refs*, the subcommand works with the current message.

You can also use special notations as the *refs* value:

refs	Meaning
`n`	Message number `n`
`n-m`	Messages `n` through `m`
`.`	The current message
`^`	The first undeleted message (or first deleted message for *undelete*)
`$`	The last message
`+`	Next message
`–`	Previous message
`user`	All messages from the given user
`/string`	All messages with `string` in the subject line (the case of characters in `string` is ignored)
`:d`	All deleted messages
`:n`	All new messages
`:o`	All old messages
`:r`	All messages that have already been read
`:u`	All unread messages

Several *refs* arguments may be specified for the same subcommand, separated by spaces. For example:

```
p alice lewis
```

displays all messages from `alice` plus all messages from `lewis`.

The arguments allowed at the end of a command-mode subcommand depend on the subcommand itself. If a subcommand allows a filename as an argument, you can use the usual filename generation characters in the filename (see **sh**).

The following list shows the subcommands recognized in command mode. In every subcommand name, some characters are enclosed in square brackets. These characters are optional. For example, the **p**[**rint**] command may be given as **print** or **p**.

? Displays a summary of command-mode subcommands

= Displays the current message number

a[lias] [*alias* **[***name* **...]]**
Sets up an address *alias*. If you enter a subcommand to send mail to the given *alias*, the messages are actually sent to the given list of names. For example, you might enter the subcommand:

```
alias joe JSMITH
```

From this point onward, you can address messages to `joe` and they are sent to `jsmith`. You may also set up an alias for several people, as in:

```
alias choir SOPRANO ALTO TENOR BASS
```

After you have done this, you can send messages to `choir` and they are sent to the names that follow `choir` in the command.

If you use only one argument, **alias** lists the value of that alias. For example, `alias joe` would display `jsmith`. Entering the **alias** subcommand without any arguments displays a list of the currently defined aliases.

Note: Aliases entered interactively remain in effect only until the end of the current interactive session. To make an alias permanent, include the **alias** subcommand in your startup file (see “Startup Files” on page 348). See also **group**.

alt[ernates] *name*
Lists a set of alternate names for your own login name. This is useful for people who login under several different names. When you reply to a message, **mailx** usually sends your reply to the author of the message and all the recipients as well; however, it does not send the message to any of your alternate login names. You don't have to worry about sending mail to yourself.

Specifying alternates without names displays your list of currently defined alternate names.

cd *directory*
Makes *directory* your new working directory. If no *directory* is specified, **cd** goes to your **HOME** directory.

ch[dir] *directory*
Is the same as **cd**.

c[opy] [*refs***] [***filename***]**
Copies the messages referred to by *refs* into the given file. The *filename* must be specified. If the file does not already exist, it is created.

If no *refs* are specified, the current message is saved. If no *filename* is specified, your *mbox* is saved.

This operation does not mark the message as saved; if it was previously unread, it is still regarded as an unread message. Thus, the original message remains in your system mailbox. See also **save**.

C[opy] [*refs***]**
Is similar to the **copy** command, except that the messages referred to are saved in a file the name of which is derived from the author of the first message referred to. The name of the file is the author's name, stripped of any network addressing. If the **folder** variable is set, the file is saved to the specified directory. The copied messages are not marked as “saved”. If *refs* is not specified, the current message is copied.

d[elete] [*refs***]**
Deletes the specified messages from your system *mailbox*. If *refs* is not specified, the current message is deleted. After a delete operation, the current message is set to the message after the last message deleted. Deleted messages are not thrown away until you end your session with the current mailbox (see **quit** and **file**). Until then, they can be undeleted (see **undelete**).

di[scard] [*header***...]**
Does not display the given *header* fields when displaying a message. For example:

```
discard References
```

tells **mailx** not to display the `References` line at the beginning of any mail message. These header lines are retained when the message is saved; they are just not shown when the message is displayed. See also **ignore** and **retain**.

dp [*refs***]**
Deletes the specified messages and then displays the message after the last message deleted. If there is no subsequent message, **mailx** displays its command prompt.

dt [*refs***]**
Is the same as the **dp** subcommand.

ec[ho] *string* **...**
Echoes the given *strings* (like the **echo** subcommand).

e[dit] [*refs***]**
Lets you edit the messages specified by *refs*. The messages are stored in a temporary file and an editor is invoked to let you edit the file. The default editor is **ed**, but you can change this using the **editor** environment variable.

ex[it]
Quits **mailx** without changing the system *mailbox*. Contrast this with **quit**, which ordinarily removes from the system mailbox those messages you've read, saved, or deleted.

fi[le] [*filename***]**
Quits the system mailbox (as if a **q**[**uit**] subcommand were run) and then reads in the specified file as the new mailbox to examine. If no filename is specified, the default is your current mailbox.

Several special strings can be used in place of *filename*:

`%`	Your system mailbox.
`%user`	The system mailbox for user
`#`	The previous file
`&`	Your *mbox* (personal mailbox)
`+file`	The named file in the **folder** directory

fold[er] [*filename***]**
Is the same as the **file** subcommand.

folders
Displays the names of the files in the directory given by the **folder** variable. See “Environment Variables” on page 349.

F[ollowup] [*refs***]**
Replies to the first message given in *refs*; **mailx** sends this reply to the authors of every message given in *refs*. The `Subject` line is taken from the first message in *refs*. Your reply is automatically saved in a file which derives its name from the author of the message to which you are replying.

To create your reply, **mailx** puts you into input mode, where you can use all of the input mode commands.

fo[llowup] [*ref***]**
Replies to the specified message; if no message *ref* is given, you reply to the current message. Your reply is automatically saved in a file which derives its name from the author of the message to which you are replying. This overrides the **record** environment variable if **record** is set.

To create your reply, **mailx** puts you into input mode, where you can use all of the input mode commands.

f[rom] [*refs***]**
Displays the header summary for the specified messages. If *refs* is not given, the current message is used.

g[roup] [*alias* **[***name* **...]]**
Is the same as the **alias** command.

h[eaders] [*ref***]**
Displays the headers of a screenful of messages surrounding the message given by *ref*. The number of lines in a screen is given by the **screen** variable.

hel[p]
Displays a summary of the command-mode subcommands.

ho[ld] [*refs***]**
Retains the specified messages in your system mailbox. For example, you might decide to **ho**ld a message if you read it, but decide not to act upon it immediately. If *refs* is not specified, the current message is held. If any of the specified messages have been marked as deleted, the **hold** subcommand overrides that and still retains the messages. Subsequent **delete**, **dp**, and **dt** commands during the same **mailx** session can delete files marked for retention. See also **preserve** and the variables **hold** and **keepsave**.

i[f] *code mailx subcommands* **| [el[se]** *mailx subcommands***] | [en[dif]]**
Is primarily intended for use in startup files. The *code* must be the character `r` or `s`. If it is `r`, the first set of mailx subcommands are executed if **mailx** is in receive mode, and the second set if **mailx** is in send mode. If *code* is `s`, the opposite is true. The **else** part is optional. See “Startup Files” on page 348.

ig[nore] [*header* **...]**
Is the same as the **discard** subcommand.

l[ist]
Displays the names of all command-mode subcommands.

m[ail] *address* **...**
Sends a message to the specified recipients. **mailx** goes into input mode to let you enter the text of the message.

mb[ox] [*refs***]**
Indicates that the given messages are to be saved in your *mbox* (personal mailbox) when **mailx** quits normally (that is, through the **quit** command as opposed to **exit**).

n[ext] [*refs***]**
Goes to the next message in the mailbox that appears in the list of *refs*. For example:

```
n user
```

goes to the next message from the specified *user*.

pi[pe] [[*refs***] *command***]**
Pipes the messages given by *refs* through the specified shell *command*. These messages are considered read. If *refs* is not specified, the current message is used. If no *command* is specified, **mailx** uses the command specified by the **cmd** variable. See "Environment Variables" on page 349. If the **page** variable has a value, a form feed character is sent into the pipe after every message.

The subcommand | [*refs*] [*command*] is equivalent to **pipe**.

pre[serve] [*refs***]**
Is the same as the **hold** subcommand.

P[rint] [*refs***]**
Displays the specified messages on the screen. If *refs* is not specified, the current message is displayed. All header fields are displayed; the **discard** and **ignore** subcommands do not affect **Print**.

p[rint] [*refs***]**
Displays the specified messages on the screen. If *refs* is not specified, the current message is displayed. Header fields specified by **discard** and **ignore** subcommands are not displayed. If the **crt** variable is set to an integer, messages with more lines than that integer are "paginated" using the command specified by the **pager** variable. For more information, see "Environment Variables" on page 349.

q[uit]
Ends a **mailx** session. This is the usual method to leave **mailx**. Messages that have been read but not saved or deleted are stored in your *mbox* (personal mailbox). Messages that are still unread are retained in your system mailbox. Messages that have been deleted or explicitly saved in other files are discarded. Typing the end-of-file character has the same effect.

R[eply] [*refs***]**
Sends a reply to the authors of each of the messages specified by *refs*. If *refs* is not specified, the current message is used. The `Subject` line of the reply message is taken from the first message in *refs*. If the **record** environment variable is set to a filename, your reply message is appended to the end of that file.

Normally, you use **Reply** if you just want to send your reply to the author of a message, and **reply** if you want to send your reply to the author and all recipients. If set, the **flipr** environment variable reverses the meanings of the **R** and **r** commands. See "Environment Variables" on page 349.

r[eply] [*ref***]**
Sends a reply to the author of a specific message, and all other recipients of the message. If *ref* is not specified, **mailx** replies to the current message. If the **record** environment variable is set to a filename, your reply message is appended to the end of that file.

R[espond] [*refs***]**
Is the same as the **Reply** subcommand.

r[espond] [*ref***]**
Is the same as the **reply** subcommand.

ret[ain] [*header* **...]**
Is the opposite of the **discard** subcommand. It tells **mailx** to display the given *header* fields when displaying a message. The comparison of *header* fields is not case sensitive. You can use **retain** to override existing **discard** and **ignore** commands. If you do not specify any *header* fields, **retain** displays a list of currently retained header fields.

S[ave] [*refs***]**
Saves the specified messages in a file the name of which is taken from the author of the first message (the filename is the author's name, without any attached network addressing). If the **folder** variable is set, the file is saved to the specified directory.

s[ave] [*refs***][***filename***]**
Saves the specified messages in the given file. If *refs* is not given, the current message is added to the *mbox*. (The value of the **append** variable determines whether the message is added to the beginning or the end of the *mbox*). The file is created if it does not already exist. If you do not specify *filename*, **mailx** saves the messages in *mbox* (your personal mailbox). A message that has been saved with **s**ave is normally deleted from *mailbox* when *mailx* ends (see **quit**); but see the variables **hold** and **keepsave**.

se[t] *name*
Defines a variable with the given *name* and assigns it a null value. If you omit *name*, **set** displays a list of all defined variables and their values.

se[t] *name***=***value*
Defines a variable with the given *name* and assigns it the given *value*, which may be a string or a number.

se[t] no*name*
Is the same as the **unset** *name* subcommand.

sh[ell]
Invokes the shell given by the **SHELL** environment variable.

si[ze] [*refs***]**
Displays the size in bytes of each of the specified messages. If no *refs* are specified, the current message is used.

so[urce] *file*
Reads the specified text *file*, executes its contents as command-mode subcommands, and then returns to read more commands from the original source.

to[p] [*refs***]**
Displays the first few lines of each of the specified messages. If *refs* is not specified, the current message is used. If the **toplines** variable has a numeric value, that many lines are displayed from each message; otherwise, five lines are displayed from each message.

tou[ch] [*refs***]**
“Touches” the specified messages, making them appear to have been read. This means that when you **quit mailx**, the messages are saved in your *mbox*

(personal mailbox) if they are not deleted or explicitly saved in another file. If *refs* is not specified, the current message is touched.

T[ype] [*refs***]**
Is the same as the **Print** subcommand.

t[ype] [*refs***]**
Is the same as the **print** command.

una[lias] [*alias***[***name* **...]]**
Deletes specified alias names.

u[ndelete] [*refs***]**
Restores previously deleted messages. When messages are deleted, they are not discarded immediately; they are just marked for deletion and are actually deleted when **mailx** ends. Until **mailx** ends, you can use **undelete** to restore the specified messages. You cannot **undelete** messages deleted in previous sessions. If you do not specify *refs*, this command restores the first deleted (but not yet undeleted) message following the current message; if no such message exists, it restores the last deleted (but not yet undeleted) message preceding the current message. If the **autoprint** variable is set, the last restored message is displayed. This is the only subcommand that lets you give a *ref* to a message that has been deleted.

U[nread] [*refs***]**
Marks the specified messages as unread.

uns[et] *name* **...**
Discards the specified variables.

ve[rsion]
Displays version information about **mailx**.

v[isual] [*refs***]**
Edits the specified messages with a screen editor. If *refs* is not specified, the current message is edited. The messages are saved in a temporary file and the screen editor is invoked to edit that file. The editor used is given by the **VISUAL** variable. See “Environment Variables” on page 349.

w[rite] [*refs***]** *filename*
Writes the specified messages into the given file. If *refs* is not specified, the current message is written. **write** is the same as **save**, except that it does not write out the header lines and the blank line at the end of the message.

x[it] Is the same as the **exit** command.

z+ Scrolls the header display forward one screenful.

z– Scrolls the header display backward one screenful.

! *command*
Executes the given shell *command*. For example:

```
!lc
```

lists all files in the current directory. The shell that will be used to run the command is given by the **SHELL** environment variable. See “Environment Variables” on page 349.

#*comment*
Specifies that **mailx** should ignore everything from the **#** to the end of the line. This is useful for putting comments into startup files.

? Is the same as the **help** command (it displays a summary of the command-mode subcommands).

= Displays the current message number.

Input-Mode Subcommands

You can use input-mode subcommands when entering the text of a message. You must type mode subcommands at the beginning of an input line; you cannot type them in the middle of a line. By default, each input-mode subcommand begins with the tilde (~) character, called the *escape character.* You can use the **escape** variable to change the escape character, but in the documentation that follows, we always use tilde.

~. Marks the end of input in a mail message.

~? Displays a summary of the input-mode subcommands.

~A Inserts the autograph string at this point in the message. This autograph string is given by the **Sign** variable.

~a Is similar to **~A**, except that it uses the variable **sign**.

~b *name* **...**
Adds the specified names to the blind carbon copy list.

~c *name* **...**
Adds the specified names to the carbon copy list.

~d Reads in the **dead.letter** file.

~e Invokes an editor on the message that you have composed. The **editor** variable determines the editor that is invoked.

~F [*refs</ pv>]*
"Forwards" the given messages. The text of the messages is inserted at this point in the message you are composing. The message headers are also inserted with all header fields regardless of the **discard**, **ignore**, and **retain** subcommands. This is valid only when you entered **mailx** in command mode and then went into input mode to compose a message.

~f[*refs***]**
Is similar to ~F except that the header fields included are determined by the **discard**, **ignore**, and **retain** subcommands.

~h Prompts you to enter the following header lines:

```
Subject   Cc   Bcc   To
```

For some of these, **mailx** displays an initial value for the header. You can edit this initial value as if you had just typed it in yourself, using backspaces and line deletes.

~i *name*
Inserts the value of the named variable followed by a newline at this point in the message.

~M[*refs***]**
Inserts the text of the specified messages at this point in the message. If *refs* is not specified, the current message is used. Messages inserted in this way have each line prefixed with the value of the **indentprefix** variable. The message headers are also inserted with all header fields included regardless of the **discard**, **ignore**, and **retain** subcommands. This is valid only when you

entered **mailx** in command mode and then went into input mode to reply to a message.

~m< Is similar to ~M, except that the header fields are determi ned by the **discard**, **ignore**, and **retain** subcommands.

~p Displays the message being composed.

~q Quits input mode as if you had interrupted the message. If you have already composed part of a message, the partial message is saved in the **dead.letter** file; the description of the **dead** environment variable has more information..

~r *filename*
Reads in the contents of the specified file and adds that text at this point in the message.

~s *text*
Sets the `Subject` line to the given *text*.

~t *address address ...*
Adds the given addresses to the `To:` list (people who will receive the message).

~v Invokes a screen (visual) editor on the message that you have composed. The **VISUAL** variable determines the editor that is invoked.

~w *file*
Writes the current text of your message to the specified *file*. The header lines for the message are not written.

~x Quits in the same way as ~q, except that the message is not saved in the **dead.letter** file.

~< *filename*
Is the same as the ~r command.

~< **!***command*
Runs the given shell *command* and adds the standard output of that command at this point in the message. For example, your message might contain:

```
My program is giving me this odd output:
~< !prog
What do you think is causing it?
```

~: *mail_command*
Runs the given command-mode *mail_command*. This is valid only when you entered **mailx** in command mode and then went into input mode to compose a message.

~_ *mail_command</ pv>*
Is the same as the **~:** command.

~! *command*
Runs the given shell *command*. For example, you can use:

```
>~! ls
```

to get a list of files in the working directory. The shell that is invoked to run the command is given by the **SHELL** environment variable. If the **bang** variable is set, **mailx** replaces each unescaped exclamation mark (!) in *command* with the command run by the previous command or ~! command escape.

~ Marks the end of input in a mail message.

~| *command*
Pipes the current message through the specified shell *command*. If the *command* ends with a successful exit status, the output of the command replaces the text of the current message. For example:

```
~|fmt
```

fills and justifies the lines of your message and replaces the message with the formatted message. ~| uses the shell given by the **SHELL** environment variable to run *command*.

Startup Files

When you run **mailx** in command mode, **mailx** does the following:

- Sets all variables to their default values. **mailx** processes command-line options, using them to override any corresponding default values.
- Imports appropriate external environment variables, using them to override any corresponding default values.
- Reads commands from the system startup file, **/etc/mailx.rc**. This sets up variable values and definitions that should be common to all users. If you do not want **mailx** to read the system startup file, use the **–n** option on the **mailx** command line.
- After reading and processing the system startup file, **mailx** does the same with a personal startup file, which is **MAILRC** by default. This is a file in your **HOME** directory. The name of the file is **.mailrc.**

Startup files typically set up display options and define aliases. However, any command is valid in a startup file except for the following:

Copy
edit
followup
Followup
hold
mail
preserve
reply
Reply
respond
Respond
shell
visual
!

If a line in a startup file contains an error or an incorrect command, the rest of the startup file is ignored. **mailx** ignores blank lines in a startup file.

Example

The following example composes and sends a message to several users. Items shown in italics are output by **mailx** itself.

```
mailx JEAN
Subject: Greetings
This is just a short note to say hello.
~c JUAN JOHN JOHANN
~.
```

On the first line, the message is just addressed to `jean`. The **~c** line adds more people who will receive copies of the message.

Environment Variables

A large number of variables are used to control the behavior of **mailx**. These environment variables are divided into two classes: those that always come from the external environment, and those that may be set up in either the external environment or within a **mailx** session.

The following variables always come from the external environment; they can be changed inside a **mailx** session, except where marked.

HOME
Gives the name of your home directory. This cannot be changed inside **mailx** .

LOGNAME
Gives your login name.

MAIL
Gives the pathname of the user's mailbox file for purposes of incoming mail notification.

MAILDIR
Gives the name of the directory where system mailboxes are stored. If this is not set, the default is **/usr/mail**. The actual name of a user's system mailbox is derived in a system-dependent way by combining **MAILDIR** and the user's login name. For **mailx** to work properly, the **MAILDIR** directory must exist.

MAILRC
Gives the name of your startup file. This cannot be changed inside **mailx**. By default, **MAILRC** has the value **$HOME**`/.mailrc`. For more information about startup files, see “Startup Files” on page 348.

The **HOME** and **LOGNAME** variables must be set before you enter **mailx**. Otherwise, **mailx** will not work properly. These variables are set automatically for you if you enter the shell using the TSO/E OMVS command. If you do not log in, you must set the variables in some other way, using the commands:

```
export LOGNAME=name
export HOME=directory
```

The remaining variables can be set in the external environment or in the course of a **mailx** session. You can set or change the value of a variable with the **set** subcommand; you can discard a variable with the **unset** subcommand. You may find it convenient to create a startup file that sets these variables according to your preferences; this eliminates the need to set variables each time you enter **mailx**.

Many of the following variables represent on-off options. If you set the variable itself (to any value), the option is turned on. To turn the option off, you can unset the variable, or set a variable consisting of *no* followed by the name of the original variable. For example, setting **autoprint** turns the autoprint option on, and setting **noautoprint** turns it off.

allnet
Assumes that network addresses with the same login component refer to the same person. Network addresses typically consist of several components, giving information that lets a mail server identify a machine on the network, a route to that machine, and the login name of a user on that machine. **mailx** assumes that the login name is the last component. For example:

```
print name
```

displays all messages that originated from the same login `name`, regardless of the rest of the network address. The default is **noallnet**, where different addresses are assumed to be different users, even if the login name components are the same.

append
Appends messages to the end of the *mbox* file (your personal mailbox) after termination. The default is **noappend**; messages are placed at the beginning of the *mbox* file instead of the end.

ask Prompts you for a `Subject:` line when composing a message (if you have not already specified one with the **–s** option). This option is on by default; to turn it off, set *noask*. *ask* is the same as **asksub**. **noask** is the same as **noasksub**.

askbcc
Prompts you for a `Bcc:` list when composing a message. The default is **noaskbcc**; you are not prompted.

askcc
Prompts you for a `Cc:` list when composing a message. The default is **noaskcc**; you are not prompted.

asksub
Prompts you for a `Subject:` line when composing a message (if you have not already specified one with the **–s** option). This option is turned on by default; to turn it off, set **noasksub**. **asksub** is the same as **ask**. **noasksub** is the same as **noask**.

autoprint
Automatically displays the last message deleted with the **delete** subcommand or the last message undeleted with **undelete**. The default is **noautoprint**; you are not shown messages that you delete or undelete.

bang
Records shell commands run inside the **mailx** session (for example, through the **~!** input-mode command). Then, if you issue a shell command and the shell command contains a `!` character, **mailx** replaces that character with the command line for the previous shell command. The default is **nobang**, in which case a `!` in a shell command line is not treated specially.

cmd
Contains a command, possibly with options. This specifies a default command line to be used for the command-mode **pipe** subcommand. For example:

```
set cmd="cat"
```

pipes messages through **cat** when the **pipe** subcommand is invoked.

crt Contains an integer number. If a message has more than this number of lines, the message is piped through the command given by the **PAGER** variable, whenever the message is displayed. **crt** is not set; the default is **nocrt**.

dead
Contains the name of a file that can be used as the **dead.letter** file. Partial messages are saved in this file if an interrupt or error occurs during creation of the message or delivery. By default, the name of this file is **$HOME/dead.letter**.

dot Accepts a line consisting only of a dot (**.**) to indicate the end of a message in input mode. Thus **.** is equivalent to ~.. The default is **nodot**. If **ignoreeof** is set, **mailx** ignores a setting of **nodot**; the period is the only way to end input mode.

editor
Gives a command, possibly with options, that is run when using the command mode **edit** or the input mode ~e. The default is **ed** (see **ed**).

escape
Gives the character used to begin input-mode subcommands. The default is the tilde (~). If this variable is set to null, **mailx** disables command escaping.

flipr Reverses the meanings of the **R** and **r** subcommands. The default is **noflipr**. See also **Replyall**.

folder
Contains the name of a directory in which **mailx** saves mail files. This lets you specify a standard directory for saving mail files. Whenever you specify a filename for a **mailx** command, putting a plus sign (+) in front of the name specifies that the file is to be accessed in the **folder** directory.

If the value of **folder** begins with a slash, it is taken as an absolute pathname; otherwise, **mailx** assumes that the directory is directly under your **HOME** directory. **folder** has no default value. If you want to use + in filenames that appear on the **mailx** command line itself (as opposed to commands in a **mailx** session), you must make **folder** an exported shell environment variable.

header
Displays a summary of message headers at the beginning of a **mailx** command-mode session. This is the default.

hold
Keeps all messages in your system mailbox instead of saving them in your personal *mbox*. The default is **nohold**.

ignore
Ignores interrupts received while composing a message. The default is **noignore**.

ignoreeof
Ignores end-of-file markers found while entering a message. The message can be ended by "**.**" or **~.** on a line by itself. The default is **noignoreeof**.

indent
Contains a string that **mailx** uses as a prefix to each line in messages that ~m and ~M insert. The default is one tab character.

indentprefix
As with **indent**, contains a string that **mailx** uses as a prefix to each line in messages that ~m and ~M insert. The default is one tab character. If both **indent** and **indentprefix** are set, **indentprefix** takes precedence.

keep
Does not remove your system mailbox if the mailbox contains no messages. The mailbox is truncated to zero length—that is, it is merely emptied, although it still exists. If the default **nokeep** is in effect, empty mailboxes are removed.

keepsave
Keeps messages in your system mailbox even if they have been saved in other files. The default, **nokeepsave**, deletes messages from the system mailbox if they have been saved elsewhere.

lister
Contains a command, possibly with options. **mailx** invokes this command when displaying the contents of the **folder** directory for the **folders** subcommand. If this variable is null or unset, **mailx** uses **ls**. By default, this variable is unset.

mailrc
Is the location of personal startup file. See “Startup Files” on page 348.

mailserv
Identifies the mail server being used for remote mail.

mbox
Gives the name of your *mbox* (personal mailbox) file. Messages that have been read but not saved elsewhere are saved here when you run **quit** (but not when you run **exit**). The default is **$HOME/mbox**.

metoo
When replying to a message with your login name in the recipient list, sends a reply to all other recipients, the author, and you. If **nometoo** is set, you are not to be sent the reply. The default is **nometoo**.

onehop
Attempts to send replies directly to the recipients instead of going through the original author's machine. When you reply to a message, your reply is sent to the author and to all recipients of the message. On a network, **mailx** normally specifies the recipient addresses so that all the replies go to the original author's machine first, and then on to the other recipients.

outfolder
Causes files used to record outgoing messages (see the description of **record**) to be located in the directory given by folder unless folder contains an absolute pathname.

The default is **nooutfolder**.

page
Tells the **pipe** subcommand to insert a form-feed character after each message that it sends through the pipe. The default is **nopage**.

pager
Contains a command, possibly including options. **mailx** sends display output through this command if the output is longer than the screen length given by **crt**. The default value is `more` (see **more**).

prompt
Contains a string that **mailx** displays to prompt for output in command mode. The default is a question mark followed by a space (`? `).

quiet
Does not display the opening message and version number when **mailx** begins a session. The default is **noquiet**.

record
Contains a filename where every message you send is to be recorded. If **record** is not an absolute pathname and the **outfolder** variable has not been set, the file is located in the current directory. If the **outfolder** variable is set, the file is located in your **folder** directory. The default is **norecord**.

replyall
Reverses the senses of the **reply** and **Reply** subcommands (so that **reply** replies only to the author of a message, and **Reply** replies to the author and all other recipients). See also **flipr**.

save
Saves messages in your **dead.letter** file if they are interrupted while being composed. The name of your **dead.letter** file is given by the **dead** variable. Setting **nosave** disables this automatic save feature. The default is **save**.

screen
Gives the number of headers that are to be displayed by the **headers** and **z** subcommands.

sendmail
Contains a command, possibly with options, that **mailx** invokes to send mail. The default is **/usr/lib/tsmail**. It can be any command that takes addresses on the command line and message contents on standard input.

sendwait
When sending a message through a network, **mailx** waits for the mail server to finish before returning to your session. Normally, it just submits the message to the server and then returns immediately. The default is **nosendwait**.

shell
Contains a command, possibly with options. **mailx** assumes that this command is a command interpreter. **mailx** invokes this command interpreter whenever it is asked to run a system command (for example, through the ! command-mode command). The default is **sh** (see **sh**).

showto
When displaying a header summary, displays the recipient's name instead of the author's for messages where you are the author. The default is **noshowto**.

sign
Contains a string that is inserted into a message when you use the input mode **~a** subcommand. **mailx** interprets *\n* and *\t* in this string as the newline and tab characters, respectively. The default is **nosign**.

Ssign
Contains a string that is inserted into a message when you use the input mode ~**A** subcommand. The default is **noSign**.

term
Contains the name of the terminal type. If **screen** is not set, **term** individually determines the number of lines in a screenful of headers.

toplines
Gives the number of header lines that the **top** subcommand is to display. The default is 5.

VISUAL
Contains a command, possibly with options, that **mailx** invokes when using the command-mode **visual** subcommand or the input mode ~v subcommand. The default is **vi** (see **vi**).

Files

mailx uses the following files:

/etc/mailx.rc	Systemwide startup file.
$MAILRC	Personal startup file. By default, **MAILRC** has the value **$HOME/.mailrc**.
$HOME/mbox	Default location to save read messages. You can choose a different file by assigning the filename to the environment variable **MBOX.**
$MAILDIR	Directory containing system mailboxes. By default, this is **/usr/mail**. The system programmer must create the **MAILDIR** directory if it does not already exist. See *OS/390 UNIX System Services Planning* for information on creating the **MAILDIR** directory.

$HOME/dead.letter Default location to save partial letters.

Localization

mailx uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_TIME**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Values

`0` Successfully sent. (However, this does not guarantee that the mail was successfully received). `0` is also returned if **–e** is specified and there is no new mail. `0` is returned if there is new or unread mail. `1` means that there is no new or unread mail.

1 Returned if **–e** is specified and there is new mail. Also returned to indicate failure due to any of the following:

- There is no mail to read.
- Inability to create temporary file name or temporary file.
- Receipt of user interrupt while message was being composed.
- Inability to determine the user's identity.

2 Failure due to any of the following:

- Missing *number* after **–h**
- Missing *address* after **–r**
- Missing *subject* after **–s**
- Missing *user* after **–u**
- Incorrect command-line option
- Use of interactive options when not using command interactively

Portability

POSIX.2, X/Open Portability Guide, UNIX System V.

UNIX System V has a compatible **mailx** utility, whereas Berkeley Software Distribution (BSD) has a similar utility, known as **Mail**.

The **–F**, **–r**, and **–U** options; the **Copy**, **echo**, **followup**, **Followup**, **Save**, **Unread**, and **version** subcommands; and the **allnet**, **conv**, **mailserv**, **onehop**, **replyall**, **sendmail**, and **sendwait** variables are extensions of the POSIX standard.

Related Information

echo, **ed**, **sh**, **vi**

---

make — Maintain program-generated and interdependent files

Format

make [**–EeinpqrstuVvx**] [**–kl–S**] [**–c** *dir*] [**–f** *file*] ...
[*macro definition* ...] [**–D** *macro definition* ...] [*target* ...]

Description

make helps you manage projects containing a set of interdependent files, such as a program with many source and object files, or a document built from source files, macro files, and so on. **make** keeps all such files up to date with one another. If one file changes, **make** updates all the other files that depend on the changed file.

Note: This implementation of **make** features the `.POSIX` special target to provide maximum portability. When you specify this target, **make** processes the makefile as specified in the POSIX standard. For details, see the description of `.POSIX` in “Special Target Directives” on page 368.

In a doublebyte locale, environment variable values, here documents, and the command line may all contain environment values.

Options

–c *dir*
Attempts to change into the specified directory when **make** starts up. If **make** cannot change to the directory, an error message is printed. This is useful for recursive makefiles when building in a different directory.

–D *macro definition*
Define *macro* on the command line before reading any *makefile.* Use the same form as a normal macro definition (*macro=string*). If you use this option, **make** assigns the value to the macro before reading the makefile; any definition of the same macro contained in the makefile supersedes this definition.

Note: **make** uses any macros defined in this way before reading any makefile, including the startup file. This allows you to define a startup file by providing a value for **MAKESTARTUP** on the command line:

```
make –D MAKESTARTUP=$HOME/project/startup.mk
```

–E Suppresses reading of the environment. If you do not specify either **–E** or **–e**, **make** reads the environment before reading the makefile.

–e Reads the environment *after* reading the makefile. If neither **–E** nor **–e** are specified, **make** reads the environment *before* reading the makefile, except for the **SHELL** environment variable, which you must explicitly export. This option does not affect the value of **MAKEFLAGS**.

–f *file*
Uses *file* as the source for the makefile description. **make** ignores the makefiles specified as prerequisites to the `.MAKEFILES` target. If you specify a minus sign (–) in place of *file*, **make** reads the standard input. (In other words, **make** expects you to enter the makefile from the terminal or to redirect it from a file.) You can use more than one **–f** option.

–i Tells **make** to ignore all errors and continue making other targets. This is equivalent to the `.IGNORE` attribute or macro.

–k Makes all independent targets, even if an error occurs. Ordinarily, **make** stops after a command returns a nonzero status. Specifying **–k** tells **make** to ignore the error and continue to make other targets, as long as they are not related to the target that received the error. **make** does not attempt to update anything that depends on the target that was being made when the error occurred.

–n Displays the commands that **make** would execute to update the chosen targets, but does not actually execute any recipe lines unless they have a plus sign (+) command prefix. **make** displays recipe lines with an at sign (@) command prefix on standard output (**stdout**). For more information about recipe lines, see *OS/390 UNIX System Services Programming Tools*.

With group recipes, **make** displays the commands it uses to update a given target, but it also executes the commands.

Note: OS/390 **make** supports *group recipes*, but traditional implementations of **make** do not. A group recipe signifies a collection of command lines fed as a unit to the command interpreter. By contrast, **make** executes commands in a normal recipe one by one. For more information about group recipes, see *OS/390 UNIX System Services Programming Tools*.

If **make** finds the string $ (MAKE) in a recipe line, it expands it, adds **–n** to the MAKEFLAGS, and then executes the recipe line. This enables you to see what happens when recursive calls are made to **make**. The output correctly shows line breaks in recipes that are divided into several lines of text using the \<newline> sequence.

–p Prints the makefile after it has been processed to include macro and target definitions. This display is in human-readable form useful for debugging, but you cannot use it as input to **make**.

–q Checks whether the target is up to date. If it is up to date, **make** exits with a status of 0; otherwise, it exits with a status of 1 (typically interpreted as an error by other software). No commands are run when **–q** is specified.

–r Does not read the startup file. Various control macros and default rules will not be defined.

–S Ends **make** if an error occurs during operations to bring a target up to date (opposite of **–k**). This is the default.

–s Specifies that recipe commands, warning messages, or touch messages (see the **–t** option) not be displayed. This is equivalent to the .SILENT attribute or macro.

–t Touches the target to mark them as up-to-date, but only executes commands to change a target if the target has a plus sign (+) command prefix. **make** does not touch up-to-date targets or targets that have prerequisites but not recipes. **make** displays a message for each touched target file indicating the filename.

–u Forces an unconditional update: **make** behaves as if all the prerequisites of the given target are out of date.

–V Prints the version number of **make** and a list of built-in rules.

–v Causes **make** to display a detailed account of its progress. This includes what files it reads, the definition and redefinition of each macro, metarule and suffix rule searches, and other information.

–x Exports all macro definitions to the environment. This happens just before **make** begins making targets (but after it has read the entire makefile).

Targets

A *target* is normally a file that you want to ensure is up to date with the files on which it is dependent (the prerequisites). For example, you may want to check to see if *a* is based on the most recent version of the corresponding source code. If it is not, then have the source code recompiled to get an up-to-date version. In this case, the compiled program file is the target and the corresponding source files are *prerequisites* (that is, the files on which a target is dependent).

make updates all targets that are specified on the command line. If you do not specify any target, **make** updates the targets in the first rule of the makefile. A target is out of date if it is older than any of its prerequisites (based on modification times) or if it does not exist. To update a target, **make** first recursively ensures that all the target's prerequisites are up to date, processing them in the order in which they appear in the rule. If the target itself is out of date, **make** then runs the recipe associated with the target. If the target has no associated recipe, **make** considers it up to date.

make also supports another form of targets, known as *special targets,* described in "Special Target Directives" on page 368.

Makefiles

A **makefile** is a text file that describes the dependencies between various files. It normally contains a list of targets and identifies the prerequisites on which each depends. It also contains a series of instructions, called *recipes*, which describe the actions to be taken if a given target is out of date with its prerequisites.

By default, if you do not specify the **–f** option, **make** looks for a file in your current directory named **makefile**. If it does not find this file, it searches your current directory for a file named **Makefile**. If **make** finds either file, it uses this file as your makefile.

You can change the default makefiles with the `.MAKEFILES` special target. This target is already specified in the **startup.mk** file. See "Special Target Directives" on page 368 for more information.

Macro Definitions

Macro definitions can appear on the command line or in makefiles. Macro definitions on the command line overrule definitions in makefiles; makefile definitions never overrule command-line definitions. Macro definitions on the command line may not have any white space between the macro name and the `=` character.

Macro definitions may take several forms.

macro = string

is the usual form. If *string* contains macro references, **make** does not expand them when the macro is defined, but when the macro itself is expanded.

macro **:=** *string*

expands macros inside *string* before assigning a value to *macro*.

macro **+=** *string*

adds *string* to the previous value of *macro*.

You can use any amount of white space on both sides of macro operators. **make** defines the name *macro* to have the value *string* and replaces it with that value whenever it is used as `$(`*macro*`)` or `${`*macro*`}` within the makefile. It is possible to specify a `$(`*macro_name*`)` or `${`*macro_name*`}` macro expansion, where *macro_name* contains more `$(....)` or `${...}` macro expansions itself.

Normally, **make** does not include white space at the beginning and end of *string* in the definition of *macro*; however, it never strips white space from macros imported from the environment.

If you want to include white space in a macro definition specified on the **make** command line, you must enclose the definition in quotes.

make resolves macro definitions in the following order:

1. Macro definitions in the built-in rules
2. Macro definitions on the command line associated with the **–D** option
3. Macro definitions in the startup file

4. Contents of the environment
5. Macro definitions in the makefiles (in the order they appear)
6. Macro definitions on the command line without the **–D** option

Note: If you specify the **–e** options, **make** reads the makefiles before reading the contents of the environment. If you specify the **–E** option, **make** does not read the contents of the environment.

If a macro is already defined when **make** encounters a new definition for it, the new definition replaces the old one. For example, a macro definition for *name* on the command line overrides a definition for *name* in the makefile. You can use the **–v** option to display macro assignments, as **make** performs them.

Macro Modifiers

make supports macro expansions of the form:

```
$(macro_name:modifier_list:modifier_list:...)
```

Possible modifiers are:

^"*string*"	Prefix tokens
+"*string*"	Suffix tokens
`b`	File portion of all pathnames, without suffix
`d`	Directory portion of all pathnames
`f`	File portion of all pathnames, including suffix
`l`	All characters mapped to lowercase
`s/`*pat*`/`*string*`/`	Simple pattern substitution (you can use any character to separate the pattern from the substitution text)
`suffix=`*string*	Suffix replacement
`t"`*separator*`"`	Tokenization with given *separator*
`u`	All characters mapped to uppercase

You can specify macro modifiers in either uppercase or lowercase. For example, the macro assignment:

```
test = D1/D2/d3/a.out f.out d1/k.out
```

produces the following expansion:

```
$(test:d)                → D1/D2/d3 . d1
$(test:b)                → a f k
$(test:f)                → a.out f.out k.out
${test:db}               → D1/D2/d3/a f d1/k
${test:s/out/in}         → D1/D2/d3/a.in f.in d1/k.in
$(test:f:t"+")           → a.out+f.out+k.out
$(test:t"+")             → D1/D2/d3/a.out+f.out+d1/k.out
$(test:u)                → D1/D2/D3/A.OUT F.OUT D1/K.OUT
$(test:l)                → d1/d2/d3/a.out f.out d1/k.out
$(test:^"/rd/")          → /rd/D1/D2/d3/a.out /rd/f.out /rd/d1/k.out
$(test:+".Z")            → D1/D2/d3/a.out.Z f.out.Z d1/k.out.Z
```

Runtime Macros

Runtime macros can take on different values for each target.

`$@` The full target name. When building a normal target, this macro evaluates to the full name of the target. When building a library, it expands to the name of the archive library. For example, if the target is:

```
mylib(member)
```

`$@` expands to:

```
mylib
```

`$%` The full target name. When building a normal target, this macro evaluates to the full name of the target. When building a library, it expands to the name of the archive member. For example, if the target is:

```
mylib(member)
```

`$%` expands to:

```
member
```

`$&` The list of all prerequisites.

`$?` The list of all prerequisites that are newer than the target.

`$^` The list of all prerequisites taken from the list specified on the rule line of the recipe where the `$^` appears.

`$<` In inference rules, it evaluates to the single prerequisite that caused the execution of the rule. In normal rules it evaluates the same as `$?`.

`$>` The name of the library if the current target is a library member.

`$*` The target name with no suffix (`$(%:db)`) or the value of the stem in a metarule.

The constructs `$$@`, `$$%`, `$$>`, and `$$*` can appear in a prerequisite list as dynamic prerequisites. `$$@` stands for the target currently being made. For example:

```
fred : $$@.c
fred : fred.c
```

are equivalent. The construct can be modified, as in:

```
fred.o : $$(@:b).c
```

The runtime macros can be modified by the letters `D` and `F` to indicate only the directory portion of the target name or only the file portion of the target name. (The working directory is represented by a dot.) If **define.h** is the only prerequisite that is newer than the target, the macros `$?D` and `$?F` expand to dot (`.`) and to `define.h`.

If you are building a library, `$$%` stands for the name of the archive member being made. If you are building a normal target, `$$%` stands for the name of the target currently being made.

`$$*` stands for the name of the current target being made, but with no suffix.

If you are building a library, `$$>` stands for the name of the archive library being made. If you are not building a library, `$$>` is not valid.

Usage Note

Comments begin with the pound (#) character and extend to the end of the line. **make** discards all comment text.

Makefile Contents

Inside makefiles, you can split long lines over several lines of text. To do this, put a backslash (\) at the very end of the line. You can use this technique to extend comments as well as recipe lines and macro definitions, for example.

If a rule or macro definition must contain a # character, use \#; otherwise, **make** mistakes the # for the beginning of a comment. Also, if a macro definition must contain a single $ character, use $$.

Filenames that contain a colon must always be enclosed in quotes, as in:

```
"a:target" : "a:prereq"
```

Rules

The general format of a rule is:

```
targets [attributes] ruleop [prerequisites]
[;recipe]
{<tab> recipe}
```

where the items enclosed in square brackets are optional. (This is just a documentation convention; you do not actually enter the square brackets.) The parts of the rule are described as follows:

targets One or more target names.

attributes A list, possibly empty, of attributes to apply to the list of targets.

ruleop An operator token, usually a colon (**:**), that separates the target names from the prerequisite names and may also affect the processing of the specified targets.

prerequisites
A list of zero or more names on which the specified targets depend.

recipe A command to execute to update targets. May follow on the same line as the prerequisites, separated from them by a semicolon. If such a recipe is present, **make** takes it as the first in the list of recipe lines defining how to make the named targets. Additional recipe lines may follow the first line of the rule. Each subsequent recipe line must begin with a tab character.

The possible rule operators are listed as follows:

`targets : prereqs`
Is a simple rule definition. For explicit targets, at most one simple rule may have a recipe, in contrast with the **::** rule operator, whose description follows.

`targets :! prereqs`
Executes the recipe for the associated targets once for each recently changed prerequisite. In simple rules, the recipe is executed only once, for all recently changed prerequisites at the same

time. The $< macro expands to the current recently changed prerequisites if it appears in rules with this rule operator.

`targets :^ prereqs`
Inserts the specified prerequisites before any other prerequisites already associated with the specified targets.

`targets :– prereqs`
Clears the previous list of prerequisites before adding the new prerequisites.

`targets :: prereqs`
If no prerequisites are specified, the targets are always remade. Otherwise it is used for multiple rules applying to the same targets. Each rule can specify a different set of prerequisites with a different recipe for updating the target. Each rule is treated independently; the target is remade for each rule with recently changed prerequisites, using the corresponding recipe.

`targets :| prereqs`
Can only be used in metarules. It tells **make** to treat each metadependency as an independent rule. For example:

```
%$0 :| archive/%.c rcs/%.c /srcarc/RCS/%period.c
recipe...
```

is equivalent to

```
%$0 : archive/$.c
  recipe:
%$0 : rcs/%.c
  recipe:
%$0 : /srcarc/rcs/%.c
  recipe:
```

Circular Dependencies

There are two types of circular dependencies: within-rule and between-rule.

A *within-rule* circular dependency occurs when the target's name is included in the list of prerequisites for that target. For example,

```
c.o : a.o b.o c.o
```

is a within-rule circular dependency. **make** detects a within-rule circular dependency when it is parsing the makefile to build the dependency tree.

A *between-rule circular dependency* occurs when you have two targets, each of which includes the other's name in its prerequisite list. For example,

```
a.o : b:o
b:o : a.o
```

is a between-rules circular dependency. **make** detects a between-rule circular dependency when it is processing the dependency tree built during the parse phase.

Normally **make** only detects circular dependencies for those targets actually being built. When a circular dependency is encountered, **make** issues a warning message, removes the offending prerequisite from the list, and continues parsing the makefile. You can use the `.CYCLECHECK` special target to alter. **make**'s treatment

of circular dependencies. See also “Special Target Directives” on page 368, which describes special targets.)

Recipes

You can use a target that has prerequisites but no recipes to add the given prerequisites to that target's list of prerequisites.

You can preface any recipe line with a command prefix immediately after the tab character –, `@`, + or all three). The method of entering tab characters using an ISPF editor is discussed in *OS/390 UNIX System Services User's Guide*.

– indicates that **make** is to ignore nonzero exit values when it runs this recipe line.
`@` indicates that **make** is not to display the recipe line before running it.
+ tells **make** to always run this line, even when **–n**, **–p**, or **–t** is specified.

Group recipes begin with [in the first non-white-space position of a line, and end with] in the first non-white-space position of a line. Recipe lines in a group recipe need not have a leading tab. **make** executes a group recipe by feeding it as a single unit to a shell. If you immediately follow the [at the beginning of a group recipe with one of –, `@` or +, they apply to the entire group in the same way that they apply to single recipe lines.

Inference Rules

With inference rules you can specify general rules for building files rather than creating a specific rule for each target.

make provides two forms of inference rules: suffix rules and metarules. It includes suffix rules to ensure compatibility with older makefiles. Metarules, however, provide a more general mechanism for specifying **make**'s default behavior. They provide a superset of the functionality of suffix rules.

make searches all metarules before using suffix rules.

make uses the inference rules to infer how it can bring a target up to date. A list of inference rules defines the commands to be run. The default **startup.mk** file contains a set of inference rules for the most common targets. You can specify additional rules in the makefile.

When **make** finds no explicit target rule to update a target, it checks the inference rules. If **make** finds an applicable inference rule with an out-of-date prerequisite, it runs on that rule's recipe. See “Special Target Directives” on page 368, which describes the `.DEFAULT` special target).

Metarules

Metarules have one target with a single percent symbol that matches an arbitrary string called the stem; The `%` in a dependency stands for the stem.

The inference rule to update a target matching pattern *p1%s1*, where *p1* and *s1* are prefix and suffix strings of the target, having a prerequisite *p2%s2*, where `%` is the stem from the target, is specified as a rule:

```
p1%s1 : p2%s2 ; recipe....
```

Either the prefix or suffix string may be empty.

Transitive Closure

Metarules provide a mechanism that allows several metarules to chain together to eventually create the target.

This is called *transitive closure*. For example, if you have metarules:

```
%.o : %.c
        ... rule body....
```

and:

```
%.c : %.y
        ... rule body ...
```

c When you specify:

```
make file.o
```

make uses the first metarule to look for **file.c**. If it can't find an explicit rule to build **file.c**, it again looks through the metarules and finds the rule that tells it to look for **file.y**.

make allows each metarule to be applied only once when performing transitive closure to avoid a situation where it loops forever. (For example, if you have the rule:

```
% : %.c
        ... rule body ...
```

the command:

```
make file
```

causes **make** to look for **file.c**. If the metarules were not restricted and **file.c** did not exist, then **make** would look for **file.c.c**, and then **file.c.c.c**, and so on. Because each metarule is applied only once, this can't happen.)

Transitive closure is computed once for each metarule head the first time the pattern matches a target. When transitive closure is computed, all the computed rules are added to the rule set for that metarule head. For example, if you have the rules:

```
% : %.o
        recipe 1...
%.o : %c
        recipe 2...
```

and you are making *file*, this target matches successfully against `%` causing transitive closure to be computed for `%`. As a result of this computation, a new rule is created:

```
% : %.c
   recipe 2...
   recipe from .REMOVE target for %.o, if not .PRECIOUS
   recipe 1...
```

which is executed if **file.o** doesn't exist. When the computation for the rule head has been done, it is marked as *transitive closure computed*. Since all possible new rules have been added to the rule set the first time the computation is done, it is

not necessary to do it again: Nothing new is added. The term *transitive closure* is adapted from the mathematical set theory.

Note: In set theory, if you have a set composed of pairs (*a,b*) and (*b,c*), then the set would be transitively closed if (*a,c*) is also in the set.

The best way to understand how this works is to experiment with little **make** files with the **–v** flag specified. This shows you in detail what rules are being searched, when transitive closure is calculated, and what rules are added.

Order of Rule Generation

Since transitive closure allows **make** to generate new rules, it is important to understand the order in which this is done:

1. **make** searches for explicit rules in the order in which they appear, so explicit rules always take precedence.
2. **make** reads metarules in the order in which they appear in the makefile. The first rule that appears in the makefile is the first one checked.
3. New explicit metarules (as distinct from metarules generated by transitive closure) replace old ones. In other words, if your makefile contains an explicit rule like this one, it replaces the default rule in **startup.mk**:

   ```
   %$0 : %.c
           rule1
   ```

   If you use the **–v** option, **make** prints a warning when it replaces a metarule.
4. When transitive closure is calculated, the new metarules generated are added to the end of the list of possible metarules. Thus, **make** always finds the explicit rules first, so they take precedence over generated rules. You can use the **–v** option to see what rules **make** generates and the order in which they appear.
5. **make** performs two passes through the rules. On the first pass it tries to find a match with an explicit rule in the makefile; if this does not succeed, **make** performs a second pass to find a match with an existing file.

Suffix Rules

make treats targets that begin with a period and contain no slashes or percent signs as suffix rules. If there is only one period in the target, it is a single suffix inference rule. Targets with two periods are double-suffix inference rules. Suffix rules do not have prerequisites but do have commands associated with them.

When **make** finds no explicit rule to update a target, it checks the suffix of the target (*.s1*) to be built against the suffix rules. **make** examines a prerequisite based on the basename of the target with the second suffix (*.s2*) appended, and if the target is out of date with respect to this prerequisite, **make** runs the recipe for that inference rule.

Metarules take precedence over suffix rules.

If the target to be built does not contain a suffix and there is no rule for the target, **make** checks the single suffix inference rules. The single suffix inference rules define how to build a target if **make** finds a rule with one of the single suffixes appended. A rule with one suffix *.s2* defines how to build *target* from *target.s2*. **make** treats the other suffix (*.s1*) as null.

For a suffix rule to work, the component suffixes must appear in the prerequisite list of the .SUFFIXES special target. You can turn off suffix rules by placing the following in your makefile:

```
.SUFFIXES:
```

This clears the prerequisites of the .SUFFIXES target, which prevents suffix rules from being enacted. The order that the suffixes appear in the .SUFFIXES rule determines the order in which **make** checks the suffix rules.

The search algorithm used for suffix rules depends on whether the .POSIX special target is specified. When .POSIX is specified, the following steps describe the search algorithm for suffix rules:

1. Extract the suffix from the target. If that target does not have a suffix, go to step 6.
2. Is it in the .SUFFIXES list? If not, quit the search.
3. If it is in the .SUFFIXES list, look for a double suffix rule that matches the target suffix.
4. If there is a match, extract the base name of the file, add on the second suffix, and determine if the resulting file exists. If the resulting file does not exist, keep searching the double suffix rules.

   If the resulting file does exist, use the recipe for this rule.
5. If a successful match is not made, the inference has failed.
6. If the target did not have a suffix, check the single suffix rules in the order that the suffixes are specified in the .SUFFIXES target.
7. For each single suffix rule, add the suffix to the target name and determine if the resulting filename exists.
8. If the filename exists, execute the recipe associated with that suffix rule. If the filename doesn't exist, continue trying the rest of the single suffix rules. If a successful match is not made, the inference has failed.

When the .POSIX special target is not specified, **make** handles suffix rules in the same manner as traditional implementations of **make**. The following steps describe the search algorithm for suffix rules in this situation.

1. Extract the suffix from the target. If that target does not have a suffix, go to step 8.
2. Is it in the .SUFFIXES list? If not, then quit the search.
3. If it is in the .SUFFIXES list, look for a double suffix rule that matches the target suffix.
4. If you find one, then extract the base name of the file, add on the second suffix and see if the resulting file exists. If it does, go to step 7. If not, continue with step 5.
5. Is there an inference rule for the resulting file? If yes, run the recipe associated with that rule (which should describe how to make the file exist) and go to step 7.
6. Search for the next double-suffix rule that matches the target suffix and return to step 4. If the double-suffix rules are exhausted, then the inference has failed.
7. Use the recipe for the target rule.

8. If the target did not have a suffix, then check the single-suffix rules in the order that the suffixes are specified in the .SUFFIXES target.
9. For each single-suffix rule, add the suffix to the target name and see if the resulting filename exists.
10. If the file exists, then run the recipe associated with that suffix rule. If it doesn't exist, continue trying the rest of the single-suffix rules.
11. If a successful match is not made, then the inference has failed.

make also provides a special feature in the suffix rule mechanism for archive library handling. If you specify a suffix rule of the form:

```
.suf.a:
        recipe
```

the rule matches any target having the LIBRARY attribute set, regardless of what the actual suffix was. For example, if your makefile contains the rules:

```
.SUFFIXES:  .a .c
     echo adding $< to library $@
```

then if mem$0 exists, then the following command:

```
make "mylib(mem.o)"
```

causes:

```
adding mem.o to library mylib
```

to be printed.

Attributes

make defines several target attributes. Attributes can be assigned to a single target, a group of targets, or to all targets in the makefile. Attributes affect what **make** does when it needs to update a target. You can associate attributes with targets by specifying a rule of the form:

attribute_list : *targets*

This assigns the attributes in *attribute_list* to the given targets. If you do not specify any targets, the attributes apply to every target in the makefile. You can also put attributes inside a normal rule, as in:

```
targets attribute_list : prerequisites
```

The recognized attributes are:

.EPILOG Insert shell epilog code when running a group recipe associated with any target having this attribute set.

.IGNORE Ignore an error when trying to make any target with this attribute set.

.LIBRARY Target is a library.

.PRECIOUS Do not remove this target under any circumstances. Any automatically inferred prerequisite inherits this attribute.

.PROLOG Insert shell prolog code when running a group recipe associated with any target having this attribute set.

.SETDIR Change the working directory to a specified directory when making associated targets. The syntax of this attribute is .SETDIR=*path*, where *path* is the pathname of desired working directory. If *path* contains any :

characters, the entire attribute string must be quoted, not just the pathname.

`.SILENT` Do not echo the recipe lines when making any target with this attribute set, and do not issue any warnings. You can use any attribute with any target, including special targets.

Special Target Directives

Special target directives are called targets because they appear in the target position of rules; however, they are really keywords, not targets. The rules they appear in are really *directives* that control the behavior of **make**.

The special target must be the only target in a special rule; you cannot list other normal or special targets.

Some special targets are affected by some attributes. Any special target can be given any attribute, but often the combination is meaningless and the attribute has no effect.

`.BRACEEXPAND` This target may have no prerequisites and no recipes associated with it. If set, the target enables the outdated brace expansion feature used in older versions of **make**. Older **make**s would expand a construct of the following form, beginning with each token in the token list:

```
string1{token_list}string2
```

Older **make**s would append *string1* to the front of each token in the list, and *string2* to the end of each token in the list. A more productive means for achieving the same result with modern versions of **make** relies on macro expansion with prefix and suffix modifiers:

```
$ (TOKEN_BASE:⅛"prefix:+"suffix")
```

The double quotes are required. Brace expansion is an outdated feature available in past versions of **make**.

`.CYCLECHECK` This special target cannot have any prerequisites or recipes associated with it. If set, it determines how **make** treaters circular dependencies (see “Circular Dependencies” on page 362).

You can specify one of five attributes with this target. If you specify more than one attribute, an error message results. The five attributes are:

`.SILENT` **make** remains silent about any within-rule and between-rule circular dependencies, removes the offending dependency from the list of prerequisites, and continues.

`.WARNTARG` **make** issues warnings for named targets with circular dependencies. If the name of the dependency is the same as the named target, it is removed from the list of prerequisites and **make** continues. This is the default behavior if `.CYCLECHECK` is not specified or is specified with no attributes.

.WARNALL **make** issues warnings for all within-rule circular dependencies regardless of whether the target is being built or not and for all between-rule circular dependencies for the named targets. The offending dependency is removed from the list of prerequisites and **make** continues.

.FATALTARG **make** treats all circular dependencies for named targets as fatal errors. It issues an error message and exits.

.FATALALL **make** treats all within-rule circular dependencies as fatal errors regardless of whether the target is being built or not. It also treats all between-rule circular dependencies for named targets as fatal errors. **make** issues an error message and exits.

For example, to set the circular dependency check to **make**'s default, use the rule:

```
.CYCLECHECK .WARNTARG:
```

.DEFAULT This target has no prerequisites, but it does have a recipe. If **make** can apply no other rule to produce a target, it uses this rule if it has been defined.

.ERROR **make** runs the recipe associated with this target whenever it detects an error condition.

.EXPORT All prerequisites associated with this target that correspond to macro names are exported to the environment at the point in the makefile at which this target appears.

.GROUPEPILOG **make** adds the recipe associated with this target after any group recipe for a target that has the .EPILOG attribute.

.GROUPPROLOG **make** adds the recipe associated with this target after any group recipe for a target that has the .PROLOG attribute.

.IMPORT **make** searches in the environment for prerequisite names specified for this target and defines them as macros with their value taken from the environment. If the prerequisite .EVERYTHING is given, **make** reads in the entire environment (see **–e** and **–E** options).

.INCLUDE **make** reads one or more additional makefiles (specified in the prerequisite list), as if their contents had been inserted at this point. If the prerequisite list contains more than one file, **make** reads them in order from left to right.

make uses the following rules to search for extra makefiles:

- If a relative filename is enclosed in quotes, or is not enclosed with angle brackets (< and >), **make** looks in the current directory. If the file isn't present, **make** then looks for it in each directory specified by the .INCLUDEDIRS special target.
- If a relative name is enclosed with angle brackets (< and >), **make** make only searches in directories specified by the .INCLUDEDIRS special

- If an absolute pathname is given, **make** looks for that file and ignores the list associated with the .INCLUDEDIRS special target.

.INCLUDEDIRS The list of prerequisites specified for this target defines the set of directories to search when including a makefile.

.MAKEFILES The list of prerequisites is the set of files to try to read as the user makefile. These files are made in the order they are specified (from left to right) until one is found to be up to date. This is the file that is used.

.NOAUTODEPEND Disables the autodependency feature when building libraries. When this special target is used, only library members that have been explicitly given as dependents are considered prerequisites.

.POSIX **make** processes the makefile as specified in the POSIX.2 draft standard. This target may have no prerequisite and no recipes associated with it. This special target must appear before the first non-comment line in the makefile. If this special target is present, the following facilities are disabled:

- All recipe lines are run by the shell, one shell per line, regardless of the setting of **SHELLMETAS**.
- Metarule inferencing is disabled.
- Conditionals are disabled.
- Dynamic prerequisites are disabled.
- Group recipes are disabled.
- Disables brace expansion (set with the **.BRACEEXPAND** special target).
- **make** does not check for the string *$ (MAKE)* when run with the **–n** options specified.

.REMOVE **make** uses the recipe of this target to remove any intermediate files that it creates if an error is encountered before the final target is created. This .REMOVE target only deletes files that satisfy all of the following criteria:

- The file didn't exist when **make** began running.
- The file is named as an intermediate target, produced by invoking a metarule that was produced by transitive closure.
- The file is not explicitly named in the makefile.
- The generated target doesn't have the .PRECIOUS attribute.
- The file is a prerequisite of a rule that is actually used.

.SOURCE The prerequisite list of this target defines a set of directories to check when trying to locate a target filename. **make** defaults to creating target files in the same directory that it finds the source file.

.SOURCE.x Same as .SOURCE, except that **make** searches the .SOURCE.x list first when trying to locate a file matching a target with a name that ends in the suffix **.x**.

.SUFFIXES — **mk** appends the prerequisite list of this target to the set of suffixes used when trying to infer a prerequisite for making a target using suffix rules. If you specify no prerequisites, **make** clears the list of suffixes, effectively disabling suffix rules from that point on.

A name of the form *library(member)* indicates a member of a library. The *library* portion is a target with the .LIBRARY attribute, and the *member* portion is a prerequisite of the library target.

A name of the form library((entry)) indicates the library module that contains the given entry point. Once again, the library portion is a target with the .LIBRARY attribute. **make** regards the library member that contains the entry point *entry* as a prerequisite of the library target.

Control Macros

make defines a number of control macros that control **make**'s behavior. When there are several ways of doing the same thing, control macros are usually the best. A control macro that has the same function as a special target or attribute also has the same name.

Macros that are said to be *defined internally* are automatically created by **make** and can be used with the usual $(*name*) construct. For example, $(PWD) can be used to obtain the current directory name.

Recognized control macros are:

DIRSEPSTR — Contains the characters used to separate parts in a pathname and can be set by the user. **make** uses the first character in this string to build pathnames when necessary.

.EPILOG — If assigned a nonnull value, the .EPILOG attribute is given to every target.

GROUPFLAGS — Specifies option flags to pass to GROUPSHELL when **make** invokes it to run a group recipe.

GROUPSHELL — Gives the pathname of the command interpreter (shell) that **make** calls to process group recipes.

GROUPSUFFIX — Specifies a string for **make** to use as a suffix when creating group recipe files to be run by the command interpreter.

.IGNORE — If this is assigned a nonnull value, **make** assigns the .IGNORE attribute to every target.

INCDEPTH — This is the current depth of makefile inclusion. It is set internally.

MAKE — This is set by the startup file and can be changed by the user. The standard startup file defines it as:

$(MAKECMD) $(MFLAGS)

The MAKE macro is not used by **make** itself, but the string $(MAKE) is recognized when using the **–n** option for single-line recipes.

MAKECMD — This is the name with which **make** was invoked.

MAKEDIR — This is the full pathname of the initial directory in which **make** began execution.

MAKEFLAGS
: The MAKEFLAGS macro contains all the options (flags) and macros specified in the **MAKEFLAGS** environment variable plus all of the options and macros specified on the command line, with the following exceptions.

  - Specifying **–c**, **–f**, or **–p** in the environment variable results in an error
  - These same options specified on the command line do not appear in the MAKEFLAGS macro.

  Options in the **MAKEFLAGS** environment variable may have optional leading dashes and spaces separating the options. These are stripped out when the MAKEFLAGS macro is constructed.

  Note: **make** always reads the **MAKEFLAGS** environment variable before reading the makefile. The **–E** and **–e** options do not affect this.

MAKESTARTUP
: This has the default value:

  ```
  /etc/startup.mk
  ```

  To change this value, you can set the **MAKESTARTUP** environment variable before running **make**. You can also specify a value for this control macro on the command line if you use the **–D** option:

  ```
  make –DMAKESTARTUP=$HOME/project/startup.mk
  ```

  Since **make** processes command-line macros after reading the startup file, setting this macro on the command line does not have the desired effect.

MFLAGS
: This is the same as MAKEFLAGS, except that it includes the leading switch character.

NULL
: This is permanently defined to be the null string.

.PRECIOUS
: If this is assigned a nonnull value, **make** assigns the .PRECIOUS attribute to every target.

.PROLOG
: If this is assigned a nonnull value, **make** assigns the .PROLOG attribute to every target.

PWD
: This is the full pathname of the working directory in which **make** is executing.

SHELL
: Specifies the full pathname of the command interpreter that **make** calls to process single-line recipes, when necessary. **make** passes recipe lines to this shell only if they contain one or more of the characters given in SHELLMETAS; otherwise, it runs them directly. By default, the value of the **SHELL** environment variable does not affect the value of this macro; however, you can use the .IMPORT special target to assign the environment variable's value to this macro. You can also use the .EXPORT special target to assign this macro's value to the **SHELL** environment variable.

SHELLFLAGS
: Specifies option flags to pass to the shell when invoking it to runs a single-line recipe.

SHELLMETAS	Specifies a list of metacharacters that can appear in single recipe lines. If **make** finds any metacharacter, it invokes the recipe using the shell specified by SHELL; otherwise, it runs the recipe without the shell.
.SILENT	If this is assigned a nonnull value, **make** assigns the .SILENT attribute to every target.

Making Libraries

A library is a file containing a collection of object files. To make a library, you specify it as a target with the .LIBRARY attribute and list its prerequisites. The prerequisites should be the object members that are to go into the library.

make tries to handle the old library construct format in a sensible way. When it finds lib(member), it declares the *lib* portion as a target with the .LIBRARY attribute and the *member* portion as a prerequisite of the *lib* target. To make the library properly, old makefile scripts using this format must name the *lib* as a target and must try to bring it up to date. The same thing happens for any target of the form *lib((entry))*. These targets have an additional feature in that the *entry* target has the .SYMBOL attribute set automatically.

Conditionals

You specify the conditional expression as follows:

```
.IF expression
... if text ...
.ELSE
... else text ...
.END
```

or:

```
.IF expression
... if text ...
.ELSIF expression2
... elsif text ...
.ELSE
... else text ...
.END
```

The .ELSE or .ELSIF portion is optional, and you can nest the conditionals (that is, the text may contain another conditional). The .IF, .ELSE, .ELSIF, and .END conditionals must start in the first column of the line. *expression* or *expression2* can have one of three forms:

```
string
```

is true if the given string is nonnull,

```
string == string
```

is true if the two strings are equal, and:

```
string != string
```

is true if the two strings are not equal. Typically, one or both strings contain macros, which **make** expands before making comparisons. **make** also discards white space at the start and end of the text portion before the comparison. This means that a macro that expands to nothing but white space is considered a null

value for the purpose of the comparison. If a macro expression needs to be compared with a null string, compare it to the value of the macro `$(NULL)`.

The text enclosed in the conditional construct must have the same format that it would have outside the conditional. In particular, **make** assumes that anything that starts with a tab inside the conditional is a recipe line. This means that you cannot use tabs to indent text inside the conditional (except, of course, for recipe lines, which always begin with tabs).

Files

make uses the following file:

/etc/startup.mk
The default startup file containing default rules.

Environment Variables

make uses the following environment variables:

MAKEFLAGS
Contains a series of **make** options that are used as the default options for any **make** command. You can specify the options with or without leading minus signs (–) and blanks between them. It can also include macro definitions of the form usually found on the command line.

MAKESTARTUP
Contains the pathname of the **make** stamp file. By default, **make** uses the file **/etc/startup.mk** as its startup file. To use a different file, set this environment variable before running **make**.

SHELL
Contains a name of a command interpreter. To assign this value to the **SHELL** control macro, use the `.IMPORT` special target. You can also use the `.EXPORT` special target to assign the value of the **SHELL** macro to the environment variable.

Localization

make uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Values

If a command in a recipe line fails (exits with a nonzero status), **make** returns the exit status of that command. Because most commands use exit status values between `0` and `10`, **make** uses exit status values below `10` only for failures that do not run recipe lines.

`0` Successful completion

1 Returned if you specified **–q** and file is not up to date

2 Failure due to any of the following:

- Unknown command-line option
- Missing argument to option, such as no file name for **–f**.

126
Recipe command was not executable.

127
Recipe command was not found.

129–254
make was interrupted by a signal; the error code is the signal number ORed with 128. For example, SIGINT is frequently signal 1; the return code from **make** is 128|1, or 129.

255
Failure due to any of the following:

- Macro cannot be redefined
- Macro variables not assigned with :=
- Special target cannot be a prerequisite
- Too many makefiles specified
- Configuration file not found
- No makefile present
- Missing .END for .IF
- No target
- Inability to return to directory
- Too many open files
- Open failed
- File not found
- Inability to change directory
- No more memory
- Line too long
- Circular macro detected
- Unterminated pattern string
- Unterminated replacement string
- Token separator string not quoted
- Unterminated separator string
- Expansion too long
- Suffix too long
- Unmatched quote
- .IF .ELSE ... END nesting too deep
- .ELSE without .IF
- Unmatched .END
- Inference rules resulting in circular dependency
- No macro name
- Write error on temp file
- Target not found, and cannot be made
- Inability to make *NAME*
- <+ diversion unterminated
- <+ diversion cannot be nested
- <+ missing before +>
- Incomplete rule recipe group detected
- Inability to mix single and group recipe lines

- Unmatched] found
- Macro or rule definition expected but not found
- Name too long
- Inability to determine working directory
- Only one *NAME* attribute allowed in rule line
- Multiple targets not allowed in `%` rules
- Special target must appear alone
- Duplicate entry in target list
- Syntax error in `%` rule, missing `%` target
- Duplicate entry in prerequisite list
- Missing targets or attributes in rule
- Multiply defined recipe for target
- Empty recipe for special target
- Imported macro *NAME* not found in environment
- No `.INCLUDE` file(s) specified
- Include file *NAME*, not found
- *NAME* ignored on special target
- Attributes possibly ignored
- Inability to find member defining `SYMBOL((`*NAME*`))`
- Incorrect library format
- Inability to touch library member
- `SHELL` macro not defined
- Too many arguments
- Inability to export *NAME*
- Inability to open *file*
- Circular dependency detected
- Inability to stat /
- Inability to stat `.`
- Inability to open `..`
- Read error in `..`
- Metarule too long: "*rule*"

Limits

1. The length of a single makefile script line cannot exceed 32768 characters.
2. The length of an argument string cannot exceed 32768 characters.
3. The length of a macro name gets truncated after 256 characters.

Usage Notes

1. When the `.SETDIR` special target is used, **make** checks the file attributes of targets and prerequisites on every pass through a rule. This can significantly increase the number of system accesses.
2. In a doublebyte environment, any character interpreted by **make** can be a doublebyte character, including those in macro definitions and targets.
3. In a doublebyte locale, if **make** encounters an incorrect doublebyte sequence, it ends with an error message.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The following features of **make** are enhancements to POSIX.2:

- The options: **–c***dir*, **–D** *macro definition*, **–E**, **–u**, **–V**, **–v**, and **–x**.
- The **–n** option has enhanced functionality not covered by the standard; for more information, see the **–n** option and the `POSIX` special target for **make**.
- The runtime macros: **$&**, $^ , **$**>.
- The dynamic prerequisites: **$$%**, **$$**>, **$$***, **$$@**.
- All macro expansions.
- Macro assignments of the following form:

```
macroname  :=  stringassigned
macroname  +=  stringassigned
```

- Brace expansion.
- Backslash continuation.
- The quoting mechanism, as in the following example:

```
"a:target" :  "a:prerequisite"
```

- All rule operators except the colon (:).
- Conditionals.
- Metarules.
- All **make** attributes *except* `.IGNORE`, `.PRECIOUS`, `.SILENT` (referred to in POSIX.2 as special targets).
- All **make** special targets *except* `.DEFAULT`, `.POSIX`, `.SUFFIXES` (referred to in POSIX.2 as special targets).
- All **make** macros *except* **SHELL** (referred to in POSIX.2 as control macros).

Related Information

S. I. Feldman, “Make—Program for Maintaining Computer Programs,” *Software—Practice and Experience* 9 (no. 4, April 1979):225–65 [Bell Labs, Murray Hill, NJ]

---

man — Display sections of the online reference manual

Format

man [**–wx**] [**–M** *path*] [*section*] *entry ...*
man –k [**–M** *path*] *keyword ...*

Description

man displays help information about a shell command or searches for help files having the specified keywords associated with them.

Options

–k Searches a precomputed database of syntax lines for information on keywords.

–M *path*
Searches the directories indicated by *path* for help files. If **–M** is not specified, **man** uses the path specified in the **MANPATH** environment variable if set; otherwise **man** searches **/usr/man/%L**. The value of the **LANG** environment variable is substituted for %L in this directory and in the directories specified by **MANPATH**. All help files are found by searching similarly structured file trees rooted at one or more places. See “Files” on page 379 for a description of what files and directories **man** should find in each directory that it searches.

–w Displays only the filename of the file containing the help file.

–x Displays what files **man** is searching to find the help file. **–x** also displays any errors man encounters in extracting man pages from online book files.

section
Is a number (0–9) representing a section of the online help. When you specify a section number, **man** searches only that section for entry, instead of searching all sections. The online help available for OS/390 UNIX contains one section:

1 Commands

To find a given entry, **man** checks each directory in **MANPATH** for a file with a specific name. For each section number requested, **man** searches **MANPATH** for the following files in this order:

1. **catn/entry.n** in each directory in **MANPATH**
2. **entry.n** in **/var/man/LANG** (the **man** "cache")
3. **mann/*.book** in each directory in **MANPATH**
4. **mann/entry.n** in each directory in **MANPATH**

If no section number is specified then **man** searches all sections in order from 1 to 9, then 0. The first entry found by **man** is the one displayed.

If output is to the terminal, then **man** invokes a pager command to filter and display the manual pages. If **MANPAGER** is defined, it is used. If not, then if **PAGER** is defined, it is used; otherwise, **man** defaults to using the command:

```
pg –e –p '(Page %d)'
```

Examples

To find out which utilities do comparisons, issue:

```
man –k compare
```

You can use the **man** command to view manual descriptions of TSO/E commands. To do this, you must prefix all commands with **tso**. For example, to view a description of the MOUNT command, you would enter:

```
man tsomount
```

You can also use the **man** command to view manual descriptions of commands that support subcommands. To do this, you must prefix all subcommands with the name of the command. For example, to view a description of the **dbx alias** subcommand, you would enter:

```
man dbxalias
```

The same applies for the **pdbx** subcommands. For example, enter:

```
man pdbxcont
```

to display information on the **cont** subcommand.

To view an online manual description for the tcsh **ls-F** built-in command, you must type **ls-F** without the dash. So, to see the man page you would issue:

```
man lsF
```

Environment Variables

man uses the following environment variables:

MANPATH
Contains a list of paths to search for man pages.

MANPAGER, PAGER
Contains an output filtering command for use when displaying man pages on a terminal.

TMPDIR
Identifies the directory where temporary files reside.

Localization

man uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Files

man uses the following files:

/usr/man/%L/man¬0–9|/*.book
BookManager book files containing man pages.

/var/man/%L/entry.¬0–9|/*.bookname
Cached man pages extracted from book files.

/usr/man/%L/cat¬0–9|/ * .¬0–9|
Subdirectories containing formatted help files.

/usr/man/%L/whatis
Database used by **–k** option.

Exit Values

0 Successful completion
1 Failure due to any of the following:
- Unknown command-line option
- Missing *path* after an **–M** option
- No information available on the desired subject
- Unable to create a child process to format man page
- Child process returned with nonzero exit status

Portability

POSIX.2 User Portability Extension, X/Open Portability Guide, UNIX systems.

The elements of the environment variable **MANPATH** are separated by colons.

The **–M** option, the **–x** option, the **–w** option, the **MANPAGER** environment variable, the default pager, and the ability to specify *section* on the command line are all extensions to the POSIX standard.

Related Information

help, **more**

---

mesg — Allow or refuse messages

Format

mesg [**y**] [**n**]

Description

mesg determines whether other users can send messages to your terminal with **talk**, **write**, or similar utilities.

Options

y Specifies that other people can send you messages.
n Specifies that other people cannot send you messages.

Examples

1. To let other people send you messages, issue:

   ```
   mesg y
   ```

2. To tell the system not to let other people send you messages, issue:

   ```
   mesg n
   ```

3. To display the current setting without changing it, issue:

   ```
   mesg
   ```

The terminal is determined by the first of standard input, output, or error which is directed to a terminal.

Localization

mesg uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Receiving messages is currently allowed
1 Receiving messages is not currently allowed
2 Failure due to any of the following:
- Unknown command-line option
- Unknown argument
- An error accessing the terminal

Portability

POSIX.2 User Portability Extension, X/Open Portability Guide, UNIX systems.

Related Information

talk, **write**

---

mkcatdefs — Preprocess a message source file

Format

mkcatdefs [**–h**] **MsgFile** [**SourceFile ...**]

Description

mkcatdefs preprocesses a message source file for input to the **gencat** utility.

SourceFile is a message file (usually with a **.msf** extension) containing symbolic identifiers. **mkcatdefs** produces two outputs:

- **MsgFile.h**, which contains statements that equate your symbolic identifiers with set numbers and message numbers that **mkcatdefs** assigns. You must include this header file in your application in order to refer to the messages.
- Message source data, with numbers instead of symbolic identifiers, is sent to standard output. This output is suitable as input to the **gencat** utility. You should either save standard output to a file using redirection, or pipe the output of **mkcatdefs** to the **gencat** utility.

Options

–h Suppresses the generation of a **MsgFile.h** file. This flag must be the first argument to **mkcatdefs**

Extended Description

The format of **SourceFile** is defined as follows: The fields of a message source line must begin in column 1 and are separated by a single blank character. Any other blank characters are considered as part of the subsequent field.

$quote
See **gencat**.

$set *symbolic_name*
The symbolic_name denotes the set identifier that will be used in an application program to reference this set of messages. This name can be up to 255 characters long and can contain any alphanumeric character and the underscore character, but must begin with a non-numeric character. Any string following the set identifier is treated as a comment.

$ *comment*
See **gencat**.

Symbolic_Name *message_text*
The `Symbolic_Name` denotes a message identifier that will be used in an application program to reference this message. This name can be up to 255 characters long and can contain any alphanumeric character and the underscore character, but must begin with a non-numeric character. There must be a single blank character separating the symbolic_name from the message_text. If no quote character is defined, then any blank characters after the separating blank character are considered part of the message text. See **gencat** for more information on how to specify *message_text*.

Examples

To process the **comp1.msf** and **comp2.msf** message source files and put the output into the **comp.msg** file, enter:

```
mkcatdefs comp comp1.msf comp2.msf >comp.msg
```

The source message file looks similar to the following:

```
$ This is the message source file for COMP1
$
$quote " Use double quotation marks to delimit message text
$set MSFAC1  Message set for component comp1
$
SYM_FORM "Symbolic identifiers can only contain alphanumeric \
characters or the _ (underscore character)\n"
SYM_LEN  "Symbolic identifiers cannot be more than 65 characters long\n"
5 "You can mix symbolic identifiers and numbers\n"
```

The generated **comp.h** file looks similar to the following:

```
#ifdef  _H_COMP_MSG
#include <limits.h>
#include <nl_types.h>
/*
 Time stamp: 1994 137 19:09 UTC
*/
/* The following was generated from comp1.msf. */
/* definitions for set MSFAC1 */
/* The following was generated from comp2.msf. */
/* definitions for set MSFAC2 */
#endif
```

mkcatdef creates the **comp.msg** message catalog source file for **gencat** with numbers assigned to the symbolic identifiers:

```
$timestamp 1994 137 19:09 UTC
$quote " Use double quotation marks to delimit message text
$delset 1
$set 1
1 "Symbolic identifiers can only contain alphanumeric \
characters or the _ (underscore character)\n"
2 "Symbolic identifiers cannot be more than 65 characters long\n"
5 "You can mix symbolic identifiers and numbers\n"
```

The assigned message numbers are noncontiguous because the source file contained a specific number. **mkcatdefs** always assigns the previous number plus 1 to a symbolic identifier.

Note: **mkcatdefs** inserts a **$delset** command before a **$set** command in the output message source file. This means you cannot add, delete, or replace single messages in an existing catalog when piping to the **gencat** utility. You must enter all messages in the set.

---

mkdir — Make a directory

Format

mkdir [**–p**] [**–m** *mode*] *directory ...*

Description

The **mkdir** command creates a new directory for each named *directory* argument. The mode for a directory created by **mkdir** is determined by taking the initial mode setting of 777 (a=rwx) or the value of **–m** if specified and applying the **umask** to it.

Options

–m *mode*

Lets you specify permissions for the directories. The *mode* argument can have the same value as the *mode* for **chmod**; see **chmod** for more details.

You can also set the sticky bit on for directories. For more information, see **chmod**. The **umask** value is applied to the *mode* value to determine the new directory's actual mode setting.

–p Creates intermediate directory components that don't already exist. For example, if one of the *directory* arguments is **dir/subdir/subsub** and **subdir** doesn't already exist, **mkdir** creates it. Such intermediate directories are created with mode bits determined in the following way: Take a default mode setting of `777` (`a=rwx`), apply the process's **umask** setting to it, and then turn on the user write and user execute permissions (`u+wx`). The **–m** mode specification on the command line is not used for computing the mode of intermediate directories.

Localization

mkdir uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

`0` Successful completion

`1` Failure due to any of the following:

- Missing *mode* after **–m**
- Incorrect *mode*
- Incorrect command-line option
- Missing directory name
- Inability to create the directory

Messages

Possible error messages include:

Path not found
The preceding structure (parent directory) of the named *directory* does not exist.

Access denied
The requested directory already exists or is otherwise inaccessible.

Cannot create directory
Some other error occurred during creation of the directory.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Information

rm, **rmdir**, **umask**

mkfifo — Make a FIFO special file

Format

mkfifo [**–m** *mode*] *file* [**–p**]

Description

mkfifo creates one or more FIFO special files with the given names.

Options

–m *mode*
Lets you specify file permissions for the files. The *mode* argument can have the same value as the *mode* argument for **chmod**; see **chmod** for more details.

–p Creates intermediate directory components that do not already exist. For example, if one of the file arguments is **dir/subdir/file** and if **subdir** does not exist already, this option creates it. Such intermediate directories are created with mode bits determined in the following way: Take a default mode setting 777 (a=rwx), apply the umask setting of the process to it, and then turn on user read, write, and user execute permissions (u+rwx).

The **–m** *mode* specification on the command line is not used for computing the mode of intermediate directories. The resulting mode settings permit the file owner to access the new files without concern for any umask setting that may be in place.

Localization

mkfifo uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion
1 Failure due to any of the following:
- A missing *mode* after **–m**
- An incorrect mode:
- An incorrect command-line option
- A missing filename
- Inability to create the desired file

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The **–p** option is an extension of the POSIX standard.

Related Information

chmod, **create**, **mkdir**

---

mknod — Make a FIFO or character special file

Format

mknod *pathname* **c** *major minor*
mknod *pathname* **p**

Description

mknod creates a FIFO special file or a character special file with the given pathname. It is located in the directory **/usr/sbin**.

Options

c Indicates character special files (for example, terminals and other devices). Option **–c** can only be used by a superuser.

major minor
major gives the major device type; *minor*, the minor device type. You can specify device types in decimal, hexadecimal, or octal.

mknod differentiates between octal and decimal as follows:

- Any number that starts with 0 but not 0x is octal.
- Any number that starts with 0x is hexadecimal.
- Any number that does not start with 0x or 0 is decimal.

For additional information on assigning major and minor numbers, see *OS/390 UNIX System Services Planning*.

p Creates a FIFO special file (that is, a named pipe).

Localization

mknod uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

Exit Values

0 Successful completion
1 Failure due to any of the following:
- Inability to create the desired file
- Incorrect *major* or *minor* number

2 Failure due to any of the following:
- Too few command-line arguments
- A missing *major* or *minor* device number

Portability

UNIX systems. Within POSIX, **mknod** has been superseded by **mkfifo** for pipes. The POSIX family of standards has not yet designed an alternative to **mknod** for special files.

Related Information

mkfifo

---

more — Display files on a page-by-page basis

Format

more [**–ceiSs**] [**–Al–u**] [**–n** *number*] [**–P** *prompt*] [**–p** *command*] [**–t** *tag*] [*file ...*]
more [**–ceiSs**] [**–Al–u**] [**–n** *number*] [**–P** *prompt*] [**–t** *tag*] [**+***command*] [*file ...*]

Description

more displays files one page at a time. It obtains the number of lines per page from the environment or from the **–n** option. If standard output (**stdout**) is not a terminal device, the number of lines per page is infinite.

more displays the files specified by *file ...* (that is, a list of filenames) one at a time. When **more** finishes displaying one file, it begins displaying the next one in the list. If you give – as one of the filenames, **more** reads the standard input at that point in the sequence.

more allows paging forwards and backwards (if possible) and searching for strings.

Options

–A Displays all characters, including unprintable ones. Usually, unprintable characters are displayed in a printable format. Also, ANSI escape sequences for display modes are processed. This option cannot be used with **–u**.

Note: The character in the top left corner of the screen is always displayed in normal mode.

–c Displays one page at a time starting at the top of the screen, and clears the screen before displaying a new file. **more** may ignore this option if the terminal doesn't support such operations.

–e Exits immediately after displaying the last line of the last file. Usually, if **stdout** is a terminal device, **more** stops after displaying the last line of the last file and prompts for a new command. If the command that displays text causes **more** to reach the end of the file again, **more** exits.

–i Ignores case during searches.

–n *number*
Specifies the number of lines per page. This overrides any values obtained from the environment. Use this option when you need to override the curses screen length or LINES setting to work with your terminal. This option will give

incorrect results if used while in the OMVS shell (or another dumb terminal) and specifying *number* to be something other than the current number of screen lines.

–P *string*
Sets the prompt that appears at end of each page of text to *string*. The default prompt is [*filename*]. **more** usually displays the prompt in standout mode.

–p *command*
Initially executes the **more** command on each file. If it executes successfully and *command* is a positioning command such as a line number or a regular expression search, **more** displays the resulting page; otherwise **more** displays the first page of the file. If both the **–t** and **–p** options are specified, the **–t** option is processed first.

+*command*
Initially executes the **more** command on each file. If it executes successfully and *command* is a positioning command such as a line number or a regular expression search, **more** displays the resulting page; otherwise **more** displays the first page of the file. If both the **–t** and **–p** options are specified, the **–t** option is processed first.

–S Displays the prompt in normal mode rather than standout (reverse video) mode.

–s Replaces consecutive empty lines with a single empty line.

–t *tag*
Searches for the named *tag* and displays the page of text containing it. See **ctags** for more information.

–u Displays all backspaces as ⅛H.

Usually *character*backspace_(underscore) displays *character* as underlined and *character*backspace*character*

Interactive Commands

more also supports the following interactive commands.

[*n*]**b** [*n*]**Ctrl-B** [*n*]**PgUp**	Moves backward *n* lines, with a default of one page. If *n* is more than the page size, **more** displays only the final page.
[*n*]**d** [*n*]**Ctrl-D**	Scrolls forward *n* lines, with a default of one half of the page size. If you specify *n*, it becomes the new default for subsequent d and u commands.
[*n*]**f** [*n*]**Ctrl-F** [*n*]**PgDn**	Moves forward *n* lines, with a default of one page. At end-of-file, **more** continues with the next file in the list, or exits if the current file is the last one in the list.
[*n*]**G**	Goes to the *n*th line in the file. If you do not specify *n*, **more** advances to the end of the file.
[*n*]**g**	Goes to the *n*th line in the file, with the default being the first line of the file.
h	Displays a summary of interactive commands.

[*n*]**j** [*n*]**SPACE** [*n*]**ENTER** [*n*]↓	Scrolls forward *n* lines, with a default of one line for **j**, **ENTER** and ↓, and a default of one page for **SPACE**. This command displays the entire *n* lines even if *n* is more than the page size. At end-of-file, these commands cause **more** to begin displaying the next file in the list, or to exit if the current file is the last one in the list.
[*n*]**k** [*n*]↑	Scrolls backward *n* lines, with a default of one line. This command displays the entire *n* lines even if *n* is more than the page size.
m*letter*	Marks the current position with the lowercase *letter*. When you view a new file, all previous marks are lost.
[*n*]**N**	Repeats the previous search, but in the opposite direction. If you specify *n*, **more** repeats the search *n* times.
[*n*]**n**	Repeats the previous search. If you specify *n*, **more** repeats the search *n* times. For example if there are eight occurrences of *pattern* in the file and /*pattern* found the second occurrence then a follow-up command of `5n` finds and sets the current position to the 7th occurrence of *pattern*.
q **:q** **ZZ**	Exits **more**.
R	Refreshes the screen and discards any buffered input.
r **Ctrl-L**	Refreshes the screen.
[*n*]**s**	Skips forward *n* lines (with a default of one line) and displays one page beginning at that point. If *n* would cause less than one page to be displayed, **more** displays the last page in the file.
[*n*]**u** [*n*]**Ctrl-U**	Scrolls backward *n* lines, with a default of one half of the page size. If you specify *n*, it becomes the new default for subsequent `d` and `u` commands.
v	Invokes an editor to edit the current file. **more** uses the editor named by the environment variable **EDITOR**. The default editor is **vi**.
'*letter*	Returns to the position marked with *letter*.
"	Returns to the position from which you last issued a movement command of greater than one page or the beginning of the file if you have issued no such commands.
[*n*]**/**[**!**]*pattern*	Searches forward in the file for the *n*th line containing *pattern*. *n* defaults to one if not specified. If *pattern* is the null regular expression (/) **more** uses the previous *pattern*. If the character `!` precedes *pattern*, **more** searches for lines that do not contain *pattern*.
[*n*]**?**[**!**]*pattern*	Searches backward in the file for the *n*th line containing *pattern*. The search begins at the line immediately before the top line displayed. *n* defaults to one if not specified. If *pattern* is the null regular expression (?), **more** uses the previous *pattern*. If the character `!` precedes *pattern*, **more** searches for lines that do not contain *pattern*.
:e [*filename*]`newline`	Stops viewing the current file and views *filename* instead. If you do not specify *filename*, **more** returns to the beginning of the current file. If *filename* is `#`, **more** returns to the last file viewed before the current one.
[*n*]**:n**	Views the next file from the list given on the command line. If you specify *n*, **more** views the *n*th next file from the list.
[*n*]**:p**	Views the previous file from the list given in the command line. If you specify *n*, **more** views the *n*th previous file from the list.
:t *tagname*	Goes to *tagname*.
:w *filename*	Writes the contents of the current file to the file *filename*.

:!<*shell command*>	Escape to shell and execute *shell command*.
=	Displays, where possible, the name of the file currently being viewed, its number (relative to the total number of files specified in the command line), the current line number, the current byte number, the total bytes to display and what percentage of the file has been displayed.
Ctrl-G	Displays, where possible, the name of the file currently being viewed, its number (relative to the total number of files specified in the command line), the current line number, the current byte number, the total bytes to display and what percentage of the file has been displayed.
Home	Goes to the first line in the file.
End	Goes to the last line in the file.

Environment Variables

more uses the following environment variables:

COLUMNS
Contains the maximum number of columns to display on one line.

EDITOR Contains the name of the editor that the **v** command invokes.

LINES Contains the number of lines in a page. This value takes precedence over value from **TERM**. However, the **–n** value takes precedence over the **LINES** value.

MORE Contains a list of options as they would appear on the command line. This variable takes preference over the **TERM** and **LINES** variables.

TERM Contains the name of the terminal type.

Localization

more uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Values

`0` Successful completion

`>0`
Failure due to any of the following:

- *filename* not a text file
- **–n** option too large
- Syntax error in regular expression
- Inability to create a file
- Inability to open input file
- Insufficient memory
- Incorrect command
- Inability to access the terminal
- Missing *string* after **-p** option

Portability

POSIX.2 User Portability Extension, UNIX systems.

The **–A**, **–P**, and **–S** options and the **:w** and **!** commands are extensions of the POSIX standard. The **Home**, **End**, **PgDn**, **PgUp**, ↓, and ↑ commands are extensions to traditional implementations of **more**, available only on terminal types which support these keys.

Related Information

cat, **vi**

---

mount — Logically mount a file system

Format

mount [**–t** *fstype*] [**–rv**] [**–a yes|no**] [**–o** *fsoptions*] [**–d** *destsys*] [**–s nosecurity|nosetuid**] **–f** *fsname pathname*

mount –q [**–d** *destsys*][**v**] *pathname*

Description

The **mount** shell command, located in **/usr/sbin**, is used to mount a file system or list all mounts over a file system.

Note: A **mount** user must have UID(0) or at least have READ access to the SUPERUSER.FILESYS.MOUNT resource found in the UNIXPRIV class.

Options

–a yes|no
For a system participating in shared HFS in a sysplex, specify **yes** to allow the system to automatically move logical ownership for a specified file system due to a system outage. Specify **no** to disable this move function. The default is **yes**.

–d *destsys*
Specifies the name of the system in a shared HFS environment that will be the logical owner of the mount. Note, if **–q** is specified, the **mount –q** output will only list mounts that are owned by *destsys*.

–f *fsname*
Names the file system to be mounted. All file system names must be unique. File system names are case sensitive, however,if the file system type is HFS, *fsname* will be translated to upper case. Options **–q** and **–f** are mutually exclusive, but one must be specified.

–o *fsoptions*
Specifies an option string to be passed to the file system type. NFS, for example, uses this to identify the remote server and the object on that server. The format and content are specified by the physical file system that is to perform the logical mount. You can specify lowercase or uppercase characters. Enclose the string in single quotes.

–q Prints a list of pathnames for the mountpoints of file systems mounted over a another file system, including that system. Options **–q** and **–f** are mutually exclusive, but one must be specified. If **–v** is not specified, only pathnames for mountpoints are printed. Note that the output of **mount –q** can be used by the **unmount** utility as input. See “Examples” on page 392.

–r Specifies mounting a file system read-only.

–s nosecurity|nosetuid
Specifies that a file system is unsecured. Setuid, setgid, APF and program controlled attributes are ignored when you use **nosetuid**. To additionally disable authorization checking, use **nosecurity**. Minimum unique abbreviations can be used for the option arguments.

–t *fstype* Identifies the file system type. *fstype* may be entered in mixed case but will be treated as upper case. If this option is not specified, the default is **–t** HFS.

–v Verbose output. Includes additional information, if available, on output.

pathname specifies the pathname for the mountpoint.

Examples

1. The output of **mount –q** can be used for the input of **unmount**. For example:

   ```
   mount -q /ict/hfsfir
   ```

   can be used as input:

   ```
   unmount $(mount -q /ict/hfsdir)
   ```

2. To mount an HFS file system over **/u/wjs** with a sync interval of 120 seconds:

   ```
   mount -f omvs.hfs.user.wjs -o 'SYNC(120)' /u/wjs
   ```

3. To display a list of pathnames for all mountpoints under **/u**:

   ```
   mount -q /u
   ```

Usage Note

Systems exploiting shared HFS will have I/O to an OMVS couple data set. Because of these I/O operations to the CDS, each mount request requires additional system overhead. You will need to consider the affect that this will have on your recovery time if a large number of mounts are required on any system participating in shared HFS.

Exit Values

0 Successful completion

Related Information

chmount, **unmount**

mv — Rename or move a file or directory

Format

mv [**–fiMPUv**] [**–F format|B|T|X**] [**–P params**]*file1 file2*
mv [**–ACfiMUv**] [**–F format|B|T|X**] [**–S suffix**] *file ... directory*
mv –Rr [**–fi**] *directory1 directory2*

Description

mv renames files or moves them to a different directory. If you specify multiple files, the target (that is, the last pathname on the command line) must be a directory. **mv** moves the files into that directory and gives them names that match the final components of the source pathnames. When you specify a single source file and the target is not a directory, **mv** moves the source to the new name, by a simple rename if possible.

You can also use **mv** to move files to and from MVS data sets. If you specify more than one file to be moved, the target (last pathname on command line) must be either a directory or a partitioned data set. If the target is an MVS partitioned data set, the source cannot be a UNIX directory.

When moving records, the string " \n" is moved the same way as the string "\n": both are read back as "\n", where "\n" indicates that OS/390 C++ will write a record containing a single blank to the file (the default behavior of OS/390 C/C++). All other blanks in your output are read back as blanks, and any empty (zero-length) records are ignored on input. However, if the environment variable **_EDC_ZERO_RECLEN** is set before calling **cp**, an empty record is treated as a single newline and is not ignored. Also, if **_EDC_ZERO_RECLEN** is set, a single newline is written to the file as an empty record, and a single blank will be represented by " \n".

A file can be moved by any user who has write permission to the directory containing the file, unless that directory has its sticky bit turned on. If the file is in a directory whose sticky bit is turned on, only the file owner or a superuser can move the file.

You can move:

- One file to another file in the working directory
- One file to a new file on another directory
- A set of directories and files to another place in your file system
- A UNIX file to an MVS data set
- An MVS data set to a file system
- An MVS data set to an MVS data set

Options

–A Specifies that all suffixes (from the first period till the end of the target) be truncated. **–A** has precedence over **–M** and **–C** options. **–S** will be turned off if **–A** is the last option specified.

–B Specifies that the data to be moved contains binary data. When you specify **–B**, **mv** operates without any consideration for <newline> characters or special characteristics of DBCS data (this type of behavior is typical when

moving across a UNIX system). **–B** is mutually exclusive with **–F**, **–X**, and **–T**, i.e., you will get an error if you specify more than one of these options.

–C Specifies truncating the filename(s) to 8 characters to meet the restriction in the MVS data set member.

–F *format*
Specifies if a file is binary or text and for text files, specifies the end-of-line delimeter. Also sets the file format to *format* only if the source is an MVS data set and the target is a UNIX file. Only **cp** sets the file format for UNIX to UNIX operations. For text files, when moving from UNIX to MVS, the end-of-line delimeter will be stripped. When moving from MVS to UNIX, the end-of-line delimeter will be added (Code page IBM-1047 will be used to check for end-of-line delimeters).

–F is mutually exclusive with **–B**, **–X**, **–p**, and **–T**. If you specify one of these options with **–F**, you will get an error. If **–F** is specified more than once, the last **–F** specified will be used.

For *format* you can specify:

not not specified
bin binary data

Or the following text data delimeters:

nl newline
cr carriage return
lf line feed
crlf carriage return followed by line feed
lfcr line feed followed by carriage return
crnl carriage return followed by new line

–f (UNIX to UNIX only)
Does not ask if you want to overwrite an existing destination without write permission; it automatically behaves as if you answered `yes`. If you specify both **–f** and **–i**, **mv** uses the option that appears last on the command line.

–i When moving to a UNIX target, always prompts before overwriting an existing file, but does not overwrite the file if you do not have permission. If you specify both **–f** and **–i**, **mv** uses the option that appears last on the command line.

–M Specifies that some characters of the filename are translated when moving between a UNIX file and a data set member. Characters are translated as follows:

- _ (underscore) in UNIX is translated to @ in MVS DS members and vice versa.
- . (period) in UNIX is translated to # in MVS DS members and vice versa.
- – (dash) in UNIX is translated to $ in MVS DS members and vice versa.

–P params
Specifies the parameters needed to create a sequential data set if one does not already exist. You can specify the RECFM, LRECL, BLKSIZE, and SPACE in the format CRTL **fopen()** function uses. However, LRECL and BLKSIZE can be used for variable record format only.

`SPACE=(units,(primary,secondary)` where the following values are supported for units:

- any positive integer indicating BLKSIZE
- CYL (mixed case)
- TRK (mixed case)

For example:

```
SPACE=(500,(100,500)) units, primary, secondary
SPACE=(500,100) units and primary only
```

Note: The CRTL**fopen()** arguments: LRECL specifies the length, in bytes, for fixed-length records and the maximum length for variable-length records. BLKSIZE specifies the maximum length, in bytes, of a physical block of records. RECFM refers to the record format of a data set and SPACE indicates the space attributes for MVS data sets.

–R (UNIX to UNIX only)
Moves a directory and all its contents (files, subdirectories, files in subdirectories, and so on). For example:

```
mv –R dir1 dir2
```

moves the entire contents of **dir1** to **dir2/dir1**. **mv** creates any directories that it needs.

–r (UNIX to UNIX only)
Is identical to **–R**.

–S *d=suffix|a=suffix*

- *d=suffix*

  Removes the specifed suffix from a file.

- *a=suffix*

  Appends the specified suffix to a file.

–S has precedence over **–M** and **–C**. It also turns off the **–A** option (if **–S** is the last specified option).

–T Specifies that the data to be moved contains text data. See “Usage Notes” on page 399 for details on how to treat text data. This option looks for IBM-1047 end-of-line delimeters, and is mutually exclusive with **–F**, **–X**, and **–B**, i.e., you will get an error if you specify more than one of these options.

Note: **–T** is ignored when moving across UNIX file systems.

–U Keeps filenames in uppercase when moving from MVS data set members to UNIX files. The default is to make filenames lowercase.

–v Verbose

–X Specifies that the data to be moved is an executable. Cannot be used in conjunction with **–F**, **–X**, or **–B**.

Note: If neither **–F|B|T nor X** is specifed, **mv** will first check the format of the MVS data set indicated and then try to determine the type of file.

Limits & Requirements

General Requirements

1. To specify an MVS data set name, precede the name with double slashes (//). For example, to specify the fully qualified data set names 'turbo.gammalib' and 'turbo.gammalib(pgm1)', you write:

   ```
   "//'turbo.gammalib'"
   "//'turbo.gammalib(pgm1)'"
   ```

   The same goes for data set names that are not fully qualified:

   ```
   //turbo
   ```

2. For PDS (partitioned data set) or PDSE (partitioned data set extended), to avoid parsing by the shell, the name should be quoted or minimally, the parenthesis should be escaped. For example, to specify 'turbo(pgm1)', you can use quotes:

   ```
   "//turbo(pgm1)"
   ```

   or escape the parenthesis:

   ```
   //turbo\(pgm1\)
   ```

   As indicated above, a fully qualified name must be single-quoted (as is done within TSO). To prevent the single quotes from being interpreted by the shell, they must be escaped or the name must be placed within regular quotation marks. See the 'turbo.gammalib' examples above.

3. If you specify a UNIX file as source and the MVS data set (target) does not exist, a sequential data set will be created. If the partitioned data set exists, the UNIX file will be moved to the partitioned data set member.
4. If source is an MVS data set and target is a UNIX directory, the UNIX directory must exist.
5. You cannot have a UNIX directory, partitioned data set, or sequential data set as source if the target is a partitioned data set.
6. To move all members from a partitioned data set, you may specify the partitioned data set as source and a UNIX directory as target.

MVS data set naming limitations

- Data set names may only contain uppercase alphabetic characters (A-Z). Lowercase characters will be converted to uppercase during any moves to MVS data sets.
- Data set names can contain numeric characters 0–9 and special characters @, #, and $.
- Data set names cannot begin with a numeric character.
- A data set member name cannot be more than 8 characters. If a filename is longer than 8 characters or uses characters that are not allowed in an MVS data set name, the file is not moved. You may use the **–C** option to truncate names to 8 characters.

Limitations: UNIX to MVS data set

1. If you specify a sequential MVS data set that is in undefined record format, the file is moved as binary.

2. If you specify a PDSE that is in undefined record format, the first file successfully moved determines in what format files will be moved. Note that PDSE does not allow mixture. So if the first successfully moved file is an executable, the PDSE will have program objects only and all other files will fail. On the other hand, if the first file is data, then all files are moved as binary.
3. If you specify a PDS that is in undefined record format, UNIX executables are saved as PDS load modules. All other files are moved as binary.
4. If you specify an MVS data set that is either in variable length or fixed record length and you have not set the file format, text files are moved as text, binaries as binary, and executables as binary. (IBM-1047 end-of-line delimeters are detected in the data)
5. If you set the file format, the set value is used to determine if data is binary or text.

Limitations: MVS data set to UNIX

1. If an HFS file does not exist, one is created using 666 mode value:

   ```
   666 mode value:  owner(rw-)  group(rw-)  other(rw-)
   ```

   whether data is binary or text. If the data to be moved is a shell script or executable, an HFS file is created using 777 mode value:

   ```
   777 mode value:  owner(rwx)  group(rwx)  other(rwx)
   ```

2. If an HFS exists and the file format is set, **mv** moves the file as that format. Otherwise,
   - load modules (PDS) are stored as UNIX executables and program objects (PDSE) are moved since they are the same as executables;
   - data within data sets of undefined record format are moved as binary if the data is not a program object or load module;
   - and data found within data sets of fixed length or variable record length are moved as text. (IBM-1047 end-of-line delimeters are detected in the data)

Limitations: MVS to MVS

1. Options **–A**, **–C**, **–f**, and **–S** are ignored.
2. If target and source are in undefined record format (and neither is a sequential data set), **mv** will attempt to move the data as a load module. If that fails, then **mv** will move the data as binary.
3. If target and source are in undefined record format and either is a sequential data set, **mv** moves the data as binary.
4. If the source has a fixed or variable record length and the target is in undefined record format, **mv** moves the data as binary.
5. If the source is in undefined record format and the target has a fixed or variable record length, **mv** moves the data as binary.
6. If both source and target are in fixed or variable record length, **mv** moves the data as text.

Limitations: Moving executables into a PDS

1. A PDS may not store load modules that incorporate program management features.

2. **c89**, by default, produces objects using the highest level of program management.
3. If you plan on moving a load module to a PDS, you may use a pre-linker which produces output compatible with linkage editor. Linkage editor generated output can always be stored in a PDS.

The following table is a quick reference for determining the correct use of options with **mv**.

Figure 9. **mv** *Format: File to File and File ... (multiple files) to Directory*

Source/Target	Options Allowed	Options Ignored	Options Failed
UNIX File/UNIX File	Ffi	ABCMPSTUX	
UNIX File/Sequential Data Set	BFiPT	ACfMSU	X
UNIX File/PDS or PDSE Member	BFiTX	ACfMPSU	
Sequential Data Set/UNIX File	BFfiTU	ACMPS	X
Sequential Data Set/Sequential Data Set	BFiPT	ACfMSU	X
Sequential Data Set/PDS or PDSE Member	BFiT	ACfMPSU	X
PDS or PDSE Member/UNIX File	BFfiTUX	ACMPS	
PDS or PDSE Member/Sequential Data Set	BFiPT	ACfMSU	X
PDS or PDSE Member/PDS or PDSE Member	BFiTX	ACfMPSU	
UNIX File/UNIX Directory	Fi	ABCFMPSTUX	
PDSE or PDSE Member/UNIX Directory	BFfiMSTUX	ACP	
UNIX File/Partitioned Data Set	ABCFiMSTX	fPU	
PDS or PDSE Member/PartitionedData Set	BFiTX	ACfMPSU	
UNIX Directory/UNIX Directory	fi	ABCFMPSTUX	
Partitioned Data Set/UNIX Directory	ABCFfiMSTUX	P	

The tables that follow indicate the kind of moves allowed using **mv**.

Figure 10 (Page 1 of 2). **mv** *Format: File to File*

Source	Target	Allowed
UNIX File, Sequential Data Set, or Partitioned Data Set Member	UNIX File, Sequential Data Set, or Partitioned Data Set Member	Yes
UNIX Directory (dir)	UNIX Directory (dir2 exists)	YES (NOTE: Results will be found in dir2/dir1/ ..).
UNIX Directory (dir)	UNIX Directory (dir2 does not exist)	YES (NOTE: Results will be found in dir2/...).

Figure 10 (Page 2 of 2). **mv** *Format: File to File*		
Source	**Target**	**Allowed**
Partitioned Data Set	UNIX Directory (dir) NOTE: results in each member of data set are moved to dir.	Yes
UNIX Directory	Partitioned Data Set	No
Partitioned Data Set	Partitioned Data Set	No
UNIX File, UNIX Directory, or Partitioned Data Set Member	UNIX Directory	Yes
Partitioned Data Set Member	Partitioned Data Set (must exist)	Yes

Figure 11. **mv** *Format: File... (multiple files) to Directory*		
Source	**Target**	**Allowed**
Any combination of UNIX File and/or Partitioned Data Set Member	UNIX Directory or Partitioned Data Set	Yes
Any combination of UNIX Directory, Partitioned Data Set, Sequential Data Set	Partitioned Data Set	No
Sequential Data Set	UNIX Directory	No
Any combination of UNIX Directory, UNIX File, Partitioned Data Set, Partitioned Data Set Member	UNIX Directory	Yes

Usage Notes

UNIX to MVS

1. To move from UNIX to a partitioned data set, you must allocate the data set before doing the **mv**.
2. If an MVS data set does not exist, **mv** will allocate a new sequential data set of variable record format.
3. For text files, all <newline> characters are stripped during the move. Each line in the file ending with a <newline> character is moved into a record of the MVS data set. If text file format is specified or already exists for the source file, that file format will be used for end-of-line delimeter instead of <newline>. Note that **mv** looks for IBM-1047 end-of-line delimeters in data.

   You cannot move a text file to an MVS data set that has an undefined record format:

   - For an MVS data set in fixed record format, any line moved longer than the record size will cause **mv** to fail with a displayed error message and error code. If the line is shorter than the record size, the record is padded with blanks.
   - For an MVS data set in variable record format: Any line moved longer than the largest record size will cause **mv** to fail with a displayed error message and error code. Record length is set to the length of the line.

4. For binary files, all moved data is preserved:
   - For an MVS data set in fixed record format, data is cut into chunks of size equal to the record length. Each chunk is put into one record. The last record is padded with blanks.
   - For an MVS data set in variable record format, data is cut into chunks of size equal to the largest record length. Each chunk is put into one record. The length of the last record is equal to length of the data left.
   - For an MVS data set in undefined record format, data is cut into chunks of size equal to the block size. Each chunk is put into one record. The length of the last record is equal to the length of the data left.
5. For load modules, the partitioned data set specified must be in undefined record format otherwise the executable will not be moved.
6. If more than one filename is the same, the file is overwritten on each subsequent move.
7. If a UNIX filename contains characters that are not allowed in an MVS data set, it will not be moved. If the UNIX filename has more than 8 characters, it can not be moved to an MVS data set member. (See the **–ACMS** options for converting filenames)
8. You are not allowed to move files into data sets with spanned records.
9. PDSE cannot have a mixture of program objects and data members. PDS allows mixing, but it is not recommended.
10. Special files such as character special, external links, and fifo will not be moved to an MVS data set.
11. If a file is a symbolic link, **mv** will move the resolved file, not the link itself.
12. UNIX file attributes are lost when moving to MVS. If you wish to preserve file attributes, you should use the **pax** utility.

MVS to UNIX

1. If the specified target HFS file exists, the new data overwrites the existing data.
2. If the specified HFS file does not exist, it will be created using a 666 mode value whether the data is binary or text (this is subject to **umask**). If the data to be moved is a shell script or executable, the HFS file will be created with a 777 mode value (also subject to **umask**).
3. When you move MVS data sets to UNIX text files, a <newline> character is appended to the end of each record. If trailing blanks exist in the record, the <newline> character is appended after the trailing blanks. If the file format option is specified or the target file has the file format set, that file format is used as the end-of-line delimeter instead of <newline>.
4. When you move MVS data sets to UNIX binary files, the <newline> character is not appended to the record.
5. You cannot use **mv** to move data sets with spanned record lengths.

Examples

1. To specify **–P** *params* for a non-existing sequential target:

   ```
   mv -P "RECFM=U,space=(500,100)"file "//'turbo.gammalib'"
   ```

2. To move file **f1** to a fully qualified sequential data set `'turbo.gammalib'` and treat it as a binary:

   ```
   mv -F bin f1 "//'turbo.gammalib'"
   ```

3. To move all members from a fully qualified PDS `'turbo.gammalib'` to an existing UNIX directory `dir`:

   ```
   mv "//turbo.gammalib'" dir
   ```

4. To drop .c suffixes before moving all files in UNIX directory `dir` to an existing PDS `'turbo.gammalib'`:

   ```
   mv -S d=.c dir/* "//'turbo.gammalib'"
   ```

Localization

mv uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Values

`0` Successful completion

`1` Failure due to any of the following:

- The argument had a trailing / but was not a directory
- Inability to find file
- Inability to open input file for reading
- Inability to create or open output file for output
- Read error on an input file
- Write error on an output file
- Input and output files identical
- Inability to unlink input file
- Inability to rename input file
- Irrecoverable error when using the **–r** option, such as:
  - Inability to access a file
  - Inability to read a directory
  - Inability to remove a directory
  - Inability to create a directory
  - A target that is not a directory
  - Source and destination directories identical

`2` Failure due to any of the following:

- Incorrect command-line option
- Too few arguments on the command line

- A target that should be a directory but isn't
- No space left on target device
- Out of memory to hold the data to be moved
- Inability to create a directory to hold a target file

Messages

Possible error messages include:

cannot allocate target string
mv has no space to hold the name of the target file. Try to free some memory to give **mv** more space.

filename?
You are attempting to move a file, but there is already a file with the target name and the file is read-only. If you really want to write over the existing file, type y and press <Enter>. If you do not want to write over the existing file, type n and press <Enter>.

source *name* **and target** *name* **are identical**
The source and the target are actually the same file (for example, because of links). In this case, **mv** does nothing.

unreadable directory *name*
mv cannot read the specified directory—for example, because you do not have appropriate permissions.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The **–R** and **–r** options are extensions of the POSIX standard.

Related Information

cp, **cpio**, **rm**

---

newgrp — Change to a new group

Format

newgrp [**–l**] [*group*]
newgrp [**–**] [*group*]

tcsh shell: **newgrp** [-] *group*

Description

newgrp lets you change to a new group. You stay logged in and your working directory does not change, but access permissions are calculated according to your new real and effective group IDs. If an error occurs, your session may be ended, and you must log in again.

After the group IDs are changed, a new shell is initialized within the existing process, effectively overlaying the current shell from which **newgrp** was invoked. The new shell is determined from the initial program value of the OMVS segment of your user profile.

newgrp does not change the value of exported shell variables, and all others are either set to their default or are unset.

If you did not specify any arguments on the command line, **newgrp** changes to the default group specified for your user ID in the system user database. It also sets the list of supplementary groups to that set in the systems group database.

If you specify a group, **newgrp** changes your real and effective group ID to that group. You are permitted to change to that group only if you are a member of that group, as specified in the system group database.

group can be a group name from the security facility group database, or it can be a numeric group ID. If a numeric group exists as a group name in the group data base, the group ID number associated with that group is used.

On systems where the supplementary group list also contains the new effective group ID or where the previous effective group ID was actually in the supplementary group list:

- If the supplementary group list also contains the new effective group ID, **newgrp** changes the effective group ID.
- If the supplementary group list does not contain the new effective group ID, **newgrp** adds it to the list (if there is room).

On systems where the supplementary group list does not normally contain the effective group ID or where the old effective group ID was not in the supplementary group list:

- If the supplementary group list contains the new effective group ID, **newgrp** removes it from the list.
- If the supplementary group list does not contain the old effective group ID, **newgrp** adds it to the list (if there is room).

newgrp in the tcsh shell

newgrp in the tcsh shell, as in the OS/390 shell, allows you to change to a new group.

Options

–l Starts the new shell session as a login session. This implies that it can run any shell profile code.

– Is the obsolescent version of **–l**.

Localization

newgrp uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Usage Notes

newgrp is not supported from an address space running multiple processes because it would cause all processes in the address space to have their security environment changed unexpectedly. If you are using the OMVS interface, you must be using the NOSHAREAS parameter before you issue the **newgrp** command. Also, if you are running in an environment with the **_BPX_SHAREAS** environment variable set to YES, you must unset it and start a new shell before issuing **newgrp**. For example:

```
unset _BPX_SHAREAS; sh
```

Exit Values

If **newgrp** succeeds, its exit status is that of the shell. Otherwise, the exit status is:

`>0`
: Failure because **newgrp** could not obtain the proper user or group information or because it could not run the shell, and it ends the current shell.

Portability

POSIX.2 User Portability Extension, UNIX systems.

Related Information

export, **fc**, **sh**, **tcsh**

---

nice — Run a command at a different priority

Format

nice [**–n** *number*] *command-line* **nice** [**–***number*] *command-line*

tcsh shell: **nice** [*+number*] [*command*]

Description

nice runs a command at a different priority than usual. Normally, **nice** lowers the current priority by 10.

The *command-line* must invoke a single utility command, without using compound commands, pipelines, command substitution, and other special structures.

nice in the tcsh shell

In the tcsh shell, **nice** sets the scheduling priority for the tcsh shell to *number*, or, without *number*, to `4`. With *command*, **nice** runs *command* at the appropriate priority. The greater the number, the less cpu the process gets. The super-user may specify negative priority by using:

```
nice -number ...
```

command is always executed in a sub-shell, and the restrictions placed on commands in simple if statements apply. See “tcsh — Invoke a C shell” on page 556.

Options

–n *number*
Lowers the current priority by *number*. On systems supporting higher priorities, a user with appropriate privileges can use **nice** to increase priority by specifying a negative value for *number*. For example,

```
nice –n –3 command
```

runs the command with an increased priority of 3.

–*number*
Is an obsolescent version of **–n** *number*.

Localization

nice uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

If **nice** invokes the *command-line*, it exits with the exit status returned by *command-line;* otherwise its exit status is one of the following:

1-125 An error occurred in the **nice** utility.
126 **nice** could not invoke *command-line*.
127 **nice** could not find the utility specified in *command-line*.

Portability

POSIX.2 User Portability Extension, X/Open Portability Guide, UNIX systems.

Related Information

nohup, **renice**, **tcsh**

---

nl — Number lines in a file

Format

nl [**–b***type*] [**–d***xy*] [**–f***type*] [**–h***type*] [**–i***n*] [**–l***n*] [**–n***fmt*] [**–p**] [**–s***sep*] [**–v**[*n*]] [**–w**[*n*]] [*file*]

Description

nl is a filter that numbers lines in a single file. If you do not specify *file* on the command line, the standard input is used.

The input is displayed as a stream of text lines, possibly divided into logical pages by separators. In turn, each page consists of a *header*, *body*, or *footer*, in that order. Any missing part is assumed to be empty. Using the default page delimiter

character of \ and :, lines consisting entirely of the following combinations are logical page part delimiters and are not numbered.

Input Line	Starts
***l*:*l*:*l*:**	Page header
***l*:*l*:**	Page body
***l*:**	Page footer

Options

–b*type*
Specifies the numbering type for each page body. The numbering type is one of the following:

a Numbers all lines

n Does not number any lines

pregexp
Numbers only those lines that contain the basic regular expression *regexp*. For more information about **regexp**, see Appendix C.

t Numbers only those lines that are not empty. An empty line consists of only a newline character.

The default body numbering type is *t*.

–d*xy*
Changes the default delimiter characters (\ and :) to characters *x* and *y*. If only *x* is specified, only the first delimiter character is changed.

–f*type*
Specifies the page footer numbering *type* (see the **–b** option). The default *type* is **n**.

–h*type*
Specifies the page header numbering *type* (see the **–b** option). The default *type* is *n*. (The lines are not numbered.)

–i*n* Sets the line increment to *n* rather than the default value of *l*.

–l*n* When the page numbering *type* is (all), blank lines are treated specially. Every *n*th consecutive blank line is numbered. If you do not specify this option, *n* defaults to 1 and every blank line is numbered.

–n*fmt*
Specifies the line numbering format, which must be one of the following:

n Right-justified line number, padded to *width* (see **–w**) on the left with spaces (the default format).

rz Right-justified line number, padded on left with zeroes.

ln Left-justified line number, padded on right with spaces.

–p Specifies continuous page numbering across page boundaries. By default, **nl** restarts numbering (as in the next option) at each new page.

–s*sep*
The string *sep* is printed to separate the line number from the text of the line being numbered. When this option is not specified, this separator is a single tab character.

–v*n* Starts numbering for each new page at *n*. If you do not specify this option, page numbering starts at 1.

–w*n* Sets the width of the line number in the output to *n*. If you do not specify *n*, the default is 6.

Example

The following command numbers every second consecutive blank line, using page delimiters of ˜!:

```
nl –l2 –ha –ba –fa –n  rz –v10 –i10 –d˜! file
```

Localization

nl uses the following localization environment variable:

- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Messages

0 Successful completion

1 Failure due to any of the following:

- Incorrect command-line argument
- More than one file name was specified
- Unable to open the file
- Incorrect regular expression in **–b**, **–f**, **–h**
- Incorrect numbering type
- Badly formed number in a command-line option

Portability

POSIX.2, X/Open Portability Guide.

Related Information

awk, **pr**

For more information about **regexp**, see Appendix C.

---

nm — Display symbol table of object, library, or executable files

Format

nm [**–AaefgnoPprsuv**] [**–t** *format*] *file ...*

Description

nm displays the symbol table associated with an object, archive library of objects, or executable files.

By default, **nm** lists the symbols in file in alphabetical order by name and provides the following information on each:

- File or object name (if you specified **–A**)

- Symbol name
- Symbol type. Not all of these symbol types are available on all systems. For instance, not all systems support the ability to determine different segment information.

  `A` Absolute symbol, global
  `a` Absolute symbol, local
  `B` Uninitialized data (bss), global
  `b` Uninitialized data (bss), local
  `D` Initialized data (bbs), global
  `d` Initialized data (bbs), local
  `F` Filename
  `l` Line number entry (see the **–a** option)
  `N` No defined type, global. This is an unspecified type, compared to the undefined type `U`.
  `n` No defined type, local. This is an unspecified type, compared to the undefined type `U`.
  `S` Section symbol, global
  `s` Section symbol, local
  `T` Text symbol, global
  `t` Text symbol, local (static)
  `U` Undefined symbol
- Symbol value
- Symbol size, if applicable

Options

–A Prefixes each line with the filename or archive member.

–a Displays all symbols, including line number entries on systems that support them.

–e Displays only global (external) and static symbols.

–f Displays full output. This is the default because output is not suppressed.

–g Displays only global symbols.

–n Is equivalent to **–v**.

–o Displays output in octal (same as **–t o**).

–p Displays output in a portable POSIX-compliant format, with blanks separating the output fields.

- If you specified **–A** and *file* is not a library, the format is:

  ```
  file: name type value size.
  ```

- If you specified **–A** and *file* is a library, the format is:

  ```
  file [object_file] : name type value size
  ```

  where *object_file* is the object file in the library that contains the symbol being described.

- If you did not specify **–A**, the format is:

  ```
  name type value size
  ```

- If you did not specify the **–t** option, **nm** displays *value* and *size* in hexadecimal.

- If you did not specify **–A** and the command line contains more than one file, or *file* is a library, **nm** displays a line preceding the list of symbols for each specified file or each object file in a specified library. If *file* is a library, this line has the following format:

  ```
  file[object_file]:
  ```

  If *file* is not a library, the format is:

  ```
  file:
  ```

–p Does not sort output.

–r Reverses sort order.

–s Includes symbol size for each symbol.

–t *format*
Defines the numeric value formatting base. The format is one of `d`, `o`, or `x`, for decimal, octal, or hexadecimal, respectively. If this option is not used, numbers are displayed in decimal.

–u Displays only undefined symbols.

–v Sorts output by value.

–x Displays information in hexadecimal (same as **–t x**).

Localization

nm uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLECT**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_TIME**

See Appendix F for more information.

Exit Values

`0` Successful completion

`1` Failure due to any of the following:
- Invalid command-line option
- Missing filename
- Unknown symbol table type
- Invalid library file
- End-of-file found in library
- Bad record in the library
- Out of memory

If a file does not contain a symbol table, **nm** displays a warning and goes to the next file, but this is not considered an error.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The **–a**, **–e**, **–f**, **–n**, **–o**, **–p**, **–r**, **–s**, and **–x** options are not part of the POSIX standard.

The **–a**, **–n**, **–p**, **–r**, **–s**, and **–t d**, options are not part of the X/Open standard.

Related Information

ar, **size**, **strip**

---

nohup — Start a process that is immune to hangups

Format

nohup *command-line*

tcsh shell: **nohup** *command*

Description

nohup invokes a utility program using the given *command-line*. The utility runs normally; however, it ignores the **SIGHUP** signal.

If the standard output is a terminal, **nohup** appends the utility's output to a file named `nohup.out` in the working directory. This file is created if it doesn't already exist; if it can't be created in the working directory, it is created in your home directory.

If the standard error stream is a terminal, **nohup** redirects the utility's error output to the same file as the standard output.

nohup simply runs a program from an executable file. *command-line* cannot contain such special shell constructs as compound commands or pipelines; however, you can use **nohup** to invoke a version of the shell to run such a command line, as in:

```
nohup sh –c 'command*ssq.
```

where *command* can contain such constructs.

nohup in the tcsh shell

With *command*, **nohup** runs command such that it will ignore hangup signals. Commands may set their own response to hangups, overriding **nohup**. Without an argument (allowed only in a shell script), **nohup** causes the tcsh shell to ignore hangups for the remainder of the script. See “tcsh — Invoke a C shell” on page 556.

Localization

nohup uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

126 **nohup** found the utility program but could not invoke it.
127 An error occurred before **nohup** invoked the utility, or **nohup** could not find the utility program.

Otherwise, the exit status is the exit status of the utility program that is invoked.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Information

exec, **hup**, **nice**, **sh**, **tcsh**

---

obrowse — Browse an HFS file

Format

```
obrowse [file...]
```

Description

obrowse enables you to browse a file in the hierarchical file system (HFS). This command uses the TSO/E OBROWSE command and must be run in the foreground. The 3270 passthrough mode is used to invoke the TSO/E OBROWSE command under OMVS.

You can specify any number of files; the TSO/E OBROWSE command is invoked once for each file. If you do not specify a filename, the main entry panel is displayed. From that panel, you can enter the directory name and filename of an existing file you want to browse. If you are browsing fixed-length records, you must also indicate the record length.

The filename can be absolute or relative. Avoid using single quotes or parentheses within the filename.

Usage Note

You cannot use **obrowse** if you used **rlogin** or **telnet** to access the shell.

Exit Values

0 The TSO/E OBROWSE command was invoked once for each file specified.

1 Failure because **obrowse** could not access at least one file because single quotes or parentheses were used in the filename.

2 Failure because **obrowse** was not able to set 3270 passthrough mode.

---

ocsconfig — Configure, unconfigure, or query an OCS object

Format

ocsconfig [**–a**] [**–c**|**–q**|**–Q**|**–u**] [**–d** *suboptions*] [**–f** *filename*] [**–s**|**–v**] [*name*]

Description

ocsconfig configures, unconfigures, or queries the current status of the specified Outboard Communications Server (OCS) object or objects. It processes OCS objects that are defined as a configuration file (**/etc/system** by default). For more information on preparing configuration files, see *OS/390 UNIX System Services Communications Server Guide*.

If you do not specify any options or operands, **ocsconfig** configures all OCS objects in the default configuration file that specify **Automatic=yes**. Entering **ocsconfig** with just the **–a** option configures all OCS objects in the default configuration file.

If you enter **ocsconfig** with a configuration or query option (**–c**, **–q**, **–Q**, or **–u**) and without an OCS object name, the specified operation is applied to all of the OCS objects in the default configuration file that specify **Automatic=yes**.

If you specify an OCS object name on the command line, the requested operation is performed on that specific object in the configuration file.

ocsconfig requires superuser authority.

Options

–a Processes all OCS objects in the configuration file. If you do not specify **–a**, all OCS objects in the configuration file that specify **Automatic=yes** are processed. You cannot use the **–a** option when you specify an OCS object name on the command line.

–c Configures the OCS object or objects. You cannot use this option with **–q**, **–Q**, or **–u**.

–d *suboptions*
Prints out special debugging information. Specify this option only when requested by IBM.

–f *filename*
Uses *filename* as the configuration file. If you do not specify this option, the default configuration file, **/etc/system**, is used.

–q Queries the active status of the specified OCS object or objects and sends the information to **stdout**. You cannot use this option with **–c**, **–Q**, or **–u**.

–Q Queries the active status of the specified OCS object or objects and sends the information to **stdout**. This option provides more information than **–q** does. You cannot use this option with **–c**, **–q**, or **–u**.

–s Suppresses all messages. This option does not suppress query output. You cannot use this option with **–v**.

–u Unconfigures the OCS object or objects. You cannot use this option with **–c**, **–q**, or **–Q**.

–v Writes out each line in the configuration file, intermixed with any syntax error messages. Informational messages are also written if files or directories are created or deleted. With this option, there is also no limit to the number of repetitious error messages produced.

You cannot use this option with **–s**.

name
The name of an OCS object in the configuration file. If *name* is specified, the **osconfig** operation is performed only on that object.

You cannot use the **–a** option when *name* is specified.

Query Output

The **osconfig** query output is written to **stdout**.

Two levels of detail are available. Normal query output is produced when **ocsconfig –q** is used. Extended query output is produced when **ocsconfig –Q** is used.

Normal query output contains a section with OCS summary information that applies to the whole system, and one section for each OCS node that was queried.

Extended query output contains additional information. There is a section for each OCS Remote-TTY (Rty) terminal device on each OCS node that was queried. See the sample output below for the exact layout of the normal and extended query output.

Most items in the query output are labeled with one of the following:

Active nodes
The number of OCS nodes that are currently configured.

Command
The most recent **ocsconfig** configure or unconfigure command issued prior to the query. Up to 64 bytes of command line text is shown. The displayed text may differ from that actually passed to the shell, because **osconfig** reconstructs the command line text from the passed arguments.

Configured Rtys
The combined number of OCS Rty terminal devices that are configured on all OCS nodes.

Debug flags
Undocumented flags (present only in extended query output).

Inoperative
The RTY terminal is inoperative because of an internal error. All subsequent system calls issued against this RTY terminal device return an errorno of EMVSERR until it is closed, at which time the inoperative condition is cleared.

OCS daemon state
The current processing state of the **OCS** daemon:

- **Connection in progress.** A connect request has been sent by an OCS node and is being processed by the **OCS** daemon.
- **Inactive.** The **OCS** daemon is not active.
- **Shutting down.** The **OCS** daemon is terminating. This occurs during the **ocsconfig** command unconfigure processing for the last OCS node.
- **Starting.** The **OCS** daemon is starting. This occurs during **ocsconfig** command configure processing for the first OCS node.
- **Waiting for connections.** The **OCS** daemon is waiting for OCS nodes to connect. This is the normal state of operation for this daemon.
- **Waiting for TCP/IP.** MVS TCP/IP is not started or is in the process of starting. When TCP/IP is available, there may be a slight time delay before the **OCS** daemon begins accepting OCS node connect requests.

OCS node name
The name of the OCS node. This name comes from the **OCS_Node_Name** attribute for the OCS object in the configuration file. It is the name passed to **gethostbyname()** to get the TCP/IP address for the OCS node.

OCS object name
The name of the OCS object in the configuration file.

Maximum Rtys
The maximum number of OCS Rty terminal devices that can be defined in the file system. All minor numbers for these devices (the **MVS_Device_Char_Minor** values in the configuration file) must be strictly less than this value. This value comes from the **MAXRTYS** statement in the BPXPRMxx SYS1.PARMLIB member.

Open pending
An open for this Rty terminal device is in progress.

Open Rtys
The number of OCS Rty terminal devices that are currently open for this OCS node.

Pending
The number of syscalls for this Rty terminal device that are currently executing on the OCS node.

Rty states
A list of current processing states for this OCS Rty terminal device. One or more of the following states may be displayed:

- **Close pending.** The Rty terminal device is closing.
- **Closed.** The Rty terminal device is closed.
- **First open pending.** The first open for this Rty terminal device is in progress.
- **Open.** The Rty terminal device is open and in use.

- **Unconfigure pending.** The OCS node containing this Rty terminal device was unconfigured while this Rty terminal device was still open. All subsequent syscalls issued against this Rty terminal device returns an errno of EIO until it is closed, at which time the Rty terminal device is unconfigured.

Sent
The total number of Rty syscalls for this terminal device that have have been sent to the OCS node during the current session.

State
The current processing state of the OCS node:

- **Connected.** The OCS node has established a network connection to the RISC System/6000 OCS and can accept user logins.
- **Not configured.** The OCS node has not been configured to the shell.
- **Not connected.** The OCS node has been configured but a network connection has not been established to the RISC/6000 OCS.

TCP/IP address
The TCP/IP network address for this OCS node. This is the address that **gethostbyname()** returned for the displayed OCS node name.

Time
The time that the displayed **ocsconfig** command was run.

UID :
The UID of the user that ran the displayed **ocsconfig** command.

UID =
The UID of the most recent user to log on to this OCS Rty terminal device.

User
The name of the most recent user to log on to this OCS Rty terminal device.

User name
The name of the user that ran the displayed **ocsconfig** command.

The first line of information displayed for each device in the extended query output contains three unlabeled fields:

1. The minor number for the device.

   The minor number for each device is taken from the **MVS_Device_Char_Minor** attributes in the configuration file.

2. The pathname for this device in the MVS file system.

   The pathname for each device is taken from the **MVS_Device_Char_Files** attributes in the configuration file.

3. The pathname for this device in the RISC/6000 file system.

   The pathname for each device is taken from the **OCS_Device_Char_Files** attributes in the configuration file.

An example of normal query output (**ocsconfig –q**) follows:

```
----------------- OCS query results -----------------
Most recent ocsconfig configure/unconfigure command:

Command  : ocsconfig –c node2
UID      : 0
User name: WELLIE0
Time     : Tue Dec 13 02:57:35 1994

OCS daemon state: Waiting for connections     Active nodes = 2
RTYs            : Configured RTYs = 32        Maximum RTYs = 256

OCS object name : node1
  OCS node name : oeaix4                      TCP/IP address = 129.40.41.2
  RTYs          : Configured RTYs = 16        Open RTYs = 2
  State         : Connected

OCS object name : node2
  OCS node name : aix2012                     TCP/IP address = 9.114.7.14
  RTYs          : Configured RTYs = 16        Open RTYs = 0
  State         : Not connected
```

An example of extended query output (**ocsconfig –Q**) follows:

```
--------------------- OCS extended query results ---------------------
Most recent ocsconfig configure/unconfigure command:

Command  : ocsconfig –c node2
UID      : 0
User name: WELLIE0
Time     : Tue Dec 13 03:22:51 1994

OCS daemon state: Waiting for connections     Active nodes = 2
RTYs            : Configured RTYs = 4         Maximum RTYs = 256

OCS object name : node1
  OCS node name : oeaix4                      TCP/IP address = 129.40.41.2
  RTYs          : Configured RTYs = 2         Open RTYs = 1
  State         : Connected
  Debug flags   : C8300000

  Terminal devices
  ---------------------------------------------------------------------

  0     /dev/vty0000                          /dev/pts/0
        User = WELLIE1    UID = 11            Sent = 2374       Pending = 1
        RTY states = Open

  1     /dev/vty0001                          /dev/pts/1
        User =            UID = 0             Sent = 807        Pending = 0
        RTY states = Closed



OCS object name : node2
  OCS node name : aix2012                     TCP/IP address = 9.114.7.14
  RTYs          : Configured RTYs = 2         Open RTYs = 0
  State         : Not connected
```

```
Debug flags   : 80000000

Terminal devices
---------------------------------------------------------------------

100   /dev/vty0100                          /dev/pts/0
      User =            UID = 0             Sent = 0          Pending = 0
      RTY states = Closed

101   /dev/vty0101                          /dev/pts/1
      User =            UID = 0             Sent = 0          Pending = 0
      RTY states = Closed
```

Examples

1. To configure all OCS objects in **/etc/system** that have the **Automatic=yes** attribute, enter:

   ```
   ocsconfig
   ```

2. To configure all OCS objects in **/etc/system**, enter:

   ```
   ocsconfig –a
   ```

3. To unconfigure node `ocsnode1` defined in configuration file **/cfg/c01**, enter:

   ```
   ocsconfig –f/cfg/c01 –u ocsnode1
   ```

4. To configure node –`OCS_NODE_3` defined in configuration file **/etc/system**, enter:

   ```
   ocsconfig –vc –– –OCS_NODE_3
   ```

   All source lines in the entire **/etc/system** file, along with any error messages, are written out.

5. To query all nodes defined in the **/etc/system**, showing all available details, enter:

   ```
   ocsconfig –aQ
   ```

Note: **/samples/system** contains sample configuration stanzas.

Environment Variables

ocsconfig uses the following environment variable:

X_SITE Influences how **gethostbyname()** resolves the network address of the specified OCS node.

Localization

ocsconfig uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Files

/dev/ocslock
Prevents more than one user from running **ocsconfig** simultaneously.

/etc/system
Default configuration file.

/dev/ocsadmin
Defined by OCS to perform internal administrative functions.

TCPIP.ETC.SERVICES, /etc/services
Contains the port number for **OCS** daemon service.

Usage Notes

1. When you delete devices using the **–c** or **–u** options, be careful not to delete the terminal from which you are issuing the command. If you do, the command may not complete and the appropriate OCS device character special files may not be defined or deleted. For this reason, you should issue all **ocsconfig** commands for OCS from a non-OCS terminal.
2. **ocsconfig** ignores the current file mode creation mask set by the **umask** command.
3. Only one user at a time can run the **ocsconfig** command. If there is an **ocsconfig** operation in progress, **ocsconfig** waits for the previous operation to complete.
4. If you try to interrupt **ocsconfig** while it is processing an OCS object, it will not end until the processing of that OCS object is complete.

   You can interrupt **ocsconfig** at any other time.
5. **ocsconfig** expects all data in the configuration file to be in the code page used by the **LC_SYNTAX** locale.
6. If the OCS object name begins with a dash (**–**), you must use the double dash (**—**) end-of-options delimiter on the command line before the object name.
7. When you enter **ocsconfig** from the shell, any characters with special meaning to the shell in the OCS object name or configuration filename must be escaped. To escape these special characters, enclose the filename or OCS object name in double quotes, or precede the special characters with a backslash.
8. After you change the OCS system configuration by adding, deleting, or reconfiguring one or more OCS nodes, invoke the **kill –1** command against the process ID (PID) for **lm**. This tells **lm** to reread the mapping of the OS/390 and OCS device names and device numbers from the OCS administrative pseudodevice driver.
9. Entering **ocsconfig** can both add terminals to the current OCS node configuration and delete terminals from it when you specify the **–c** option. The RTY device driver determines which devices in the new configuration stanza are not in the current configuration for the OCS node, and adds those devices to the configuration. It also determines which devices in the current configuration for the OCS node are not in the new configuration stanza, and deletes those devices from the configuration. Devices that are in both the current configuration and the new configuration stanza are kept.

10. **ocsconfig** obtains the TCP port for the **OCS** daemon service by using the **getservbyname()** function. If the **OCS** daemon port number cannot be determined, a default value of 802 is used.

Exit Values

0 Successful completion
1 Incorrect command line arguments or options
2 Failure because of any of the following:

- Syntax errors in the configuration file prevented one or more stanzas from being processed.
- I/O error occurred while reading the configuration file.
- Errors occurred while defining **/dev/ocsadmin**.
- Errors occurred while defining **/dev/ocslock**.
- Errors occurred while defining character special files needed by OCS.
- Errors occurred while removing character special files no longer needed by OCS.
- Too many OCS devices were specified in the configuration file.
- Unacceptable OCS file names were specified in the configuration file.
- Unexpected or internal errors occurred.
- The OCS node specified on the command line was not found in the configuration file.
- No OCS node name was specified on the command line, and none of the nodes defined in the configuration file specified the **Automatic=yes** attribute.
- The network address for one or more OCS nodes cannot be found.
- One or mode nodes to be unconfigured are not currently configured.
- The minor number in one or more character special files was too large.
- One or more devices cannot be configured because they are still in use after a previous unconfiguration request.

Portability

None. **ocsconfig** is not described in any standard.

Related Information

kill, **lm**

---

od — Dump a file in a specified format

Format

od [**–v**] [**–A** *addr_fmt*] [**–j** *num* [**bkm**]] [**–N** *num*] [**–t** *type_string*] [*file ...*]

od [**–bcDdhOoSsXx**] [*file*] [[**+**]*offset*[**.**][**b**]]

Description

od (octal dump) dumps a file to the standard output in a format specified by command-line options. The default format is octal words. You can use combinations of options to generate multiple formats with the requested representation of each byte vertically aligned. The file seek address (in octal) precedes each line of new data.

od recognizes two syntaxes. The first one conforms to POSIX. If you choose the first form, **od** displays files from the list *file* one at a time. If no *file* appears on the command line, **od** reads the standard input.

Options

The first form of **od** accepts the following options:

–v Displays all lines. Normally, **od** does not display multiple lines that differ only in the address. It displays the first line with a single `*` under it. to show that any subsequent lines are the same.

–A *addr_fmt*
Specifies the format that **od** uses to display the address field. *addr_fmt* can be **d** (decimal), **o** (octal), **x** (hexadecimal), or **n** (do not display address). The default is **–A o**.

–j *num*
Skips *num* bytes from the beginning of the file. If you precede *num* with `0X` or `0x`, **od** interprets it as hexadecimal. If you precede it with `0`, **od** interprets it as octal; otherwise, **od** assumes it is decimal. You can also append **b**, **k**, or **m** to *num* to indicate 512-byte blocks, kilobytes, or megabytes instead of bytes.

Be careful with this option when working with doublebyte characters. If byte *num*+1 (the starting byte, after skipping *num* bytes) is not the first byte of a character, **od** proceeds as though it is, resulting in a misinterpretation of that and subsequent characters. This misinterpretation continues until **od** encounters a <newline>. Then it is once again synchronized with the first byte of a doublebyte character.

–N *num*
Processes a maximum of *num* bytes. Be careful with this option when working with doublebyte characters. If **od** is processing a doublebyte character when it encounters the *num*th byte and this byte is not the last byte of the character, **od** displays `???` instead of the character.

–t *type_string*
Specifies the output format. *type_string* can contain the following format characters:

a Named characters from the ISO 646 character set. Data is interpreted as if it was coded in the ISO 646 character set.

c Characters. **od** displays nonprintable characters as backslash sequences and displays printable doublebyte characters properly.

A printable doublebyte character is displayed in the first byte position, and the remaining positions to the end of the character display ** to indicate the doublebyte character. Nonprintable doublebyte characters are displayed using a 3-digit octal number to represent each byte.

Also, incorrect doublebyte sequences are displayed with ??? for each incorrect byte.

d Signed decimal. A one-digit number may follow **d** telling **od** how many bytes to use. This must correspond to the size of a *char*, a *short*, an *int*, or a *long*. The default size is the size of an *int*. A symbolic size character can follow **d**, rather than the number of bytes. These have the following meaning:

C Corresponds to number of bytes in a *char*
S Corresponds to number of bytes in a *short int*
I Corresponds to the number of bytes in an *int*
L Corresponds to the number of bytes in a *long int*

f IBM S/390 Hexadecimal Floating-point. A one-digit number can follow **f**, telling **od** how many bytes to use. This must correspond to the size of a *float*, *double*, or *long double*. The default size is the size of a *double*. A symbolic size character can follow **f**, rather than the number of bytes. These have the following meaning:

F Corresponds to size of *float*
D Corresponds to size of *double*
L Corresponds to size of *long double*

F IEEE Binary Floating-point. A one-digit number can follow **F**, telling **od** how many bytes to use. This must correspond to the size of a *float*, *double*, or *long double*. The default size is the size of a *double*. A symbolic size character can follow **F**, rather than the number of bytes. These have the following meaning:

F Corresponds to size of *float*
D Corresponds to size of *double*
L Corresponds to size of *long double*

o Octal. A one-digit number can follow **o**, telling **od** how many bytes to use. This must correspond to the size of a *char*, a *short*, an *int*, or a *long*. The default size is the size of an *int*. A symbolic size character can follow **o**, rather than the number of bytes. These have the following meaning:

C Corresponds to number of bytes in a *char*
S Corresponds to number of bytes in a *short int*
I Corresponds to the number of bytes in an *int*
L Corresponds to the number of bytes in a *long int*

u Unsigned decimal. A one-digit number can follow **u**, telling **od** how many bytes to use. This must correspond to the size of a *char*, a *short*, an *int*, or a *long*. The default size is the size of an *int*. A symbolic size character can follow **u**, rather than the number of bytes. These have the following meaning:

C Corresponds to number of bytes in a *char*
S Corresponds to number of bytes in a *short int*

I Corresponds to the number of bytes in an *int*
L Corresponds to the number of bytes in a *long int*

x Hexadecimal. A one-digit number can follow **x**, telling **od** how many bytes to use. This must correspond to the size of a *char*, a *short*, an *int*, or a *long*. The default size is the size of an *int*. A symbolic size character can follow **x**, rather than the number of bytes. These have the following meaning:

C Corresponds to number of bytes in a *char*
S Corresponds to number of bytes in a *short int*
I Corresponds to the number of bytes in an *int*
L Corresponds to the number of bytes in a *long int*

Multiple format characters can appear in one *type_string* and multiple **–t** options can appear on the command line. If there is no **–t** option, the default is `–t o2`.

The second form of **od** is the historical (Berkeley Software Distribution) implementation of the command. If you use this form, you can specify only a single input *file*. If you do not give a *file* argument, **od** reads the standard input. You can supply an offset, but you must precede it with a plus sign (+) to distinguish it from a filename if no file is given. Giving an offset causes a seek to a position in the file where output begins. If the offset ends in a period (`.`), **od** considers it to be decimal; otherwise, **od** considers it octal. If you follow the offset with a `b`, **od** multiplies it by the block size of 512 bytes. The format of the offset determines the format of the address; that is, if it is interpreted as decimal, the addresses are displayed in decimal.

Note: The **od** command does not work on a file whose filename starts with either a digit or a plus (+) sign, unless the **–A**, **–N**, **–j**, or **–t** options are used.

The second form of **od** accepts the following options:

–b Bytes in octal
–c Bytes as characters
–D Unsigned decimal longs (4 bytes)
–d Unsigned decimal words (2 bytes)
–h Bytes in hexadecimal
–O Unsigned octal longs
–o Unsigned octal words
–S Signed decimal longs
–s Signed decimal words
–X Unsigned hexadecimal longs
–x Unsigned hexadecimal words

Localization

od uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_NUMERIC**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion
1 Failure due to any of the following:
- Inability to open the input file
- Badly formed offset
- Seek or read error on the input file

2 Failure due to any of the following:
- Incorrect command-line argument
- The wrong number of command-line arguments
- Incorrect format character
- Incorrect size modifier for format character

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The options to operate on longs (**–OSXD**) and the hex byte (**–h**) are extensions to the POSIX standard.

Related Information

dd

---

oedit — Edit an HFS file

Format

```
oedit [file...]
```

Description

oedit enables you to edit a file in the hierarchical file system (HFS). This command uses the TSO/E OEDIT command and must be run in the foreground. The 3270 passthrough mode is used to invoke the TSO/E OEDIT command under OMVS.

You can specify any number of files; the TSO/E OEDIT command is invoked once for each file. If you do not specify a filename, the Edit Entry panel is displayed. From that panel, you can enter the directory name and filename of an existing file, or you can specify a directory name and filename for a new file. The Edit Entry panel also lets you specify an edit profile and an initial edit macro.

The filename can be absolute or relative. Avoid using single quotes or parentheses within the filename.

Usage Notes

1. **oedit** attempts to load the file into a VB255 session. If this is an ISPF that supports wide edit (such as ISPF 4.1) and any line exceeds 235 characters, the width for the new session is the length of the longest line plus 25% to allow for some expansion.
2. The COPY command cannot copy in files that have records wider than the edit session.
3. **oedit** attempts to open an existing file as read/write. If this fails, it will attempt opening the file read-only to allow the user to view the file. Changes made in

this mode cannot be saved to the file. If changes are made, the edit session must be ended using the ISPF CANCEL primary command. However, you can use the ISPF CREATE and REPLACE primary commands to save all or part of the changed file to another file before you CANCEL the edit session.

4. You cannot use **oedit** if you used **rlogin** or **telnet** to access the OS/390 shell.

Exit Values

0 The TSO/E OEDIT command was invoked for each file specified.

1 Failure because **oedit** could not access at lease one file because single quotes or parentheses were used in the filename.

2 Failure because **oedit** could not set 3270 passthrough mode.

---

pack — Compress files by Huffman coding

Format

pack [[**–**] [**–f**] [**–o** *file*] *file*] ...

Description

pack compresses files using a Huffman minimal redundancy code on a byte basis. Each file is compressed in place; the resulting file has a **.z** extension appended to the file name, but keeps the same owner and permissions. For example, **abc** is compressed into **abc.z**. The times of last access and last modification are also preserved.

Packed files can be identified by **file** and uncompressed by **unpack** (which unpacks the file in place) or **pcat** (which unpacks to the standard output).

Normally **pack** reports the degree of compression achieved in each file (the report is printed on **stdout**). This number can be negative for small files with little redundancy if the **–f** option is used.

pack does not pack files if:

- The file appears to have already been packed.
- The filename is too long (an error will occur if **.z** is appended).
- The file has links or is a directory
- The packed file would be larger than the existing file (this includes empty files).
- The destination file already exists, or there is an error in processing.

Options

– Displays more detail on size, overhead and entropy (information rate). If this option is used several times on the command line it acts as a toggle, inverting the detailed-report flag at each mention.

–f Forces compression when it normally would not occur. Without this option, **pack** does not compress a file if its size is not reduced by compression, the file is already compressed, or the file has more than one link.

–o *file*
Specifies a different output file so that compressed output is written to *file* rather than overwriting the original input file. Several input and output files may be specified. For example,

```
pack –o out1 in1 –o out2 in2
```

packs file **in1** into **out1** and file **in2** into **out2**. The input files are not changed.

Localization

pack uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

`0` Successful completion

`1` An error occurred related to manipulating (opening, closing, renaming) the file, or a single file could not be packed properly.

`n` Indicates that `n` files could not be packed properly. For example, if three out of six files could not be packed properly, the exit status is `3`.

file: `no saving`
The file is too small or uniform to benefit from packing. The file can still be packed using the **–f** option.

file: `already packed`
The file appears to be a packed file. It can still be packed by specifying the **–f** option.

file: `has links`
The file has more than one link. You can override it with the **–f** option.

file: `directory`
pack cannot modify directories.

file: `empty`
The file is empty.

file: `can't pack in place`
The file is too large to pack in place. You must specify an output file using the **–o** option.

`Interrupt`
If you press `BREAK` while **pack** is running, it does not stop immediately; if it did, it would leave you with a corrupted file. Thus **pack** just displays this message to show that the `BREAK` has been received and it will stop as soon as it is safe to do so.

Other messages, such as those about inaccessibility of files, are self-explanatory. The exit status is the number of *file* arguments that could not be processed.

Portability

X/Open Portability Guide, UNIX System V.

The **–o** option is an extension of the POSIX standard.

Related Information

file, **pcat**, **unpack**

---

passwd — Change user passwords

Format

passwd [**–u** *userid*]

Description

passwd changes the login password for the user ID specified. If *userid* is omitted, the login name associated with the current terminal is used. You are prompted for the new password, which may be truncated to the length defined as the maximum length for the passwords.

Users can change the password for another user if they know the user ID and current password.

Examples

1. To change your password, issue:

   ```
   passwd
   ```

   You will be prompted for the old password and the new password.
2. To change the password for user ID Steve, issue:

   ```
   passwd -u steve
   ```

   You will be prompted for the old password and the new password.

Exit Values

`0` The password was changed.

`1` Failure due to any of the following:

- The user specified does not exist.
- The current password is incorrect.
- The new password does not meet the installation-exit requirements.

`2` The new password was not entered the same way twice.

`3` The password is too long.

`4` Error obtaining user login name.

paste — Merge corresponding or subsequent lines of a file

Format

paste [**–s**] [**–d** *list*] *file ...*

Description

paste concatenates lines of all the specified input files onto the standard output. If you specify – (dash) instead of a file, **paste** uses the standard input. Normally, an output line consists of the corresponding lines from all the input files. **paste** replaces the newline character at the end of each input line (except the one from the last file on the command line) with a tab character, or characters specified by the **–d** option.

Options

–d *list*
: Specifies a list of characters to be used one at a time instead of the tab character to replace the newline at the end of input lines. In a doublebyte locale, *list* can contain doublebyte characters. **paste** uses *list* circularly; when it exhausts the characters in *list*, it returns to the first character in the list. If you also specify the **–s** option, **paste** returns to the first character of *list* after processing each file. Otherwise, it returns to the first character after each line of output.

  list can contain any of the following standard C escapes such as `\n`, `\t`, `\r`, `\b`, `\\`, and `\0`, where `\0` indicates that no separator is to be used.

–s
: Concatenates all lines from each input file together on the single output line. If the **–s** option is not specified and the end of the file is detected on any (but not all) of the input files, **paste** behaves as though empty lines have been read from those files.

Examples

The command:

```
ls | paste –s –d'\t\t\n'
```

displays the output of **ls** in three tab separated columns.

If file A contains:

```
a
b
c
```

and file X contains:

```
x
y
z
```

then the command:

```
paste A X
```

produces:

```
a       x
b       y
c       z
```

and the command:

```
paste –s A X
```

produces:

```
a       b       c
x       y       z
```

Localization

paste uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion
1 Failure due to any of the following:
- Missing input files
- Too many files specified
- Inability to open a file

2 Unknown command-line option

Messages

Possible error messages include:

Too many files at *name*
You specified more files than **paste** can handle. The name given in the error message is the name of the first file that **paste** could not open. The number of files that **paste** can open depends on the number of files that other processes have open.

Portability

POSIX.2, X/Open Portability Guide, UNIX System V.

Related Information

cut

patch — Change a file using diff output

Format

patch [**–bcefIlNnRsv**] [**–B** *prefix*] [**–D** *symbol*] [**–d** *dir*] [**–F** *n*] [**–i** *patchfile*] [**–o** *outfile*] [**–p** *n*] [**–r** *rejectfile*] [*file*]

Description

patch reads a patchfile that contains output from **diff** describing changes from an old text file to a new text file. **patch** then applies those changes to another text *file*. Typically, you use **patch** if you are keeping parallel versions of a file. When you make a set of changes to one file, you can use **patch** to incorporate those same changes in other versions of the file.

Options

–B *prefix*
Saves a copy of the original *file* in a backup file. The backup filename is the name of the original file preceded by the string *prefix*. If there is already a file with this name, **patch** overwrites it. When applying more than one patch to the same file, **patch** copies only the original for the first patch. When you also specify **–o** *outfile*, **patch** does not create *prefixfile*, but if *outfile* already exists, it creates *prefixoutfile*.

–b Saves a copy of the original *file* in a backup file. The backup filename is the name of the original file plus the suffix `orig`. If there is already a file with that name, **patch** overwrites it. When applying more than one patch to the same file, **patch** only creates *file* `orig`. When you also specify **–o** *outfile*, **patch** does not create *file*-**.**`orig`, but if *outfile* already exists, it creates *outfile***.**`orig`.

–c Interprets the patchfile as a *context* diff file (the output of **diff** when **–c** or **–C** is specified). You cannot use this option with **–e** or **–n**.

–D *symbol*
Marks changes with the C preprocessor construct:

```
#ifdef symbol
      ...
#endif
```

When you compile the resulting (patched) file, you get the original file if *symbol* is not defined, and the changed file if *symbol* is defined.

–d *dir*
Changes the current directory to *dir* before processing the patch.

–e Interprets the patchfile as an **ed** script (the output of **diff** when **–e** is specified). You cannot use this option with **–c** or **–n**.

–F *n*
Specifies the number of lines of a context diff to ignore when searching for a place to install a patch.

–f Forces processing without requesting additional information from the user.

–i *patchfile*
Reads the patchfile information from the file *patchfile* . If you do not specify *patchfile*, **patch** reads the information from the standard input.

–l Matches any sequences of blanks in the patchfile to any sequence of blanks in the input *file*. In other words, **patch** considers two lines equivalent if the only difference between the two is their spacing.

–N Ignores any patches that have already been applied. By default, **patch** rejects already-applied patches.

–n Interprets the patchfile as *normal* **diff** output. You cannot use this option with **–c** or **–e**.

–o *outfile*
Writes patched output to *outfile* instead of to the original file. When you specify more than one patch to a single file, **patch** applies the patches to intermediate versions of the file created by previous patches, resulting in multiple, concatenated versions of the file being written to *outfile*.

–p *n*
Deletes *n* components from the beginning of all pathnames found in the patch file. If a pathname is an absolute pathname (that is, starts with a slash), **patch** treats the leading slash as the first component of the path, and **patch –p** 1 deletes the leading slash. Specifying **–p** 0 tells **patch** to use the full pathnames given in the *patchfile*. If you do not specify this option, **patch** only uses the basename (the final path component).

–R Reverses the sense of the patch script. In other words, **patch** behaves as if the patch script shows the changes that make the new version into the old version. You cannot use **–R** if the patchfile is in **ed** script format.

With **–R**, **patch** attempts to reverse each change recorded in the script before applying the change. **patch** saves rejected differences in reversed format (which means that you can check the rejections to see if **patch** made the reversals correctly).

–r *rejectfile*
Records rejects in the file *rejectfile*, instead of the default reject file name. Reject files are discussed later in this section.

–s Tells **patch** to remain silent until an error occurs. Normally, **patch** writes information on the results of the patching process to standard error (**stderr**).

–v Displays the version number of **patch** and then exits.

If you do not specify either the **–b** or **–B** option, **patch** attempts to change the original *file* directly. If you do not specify **–c**, **–e**, or **–n**, **patch** looks at the format of the **diff** output and tries to determine which type of output the patch file contains.

If you do not specify a file to be patched and the *patchfile* is not in context format, **patch** prompts you for the name of the file you want to patch.

If the *patchfile* is in context format, **patch** tries to determine the filename on its own. The first two lines of a context patch file give the names of the old and new files that **diff** compared. If only one of the files exists, **patch** patches that file; if neither exists or both do, **patch** checks for a line starting with a string `Index:` before asking you for the name of the file to patch. If both files exist but one of them is empty, the empty file will automatically be patched.

After **patch** has determined the file to patch, it checks for a source control system (SCCS) subdirectory in the current directory; if one exists, it tries to obtain an editable version of that file by performing the source code control system (SCCS)

command **get –e**. If **patch** cannot determine the file to patch, it prompts you for the name of the file to use.

With a context format *patchfile*, **patch** can recognize when line numbers given in the *patchfile* do not match line numbers in the file being patched. Thus, it can patch a file with line counts that do not match the old file that was used by **diff**. To do this, it takes these steps:

1. For each section to be changed, **patch** starts with the line number found in the patch file, plus or minus any adjustment that must be made for the previous section.
2. If the line at this location does not match the line in the patch file, **patch** scans forward and backward for a line that does match. If it finds a matching line, **patch** proceeds to make the required changes. **patch** also remembers the adjustment it had to make to find the matching line, and uses this adjustment in the next section to be changed.
3. If **patch** cannot find a line matching the one in the patch file, it tries to find the line using the lines given as context. It ignores the first and last two lines of the context and goes searching again. If it finds a match this time, it makes the change and remembers the adjustment.
4. If a search ignoring the first and last lines of the context fails, **patch** searches one more time, ignoring the first two and last two lines of the context. If it finds a match, it makes the changes and remembers the adjustment.
5. If **patch** still cannot find a match, it writes the unmatching portion to the *reject file*. It then tries to process the next section of changes. Thus, the reject file contains the sections that **patch** is not able to change. Line numbers on sections in the reject file may be different than those in the patchfile, because **patch** adjusts them using the adjustment that **patch** calculated for preceding sections.

To some extent, **patch** tries the same process if the patch file is in normal format rather than context format. Because the patch file does not contain the context information, **patch** has less to work with and probably creates more rejects. **patch** always writes the **rejectfile** in context format, regardless of the format of the *patchfile*.

By default, the reject file has the same name as the original *file*, plus the suffix `.rej`. You can use **–r** to specify a different reject file on the command line. If the reject file already exists, **patch** overwrites it.

If you do not specify **–R**, **patch** starts out with the assumption that the patch file could be normal or reversed. Therefore if the first change is rejected, **patch** tries the reverse change to see if that one works. If the reverse change is also rejected, **patch** continues with other changes in the file, trying both forward changes and reverses until one of them works. If the one that works is a forward change, **patch** attempts only forward changes for the rest of the file. If the one that works is a reverse change, **patch** issues a message to this effect and ask if it should treat all the changes as reverse ones. However, if the **–R** option is specified on the command line, it is assumed to hold for all changes in the patch file.

The patch file can contain output from several **diff** comparisons. **patch** treats each collection of changes as a separate patch file, and with each, **patch** may prompt you for the name of the file you want to patch.

Localization

patch uses the following localization variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_TIME**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion
1 There were one or more rejects
>1
An error occurred

Portability

POSIX.2 User Portability Extension, X/Open Portability Guide, UNIX systems.

The **–B**, **–F**, **–f**, **–s**, and **–v** options are not extensions to the POSIX standard.

Related Information

diff, **ed**

---

pathchk — Check a pathname

Format

pathchk [**–p**] *pathname ...*

Description

pathchk checks one or more pathnames (specified by *pathname*) for validity and portability (based on the underlying file system). A pathname is valid if you can use it to create or access a file without causing a syntax error. A pathname is portable if the file system does not truncate the name when it tries to use it. **pathchk** writes an error message indicating the error detected and the erroneous pathname if any pathname:

- Is longer than PATH_MAX bytes
- Contains a component longer than NAME_MAX bytes
- Contains any component in a directory that is not searchable
- Contains any character in any component that is not valid

Options

–p Instead of using the previous criteria, writes an error message if *pathname*:

- Is longer than _POSIX_PATH_MAX bytes
- Contains any component longer than _POSIX_NAME_MAX bytes
- Contains any character in any component that is not in the portable filename character set

Localization

pathchk uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 All pathnames passed the check
1 An error occurred
2 Unknown command-line option

Portability

POSIX.2, X/Open Portability Guide.

---

pax — Interchange portable archives

Format

- **pax** [**–cdnqvzE**][**–f** *archive*] [**–s** *substitute*] ... [*pattern ...*]
- **pax –r** [**–cdiknuvzE**] [**–f** *archive*] [**–o** *options ...*] [**–p** *string ...*] [**–s** *substitute ...*] [**–V** *volpat*] [*pattern ...*]
- **pax –w** [**–diLqtuvXzE**] [**–b** *blocksize*] [[**–a**] [**–f** *archive*]] [**–o** *options ...*] [**–s** *substitute ...*] [**–V** *volpat*] [**–x** *format*] [*pathname ...*]
- **pax –r –w** [**–diklLnquvXE**] [**–p** *string ...*] [**–s** *substitute ...*] [*pathname ...*] *directory*

Description

pax reads, writes and lists archive files. An *archive file* is a single file containing one or more files and/or directories. Archive files can be HFS files or MVS datasets. A file stored inside an archive is called a *component file*; similarly, a directory stored inside an archive is called a *component directory*.

Note: MVS datasets cannot be specified for component files.

Included with each component file and directory is recorded information such as owner and group name, permission bits, file attributes, and modification time.

You can therefore use a single archive file to transfer a directory structure from one machine to another, or to back up or restore groups of files and directories.

Archives created by **pax** are interchangeable with those created with the **tar** utility. Both utilities can read and create archives in the default format of the other (USTAR for **pax** and TAR for **tar**). Archives are generally named with suffixes such as .pax or .tar (or pax.Z and tar.Z for compressed files), but this is not required.

As shown in Format, **pax** performs one of the four archive functions based on the usage of the **–r** and **–w** options: list, read, write, or copy:

list If you do not specify **–r** or **–w**, you are in list mode. In this mode, **pax** uses the standard output to display the table of contents of an existing archive file. The **–v** (verbose) and **–E** options can be used to show the attributes and extended attributes of each component. By default, **pax** displays all component files and directories contained in the archive. One or more patterns may be used to display information on specific components.

read If you specify **–r** but not **–w**, you are in read mode. In this mode, **pax** reads an archive file as input and extracts components from the archive. By default, **pax** selects all components. Patterns may also be used to identify specific components to extract. If the archive contains several components with the same name, pax extracts each of them with later components overwriting files created by earlier components with the same name. The **–k**, **–n**, or **–u** options can be used to control the extraction of files when multiple files with the same name exist in the archive or on the file system.

When extracted, if a component does not have a fully qualified pathname beginning with the root (/) directory, its path is assumed to be relative to the current working directory. The **–s** or **–i** options can be used to dynamically change the pathnames of extracted components. Ownership, permissions, and extended attributes of the extracted files are discussed under the **–p** option.

write If you specify **–w** but not **–r**, you are in write mode. In this mode, **pax** creates an archive file that contains the specified pathnames as components. If a pathname is a directory, **pax** writes to the archive file all the files and subdirectories in that directory. If you do not specify any pathname, **pax** reads the standard input to get a list of pathnames to select; the input should give one pathname per line.

The **–d**, **–X**, and **–L** options can be used to restrict pathnames to the current directory or device, or to follow symbolic links.

The **–a** option can be used to append to an existing archive.

copy If you specify both **–r** and **–w**, you are in copy mode. In this mode, **pax** reads the specified pathnames and copies them to the target directory. In this case, the given directory must already exist and you must be able to write to that directory. If a pathname is a directory, **pax** copies all the files and subdirectories in that directory as well as the directory itself. If you do not specify any pathname, **pax** reads the standard input to get a list of pathnames to copy; the input should give one pathname per line.

The name of the archive file can be specified with the **–f** *archive* option. If **–f** is not used, **pax** will read from standard input for the list and read (**–r**) functions and will write to standard output for the write (**–w**) function.

pax can read input archives in **cpio** and **tar** format. It can also write these formats; see the **–x** option.

Patterns

Command-line patterns are similar to the wildcard constructs described in the **sh** command. You can use them to select specific components when reading or listing an archive.

Slash character(s) in a pathname must be explicitly matched by using one or more slashes in the pattern; it cannot be matched by the asterisk (*) or question mark (?) special characters or by a bracket expression. For example, the pattern "`*.c`" will only match files in the archive with name that are not preceded by a slash. The pattern "`*/*.c`" will match files in the archive preceded by a single slash.

Note: Patterns should be quoted to prevent the shell from first expanding them. For example, if the pattern `*.h` is not quoted, the shell will first resolve it into the list of files in the current directory ending with `.h`. If there are none, the shell will replace `*.h` with an empty list and **pax** will then list every component in the archive because no pattern is specified. If one or more `.h` files are returned by the shell, **pax** will list only those components in the archive matching the `.h` files found in the current directory.

pax does not support patterns when writing or copying — however, wildcards can be used in specifying the pathname with the write or copy function because the shell will first expand them before passing the results to **pax**.

The **–c** option can be used to select files that do not match the pattern.

Options

The following options can appear on **pax** command lines. Some of them are appropriate to only some forms of the command, as shown in “Format” on page 433.

–a Appends specified files or directories to the end of the contents of an existing archive. If the archive does not already exist, **pax** creates it.

Note: The following types of archives cannot be appended to:

- Compressed archives
- Archives residing in MVS partitioned datasets

–b *blocksize*
Specifies the block size in an output operation. Each output operation writes *blocksize* bytes, where *blocksize* is an integer appropriate to the output device. If `b` follows the *blocksize* number, the block size is the given number of 512-byte blocks. If `k` follows the blocksize number, the block size is the given number of 1024-byte blocks. The default *blocksize* is `10k` for **tar** archives, `5k` for **cpio** archives. The block size must be at least 512 bytes for reading.

–c Selects all those files that do *not* match any of the patterns given on the command line; this is the opposite of the usual behavior. If a pattern is not given, then no files will match.

–d Does not traverse directories. A pattern matching a directory extracts only the directory itself. When creating an archive, a directory name stores only the directory itself.

–E Same as verbose (**–v**) output, but additionally displays extended attributes. See "Output" on page 440 for more information.

–f *archive*

Lets you specify the name of the archive file instead of using the standard input for list mode, read mode (**–r** operations), and the standard output for write mode (**–w**). The archive file you specify may be an MVS dataset. For more information, see Appendix K, "Specifying MVS Dataset Names in the Shell Environment" on page 843.

Note: Avoid writing to an archive which is in the directory tree or the set of files being archived. Doing so causes **pax** to write the archive to itself and results in unpredictable results during the write or later during a read.

–i Lets you rename files as **pax** works. With extractions, **pax** displays the name of the component it is about to extract and gives you the chance to specify a name for the extracted file. With write operations, **pax** displays the name of the file or directory it is about to record in the archive, and lets you specify a different name to be assigned to the component. If you enter **.** as the name, **pax** processes the file or directory with no change to the name. If you just press <Enter>, **pax** skips the file (doesn't extract or archive it). **pax** ends if you enter end–of-file.

If you also specify **–s**, **pax** makes the given substitution before displaying the name of the component.

–k Prevents the overwriting of existing files.

–L Follows symbolic links. When you specify this option, **pax** copies the file to which a symbolic link points to the archive. Normally, only the symbolic link is copied.

–l Is applicable only when you are in copy mode—that is, when you are using the **–rw** format to copy files to another directory. If you specify **–l**, **pax** creates links to the original files whenever possible, rather than copying them.

–n Treats the *pattern* arguments as ordinary pathnames. You can use this option only when you specify **–r** but not **–w**. **pax** extracts only the first component with a given pathname, even if the archive contains several components with the same name. **pax** checks the given pathnames against the archive before applying any renaming from the **–i**, or **–s** options. **pax** writes an error message for each specified file that cannot be found in the archive.

–o *options*

Provides information for modifying the algorithm for writing and extracting files. This option is not valid when used with the **pax** copy (**–rw**) function.

There are two sets of options. The first set controls which code sets are to be used to represent component files when:

- storing them in an archive
- extracting them from the HFS
- creating them in the HFS

The second set of options controls the use of OS/390 extended support for the USTAR format to preserve and restore OS/390 specific information such as external links and extended attributes and other information (long link names, for example) not otherwise supported by the USTAR format:

–o *keyword=value[,keyword=value]...*

These options are used to convert data from one code set to another while reading or writing an archive. This is functionally equivalent to using the **iconv** utility to convert each file before or after archiving. This option has the format: where *keyword* is either **to** or **from** and *value* is the name of a code set. Two common code set names and their values are:

ISO8859-1 ASCII
IBM-1047 EBCDIC

For example, to convert from ASCII to EBCDIC, use:

```
-ofrom=ISO8859-1,to=IBM-1047
```

From EBCDIC to ASCII, use:

```
-ofrom=IBM-1047,to=ISO8859-1
```

For a more complete list of code sets, refer to *OS/390 C/C++ Programming Guide*.

Specifying an unknown keyword results in a warning message from **pax**.

You can omit either the **to** or **from** keyword. If you omit **to**, **pax** assumes that you want to write (or read) a portable archive tape and will convert the data to ISO/IEC 8859-1. If you omit **from**, **pax** assumes that you are converting from the system–specific local code set.

If your input contains a character that is not valid in the source code set, **pax** displays a warning and continues, leaving the character untranslated. If the source code set contains a character that is not in the destination code set, **pax** converts the character to an underscore (_).

Note: If you do not specify **–o**, no code set conversion is done. When making code set conversions, **pax** assumes that all files are text files, since only text files are portable.

–o *saveext* | *noext*

For USTAR formatted archives, this option controls whether extended USTAR support is enabled (*saveext*) or disabled (*noext*). *saveext* is the default behavior. This option has no effect for non-USTAR formatted archives. For more information on extended USTAR support, see "OS/390 Extended USTAR Support" on page 441.

This support is implemented with the USTAR format in such a manner that it should be tolerated by POSIX compliant **pax** or **tar** utilities on non-OS/390 UNIX systems. Those versions will not be able to restore the extended USTAR information, however, they should be able to process the archive as a normal USTAR formatted archive. To ensure an archive is portable to non-OS/390 UNIX systems, *noext* can be used.

saveext

During archive writing, *saveext* causes **pax** to preserve extended USTAR information. During archive listing, *saveext* causes **pax** to

display extended USTAR information. During archive reading, *saveext* enables **pax** to restore extended USTAR information. To restore certain information, the user must also have the appropriate privileges and have specified the corresponding options. For example, in order to restore extended attributes, **-px** must be specified.

noext
When creating archives, do not preserve extended USTAR information. When reading or listing an archive, ignore any extended support encoded within the archive. Should an archive contain OS/390 special header files, these will be displayed or restored (*special header files* are described in "OS/390 Extended USTAR Support" on page 441).

–p *string*
Specifies which file characteristics to restore. By default, **pax** will only restore the access and modification times of each component file. *string* can consist of any combination of the following characters:

a Does not preserve file access times.

e Preserves the user ID, group ID, file mode, access time, and modification time, and all extended attributes.

m Does not preserve file modification times.

o Preserves the user ID and group ID.

p Preserves the file mode.

x Preserves extended attributes.

–q For read mode only, **pax** assumes that all created files are text files and extracts them to the local text file format. On systems with fixed length records, this might mean appending blanks as padding.

On UNIX and POSIX-compliant systems, **pax** removes all carriage return characters (**\r**) and retains only the newline (**\n**) characters.

–r Reads an archive file from standard input.

–s *substitute*
Modifies pathnames using a substitution command *substitute*. This is similar to the substitution command of the **ed** text editor. The full option has the form:

```
–s /bregexp/string/[gp]
```

where *bregexp* is a basic regular expression and *string* is a string that **pax** is to insert in place of matches for the regular expression. *string* can contain an ampersand `&` (standing for the string matching *bregexp*), or `\1`, `\2`, and so on (with the meanings defined in **regexp**), for subexpression matching.

Normally, **–s** replaces only the first match for *bregexp*. A `g` following the *string* replaces all matches in the line.

A `p` following the *string* prints all successful substitutions on the standard error stream. **pax** displays a substitution in the format:

```
oldname >> newname
```

In this form of the command, the slash (/) is used as the character separating parts of *substitute*; you can use any nonnull character instead.

There may be more than one **–s** option on the command line. In this case, **pax** tries the substitutions in the order given. **pax** stops trying to make these substitutions as soon as it makes its first successful substitution. If the null string replaces a filename, **pax** ignores that filename on both input and output.

–t After reading files being archived, **pax** resets the access time to that prior to **pax**'s access.

–u Compares component dates to dates of existing files with the same name. When extracting components with **–r** (read mode), **pax** extracts a file only if its modification date is more recent than the modification date on an existing file of the same name. In other words, it doesn't overwrite an existing file if the existing file is newer than the one in the archive.

Similarly, when copying files with **–rw** (copy mode), **pax** does not overwrite an existing file if the existing file is newer than the one being copied.

In a command that uses **–w** but not **–r** (write mode), **–u** checks to see if the file being added has the same name as a file already in the archive. If so, and if the file being added is newer than the one in the archive, **pax** leaves the old file in the archive and appends the new one at the end. In this case, **–u** automatically implies **–a**, which means that **pax** adds new files to the end of the archive.

–V *volpat*

Provides automatic multivolume support. **pax** writes output to files the names of which are formatted with *volpat*. It replaces any occurrence of # in *volpat* with the current volume number. When you invoke **pax** with this option, it asks for the first number in the archive set, and waits for you to type the number and a carriage return before proceeding with the operation. **pax** issues the same sort of message when a write error or read error occurs on the archive; the reasoning is that this kind of error means that **pax** has reached the end of the volume and is to go on to a new one. An interrupt at this point ends **pax**.

–v Lists pathnames on the standard error stream just before beginning to process the files or directories, but after any **–i**, or **–s** options have had their effect. In list mode (neither **–r** nor **–w** is specified), **pax** displays a "verbose" table of contents; this verbose format shows information about the components in the same format used by the **ls** command. See "Output" on page 440 for more information.

–w Writes files to the standard output in the specified archive format.

–X Writes out only those files that are on the same device as their parent directory.

–x *format*

Specifies a file format for an output archive. The *format* argument can be:

`cpio` Standing for the ASCII format used by the **cpio** command.
`cpiob` Standing for the binary format used by **cpio**.
`tar` Standing for the old format of **tar** files.
`ustar` Standing for the (new) USTAR format used by the **tar** command.

The default *format* is `ustar`. For more information about file formats, see Appendix H.

Note: In order to preserve information about extended attributes, external links, and link names greater than 100 characters, the USTAR format must be used. See the **-o saveext** option for more information about preserving extended attributes.

–z For write or read mode, performs Lempel-Ziv compression. **–z** cannot be used when appending (**–a**) to an existing archive.

For writes, **–z** is functionally equivalent to creating a normal archive and then compressing it using the compress utility.

It is recommended that, when creating archive files using the **–f** option, the archive name be suffixed with a .Z to identify it as a compressed file and to facilitate it being processed by uncompress (if needed).

For reads, **–z** is functionally equivalent to first uncompressing the archive using the uncompress utility and then reading it. This option is not required when reading a compressed archive. **pax** will automatically detect that the archive is compressed. It may be useful, however, to use **–z** to confirm that the archive is compressed (you will receive an error message if you specify **–z** on an archive that is not compressed).

Output

When the **–v** or **–E** option is used in list mode, **pax** produces a verbose table of contents for the archive. The output for **–v** is equivalent to the output from the **ls–l** command with the following exceptions:

- The notation:

  pathname == linkname

  indicates that *linkname* is a hard link of *pathname*.
- For symbolic and external links, **pax** output always shows a filesize of 0.

Refer to the description of **ls** for an explanation of **ls –v**.

The output from the **–E** option has the same format as **–v** and additionally displays a column showing the extended attributes:

a	Program runs APF-authorized if linked AC=1
p	Program is considered program-controlled
s	Program runs in a shared address space
l	Program is loaded from the shared library region.
	Note: l is a lower-case L, not an upper-case i.
–	attribute not set

The format of the **pax –E** output is variable in length and will be extended as necessary to display additional file characteristics that are not supported by **pax –v** (**ls –l**).

Usage Notes

1. On the OS/390 system, superuser privileges or read access to the appropriate FACILITY classes are required to create character special files, restore user and group names, and to set certain extended attributes (read access to the corresponding FACILITY classes).

2. The POSIX 1003.1 standard defines formats for **pax**, **tar**, and **cpio** archives that limit the UIDs and GIDs that can be stored to the following maximum values:

Format	Max UID/GID Values
tar,USTAR	16777216
cpio	262143

   Values larger than these will not be properly restored for **tar** and **cpio** formatted archives. For USTAR formatted archives, because the user and group names are also stored in the archive, the correct UID and GID will be restored only if the name is defined on the target system.

3. The POSIX 1003.1 standard defines formats for **pax** and **tar** archives that limit the length of the target of a link file to 100 characters or less.

   Note: In the case of a hard link, the target is the first occurrence of the hard link which is archived. Subsequent hard links refer to the first instance.

   Beginning with OS/390 Release 9, **pax** and **tar** provide extended USTAR support that allows these links to be preserved when creating an archive and restored when reading an archive. See “OS/390 Extended USTAR Support” for more information.

4. When transferring archives between OS/390 and other UNIX systems, note the following:

   a. File transfers (for example, using OPUT/OGET or ftp put/get) must be done using binary or image format. This is true, even for archives consisting only of text files.

   b. You may need to convert text files from EBCDIC to ASCII (or some other character set). The **pax -o** option can be used to convert text files while an archive is being created or being restored. You can use the **iconv** utility to convert files before they are stored in the archive or after restoring them from an archive.

OS/390 Extended USTAR Support

By default, the IBM OS/390 implementation of **pax** and **tar** provide extended support with the USTAR format to preserve and restore OS/390 specific file attributes and other information not otherwise supported due to limitations with the USTAR format. Examples of these include:

- External links
- Extended file attributes (such as program-controlled, APF-authorized, etc.)
- Links whose targets exceed 100 characters

This support is only provided for archives using the USTAR format. USTAR is the default format for **pax** when creating an archive. For **tar**, the default format is the original **tar** format. The **-U** option, however, can be used to cause **tar** to use USTAR. When reading an archive, **tar** will automatically recognize the USTAR format– no special option is required. (For more information about the USTAR format, see "tar -- Format of tar archives" in Appendix H, “File Formats” on page 821.)

The extended USTAR support is provided by using two mechanisms: encoding the information within the USTAR header record and through the creation of special header files.

Encoding Information within the USTAR Header Record

External link and extended attribute information is encoded within the standard USTAR header in a manner which is compliant with POSIX standards and should be tolerated by other non-OS/390 versions of **pax** and **tar**. Because external links and extended attributes are specific to OS/390, however, they cannot be restored on other platforms.

Special Header Files

Hard links and symbolic links with targets greater than 100 characters cannot be preserved within the standard USTAR format (for a hard link, the target is the first occurrence of the hard link which is archived; subsequent hard links refer to the first instance). In order to preserve links with targets greater than 100 characters, *special header files* are created for each link and stored in the archive.

Each special header file contains information used by OS/390 **pax** and **tar** to restore the link to it's original state. Special header files are identified in the archive with type "S" (see "tar -- Format of tar archives" in Appendix H, “File Formats” on page 821 for more information about file types).

Each special header file in the archive will have the same name: **/tmp/OS390_USTAR_SUMMARY**_*timestamp* where *timestamp* is the creation time (represented in seconds since the epoch) of the first special header file. For example:

```
/tmp/OS390_USTAR_SUMMARY_919026474
```

When a special header file is required to preserve a file, an informative message along with a reason is displayed indicating that a special header file was created.

When reading or listing an archive containing special header files and when using the default extended USTAR support, **pax** and **tar** recognize type "S" files as special header files and display or restore the file described by the special header rather than the actual special header file. So, typically, the presence of special header files is not known to the user.

When the archive is complete, if one or more special header files have been created, then a final *special header* **summary** *file* is created and added to the archive. This file is identified in the archive with type "T" and has the same name as the special header files. This file summarizes, via script commands and comments, the contents of all previously archived special header files. It's primary purpose is to provide help restoring files included via special header files to those with versions of **pax** or **tar** that do not implement extended USTAR support.

So, to allow users of non-OS/390 systems to read the *special header* **summary** *file*, it is encoded in the ASCII ISO8859-1 codepage. To view the special header file in EBCDIC code page IBM-1047, first convert the file using the **iconv** command. For example:

```
 iconv  -f ISO8859-1  -t IBM-1047  /tmp/OS390_USTAR_SUMMARY_919026474  >
summary_in _ebcdic
```

If extended USTAR support is disabled during reading or listing an archive by using the **pax -o** *noext* or the **tar -O** option, or if the archive is processed by either an earlier version of OS/390 **pax** or **tar** that does not implement extended USTAR support or a non-OS390 version of **pax** or **tar**, then the special header files will not

be recognized and will be processed as unknown type regular files. During extraction, because all files have the same name, each extracted special header file will overlay the previous one with the special header summary file being the final one restored.

For an example of the *special header* **summary** *file*, see "pax — Format of pax archives and special header summary files" on page 824.

Examples

Writing (creating) an archive:

1. The following creates an archive file named **/tmp/files.pax** from all the files in the current working directory. The **-v** option is used to display each file as it is being added:

   ```
   pax -wvf /tmp/files.pax *
   ```

   or

   ```
   pax -wvf /tmp/files.pax .
   ```

   The only difference between these two forms is that in the latter example (using .), names recorded in the archive will be preceded by a "./".

2. Either of these commands creates a compressed version of the archive created in Example 1:

   ```
   pax -wzvf /tmp/files.pax.Z  *
   ```

   or

   ```
   pax -wzvf /tmp/files.pax.Z  .
   ```

   The above is equivalent to the following two commands:

   ```
   pax -wvf /tmp/files.pax *
   compress /tmp/files.pax
   ```

3. The following creates an archive **/tmp/dironly.pax** containing only the files and directory names in the current directories (it does not include the contents of subdirectories):

   ```
   pax -wdvf /tmp/dironly.pax.  *
   ```

4. This example creates the archive **/tmp/cfiles.pax** containing all c files in the current directory:

   ```
   pax -wvf /tmp/cfiles.pax  *.c
   ```

5. This example creates the archive **/tmp/allcfiles.pax** containing all c files in the current directory and all sub-directories:

   ```
   pax -wvf /tmp/allcfiles.pax  $(find . -name "*.c")
   ```

6. This example creates the archive **/tmp/ascii_src.pax** using all .c and .h files in the current directory converted into ASCII:

   ```
   pax -wv -o to=ISO8859-1 -f /tmp/ascii_src.pax  *.[ch]
   ```

7. The following creates the compressed archive **/u/smith/oldfiles.pax.Z** containing all files on the system that have not been accessed within the last 10 days:

   ```
   pax -wvzf /u/smith/oldfiles.pax.Z   $(find / -type f -atime +10)
   ```

8. The following creates the archive **/tmp/basename.pax** containing all files in the directory **sub1** stored in the archive with "sub1/" removed from each component name. Note that the pound character # is being used as the delimiter for the **–s** option:

   ```
   pax -wv -s#sub1/## -f /tmp/basename.pax  sub1/*
   ```

Reading an archive:

1. This example extracts all the components of the archive **source.pax**. The **–v** option is used to display each file or directory as it is extracted.

   ```
   pax -rvf source.pax
   ```

2. To extract all files in **source.pax** and translate them from ASCII to EBCDIC:

   ```
   pax -ofrom=IS08859-1,to=IBM-1047 -rf source.pax
   ```

3. To extract all files in the archive **source.pax** ending with **.h**:

   ```
   pax -rf source.pax  `pax -f source.pax | grep h$`
   ```

   This example uses command substitution to first read the archive and generate a list of all files in the archive that end with **/**.

4. This example extracts files into a directory that is different from the directory they are stored in within the archive. Assume the names of all files stored in the archive **source.pax** begin with the root directory (**/**). To extract them into **/newroot/**, use the following command:

   ```
   pax -rvf source.pax -s#/#/newroot/#
   ```

   The **–v** option is used to show the names of the files as they are extracted and is not required.

Files

/tmp/OS390_USTAR_SUMMARY_*timestamp*
: is an OS/390 extended USTAR special header file. See “OS/390 Extended USTAR Support” on page 441 for more information.

Localization

pax uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_TIME**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion

1 Failure due to any of the following:

- Incorrect option
- Incorrect command-line arguments
- Out of memory
- Compression error
- Failure on extraction
- Failure on creation

If **pax** cannot extract a particular file when reading, or find a particular file when writing, it generates an error message and continues to process other files but returns a status of 1. If any other sort of error occurs, **pax** ends immediately without attempting further processing.

If you see the following message after a write operation:

```
If you want to go on, type device/filename when ready
```

it indicates that your directory or device containing the archive file is full. To continue, enter the name of a new directory; to end **pax**, type <Ctrl-C>.

If you see that message after a read operation, it means that **pax** could not find the archive file you specified, or that it was damaged. In this case, type <Ctrl-C> to end the operation and then restart **pax** with the correct archive name.

Portability

POSIX.2, X/Open Portability Guide.

The **–L**, **–q**, **–V**, **–E**, **–p x** and **–z** options are extensions of the POSIX standard.

Related Information

compress, **cpio**, **ls**, **tar**, **uncompress**

See Appendix C for more information about **regexp**.

See the **cpio** and **pax** file format descriptions in Appendix H, “File Formats” on page 821.

---

pcat — Unpack and display Huffman packed files

Format

pcat *file ...*

Description

pcat uncompresses files that were compressed by **pack** using a Huffman minimal redundancy code. The uncompressed data is sent to the standard output. This is handy for packed text files, but inappropriate for binary files, because the standard output is treated as a text stream. Binary files can be decoded in place by **unpack**.

The names of compressed input files are expected to end in `.z`. If a specified input file name does not end in this suffix, **pcat** automatically adds the `.z`. For example, if the command line specifies file **abc**, **pcat** looks for `abc.z`.

Localization

pcat uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

`0` Successful completion

`n` Indicates that `n` files could not be unpacked properly. For example, if three out of six files could not be unpacked properly, the exit status is `3`.

Related Information

cat, **file**, **pack**, **unpack**

---

pg — Display files interactively

Format

pg [**–cefnst**] [**–p** *prompt*] [**–** *screen*] [**+***line*] [**+**/*pattern*/] [*file ...*]

Description

pg displays input files or piped output from another command, a screenful at a time. If you do not specify any files, the standard input (**stdin**) is read. Any file named **–** specifies the **stdin**.

“Commands” on page 447 lists commands that can be entered at page and file breaks.

Options

–c Clears the screen before displaying each new window.

–e Eliminates the `(EOF):` prompt at the end of each file.

–f Does not fold lines. Usually, lines longer than the screen width, as given by the environment variable **COLUMNS** are folded into multiple lines.

This option may be useful for files containing device-specific escape sequences.

–n Executes interactive commands immediately after receiving the command character. This works for most commands. Usually, you must press <Enter> for interactive commands.

–p *string*
Sets the prompt string that appears at the end of each screen of text to *string.* The default prompt is a colon (**:**). If *string* contains the characters `%d`, **pg** replaces those characters with the current page number as in [Page %d].

–s Displays all interactive command prompts in standout mode (most often reverse video) on the screen.

–t Does not save input in a temporary file. Usually, if any of the inputs is not directly seekable (as is the case for a serial device or pipe), **pg** reads input and saves it in a temporary file so that it can be reviewed. Because of this, you cannot scan backwards when viewing such input. This option is also recommended when reading a larger amount of data from a stream that cannot be accommodated on disk.

–*screen*
Sets the number of lines displayed in each screen to *n* lines. If you do not select this option, the number of lines displayed is one less than the number of lines on the screen as given by the environment variable **LINES**. "Commands" discusses the **w** command.

+*line*
Starts printing at line *n* of the first file. The default is to start printing at line 1.

+/*pattern***/**
Starts printing at the line containing the first occurrence of the extended regular expression *pattern*.

This is described in Appendix C.

Commands

Depending on the options you specify, **pg** pauses between windows (screenfuls) of text, at the end of each file and before starting any file after the first. At these pauses, **pg** prompts you to enter a command. To read the file, type the command ENTER (newline or Return) at each prompt.

An optional sign (**+** or **–**) followed by an optional numeric address can precede the following commands. Addresses work in multiples of screen displays: for example, an address of `+2` displays the second next screenful. Usually, an unsigned address implies direct addressing (measured from the beginning of the file). A signed address implies relative addressing in the file; a command beginning with a **+** scans forward and one beginning with a **–** scans backward from the current position.

You can edit commands interactively with the standard erase and kill characters.

These are the interactive commands:

h Prints a summary of the interactive commands.

q, Q
Exits immediately from **pg**.

!*command*
Executes the string *command* as if it were typed to the default command interpreter (as in **ed**). Whether or not you specified the **–n** option, you must end this command with a newline.

[[±]*n*] ENTER, [[±]*n*] SPACEBAR
Without a specified address, displays the next window of text. With an address, displays the *n*th next window of text.

[[±]*n*]d, [[±]*n*]CRTL–D
Scrolls a half screen of text. The address is measured in half screenfuls and defaults to the next half screen.

[[±]*n*]l
With no address, displays the next line of the file. With an address, it displays a screenful starting at the addressed line.

$ Displays the last screenful of text in the file.

<**Ctrl-L**>**, .**
Redisplays the current displayed window of text.

s *file*
Saves the entire contents of the current file in *file*. Whether or not you specified the **–n** option, you must end this command with a newline.

[*n***] n**
Displays the first screenful of the next file. The address (*n*) is actually the *n*th next file, counting from the current file. If present, *n* must be unsigned.

[*n***] p**
Displays the first screenful of the previous file. The address (*n*) is actually the *n*th previous file, counting from the current file. If present, *n* must be unsigned.

[*n***] w**
Scrolls another window of text. The argument, *n* (which must be unsigned), sets the window size to *n* and displays the next window of text.

[*i***]/***pattern***/[tmb]**
Searches forward within the current file for the *i*th next occurrence of a line matching the regular expression *pattern* (default *i* is 1, the next matching pattern). The search starts right after the current window and continues to the end of the file. Usually, the matching line is displayed at the top of the window, but this can be changed by an optional character at the end of the search command. The letter `t` is the default and displays the line at the top of the window, `m` displays it in the middle of the window, and `b` displays it in the bottom of the window. When no letter is present, **pg** uses the last letter entered (or `.t` if no letter has been entered). Whether or not you specified the **–n** option, you must end this command with a newline.

[*i***]?***pattern***?[tmb], [***i***] *pattern* [tmb]**
Is similar to the previous command, but searches backward instead of forward. The search starts just before the current window.

Examples

The following interactive commands illustrate the flexibility of **pg**. Suppose you enter the command:

```
pg –n *.c
```

and that there are a large number of source files in the current directory:

`1` Redisplays the first screenful of the current file.

–4 Goes back 4 windows in the current file and displays a screenful of text.

p Displays the first screenful of the previous file.

10w Sets the screen size to 10 lines.

/Fred/m
Finds the first line containing

Fred

searching forward from the current position in the file, and displays a screen with that line in the middle of the screen.

Localization

pg uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion

1 Failure due to any of the following:

- Unknown command-line option
- Insufficient memory
- Inability to create a temporary file
- Inability to access the terminal
- Missing *string* after a **–p** option

Files

pg uses the following file:

$TMPDIR/pg*
Temporary files to allow backward reading. You can specify a different temporary directory using the **TMPDIR** environment variable.

Environment Variables

pg uses the following environment variables:

COLUMNS
Contains the width of the screen in columns.

LINES Contains the number of lines on the screen.

TMPDIR Contains the pathname of the directory where temporary files reside.

Portability

X/Open Portability Guide, UNIX System V.

This implementation does not handle doublebyte characters.

The **–screen** and **–+line** options are extensions to the XPG standard.

Related Information

alias, **ed**, **head**, **more**, **sh**, **tail**, **vi**

See Appendix C for more information about **regexp**.

---

pr — Format a file in paginated form and send it to standard output

Format

pr [**–adFfprtW**] [**–n** | **–c** *n* | **–m**] [**–e** [*char*[*gap*]] [**–H** *header-fmt*] [**–h** *header*] [**–i**[*char*] . [*gap*]] [**–l** *n*] [**–n**[*char*] [*n*]] [**–o** *n*] [**–s**[*char*]] [**–w** *n*] [**+***n*] [*file ...*]

Description

pr prints the specified files on standard output (**stdout**) in a paginated form. If you do not specify any *file*s or if you specify a filename of –, **pr** reads the standard input. By default, **pr** formats the given files into single-column 66-line pages. Each page has a five-line header. By default, the third line contains the file's pathname, the date it was last modified, and the current page number; the other lines are blank. A five-line trailer consists of blank lines.

If you specify multiple columns, **pr** places its output in columns of equal width separated by at least one space, truncating each line to fit in its column. Input lines can be ordered down the columns or across the page on output; or different columns can each represent different files.

Options

+*n* Starts printing with the *n*th page of each file; that is, skips the first *n*–1 pages. The default for *n* is 1.

–*n* Prints *n* columns of output. When you specify this option, **pr** behaves as though you had also specified the **–e** and **–i** options. When you specify both this option and **–t**, **pr** uses the minimum number of lines possible to display the output. Do not specify this option with the *–m* option.

–a Orders input lines across the page on output, instead of down. You should use this option only with **–n**.

–c *n*
Displays *n* columns of output. When you specify this option, **pr** behaves as though you had also specified the **–e** and **–i** options. When you specify both this option and **–t**, **pr** uses the minimum number of lines possible to display the output. Do not specify this option with **–m**.

–d Produces double-spaced output.

–e[*char***][***gap***]**
Expands each occurrence of the input tab character to a string of spaces so that the following character has the next column position which is a positive multiple of *gap*, plus 1. If you do not specify *gap*, or if it is zero, **pr** assumes that *gap* has the value of 8. If you specify the nondigit character *char*, **pr** treats it as the input tab character. Otherwise, **pr** uses the standard tab character.

–F Uses form feeds to separate pages. **pr** normally separates pages by sending a series of <newline> characters to fill the length of a page.

–f Uses form feeds to separate pages. When output is to a terminal, **pr** sounds the bell and waits for you to type a carriage return before displaying the text. **pr** normally separates pages by sending a series of <newline> characters to fill the length of a page.

–H *header_fmt*
Lets you customize your header line by specifying a format with the string *header_fmt*. **pr** recognizes the following special formatting commands:

`%c` Date and time
`%F` Current filename, or *header* string given by **–h**
`%P` Page number
`%L` Line number
`%D` Date
`%T` Time
`%u` Current user name

The default header format is equivalent to the option:

```
-H "%c  %F  Page %P"
```

–h *header*
Uses the *header* string instead of the filename on each succeeding page header.

–i[*char***][***gap***]**
Replaces white space with tabs on output. *char*, if given, is the output tab character. The default is the tab character. **pr** sets tabs every *gap* positions; the default for *gap* is 8. If this tab character differs from the input tab character and the actual data contains this tab character, the result is liable to be quite a mess.

–l *n* Sets the number of lines per page of output. The default is 66. The actual number of lines printed per page is this number less 5 for the header and 5 for the trailer. If *n* is less than 10 (the number of lines needed for the header and the trailer), **pr** displays neither the header nor the trailer.

–m Prints each file in its own column down the page. This overrides the **–a** option, forcing the **–n** option to be the number of files given. When you also specify the **–n** option, it gives line numbers for the first column only.

–n[*char***][***n***]**
Numbers the lines of each file. Each number takes up *n* positions; the default for *n* is 5. The character *char* separates the number from the line; this defaults to the tab character. If *char* is the same as the input tab character, **pr** follows the number with the spaces needed to get to the next tab stop. **pr** may in turn replace these spaces with the output tab character if you specified the **–i** option. For multicolumn output, **pr** adds line numbers to each column. The **–m** option gives the line number for the first column only.

–o *n*
Offsets each line of output by *n* character positions.

–p Pauses before the beginning of each page if output is to a terminal device. **pr** sounds the bell and waits for a carriage return from the controlling workstation (not the input files).

–r Suppresses error messages due to failures when opening files.

–s[*char***]**
Prints each column at its correct length. The character *char* separates columns. The default value for *char* is the tab character. This character is never replaced by the output tab character. Normally **pr** pads each column with spaces or truncates it to the exact column width. Unless the **–w** option is also used, **–s** resets the page width to 512 column positions.

–t Does not print the headers and trailers, and quits after the last line of the file—it does not display any extra lines.

–W Folds lines at the column width when you do not specify the **–s** option; **pr** treats each separate part of the line as a separate line.

–w *n*
Sets the width of the page to *n* column positions. If you do not specify this option, the default page width is 72 (if you did not specify **–s** option) or 512 (if you did specify **–s**). This page width does not normally apply to single-column output; however, single-column output with the **–W** option does use this width.

Files

pr uses the following file:

/dev/tty For prompting.

Environment Variables

pr uses the following environment variable:

TZ Contains the local time zone. **pr** uses this value when displaying times in header lines.

Localization

pr uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_TIME**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion

1 Failure due to any of the following:

- Insufficient memory
- Insufficient line width
- Write error on **stdout**

2 Syntax error or unknown command-line option

Messages

Possible error messages include:

Missing header
You specified **–h** or **–H** but did not supply a *header* or *header_fmt* string.

Width is insufficient
The line is not wide enough to hold the given number of columns with the given column width; or a column is not wide enough to hold the minimum amount of data.

Portability

POSIX.2, X/Open Portability Guide.

The **–c**, **–H**, **–p**, and **–W** options are extensions of the POSIX standard.

In a doublebyte environment, remember that column positions are always based on the width of characters. A doublebyte character may take up two columns of output (called a *thick character*), but a singlebyte character will only take up one column of output (called a *thin character*). Specify column widths according to the expected thickness of characters.

For example, with a column width of 10, then ten thin characters or five thick characters are displayed.

Related Information

cat, **expand**, **fold**, **unexpand**

Appendix I also explains how to set the local time zone with the **TZ** environment variable.

---

print — Return arguments from the shell

Format

print [**–npRrs**] [**–u**[*descriptor*]] [*argument ...*]

Description

Calling **print** without options or with only the **–** option displays each *argument* to the standard output using the same escape conventions as **echo**. In this case, **print** and **echo** work the same way; see **echo**.

Options

The options accepted by **print** increase its utility beyond that of **echo**.

–n Does not automatically add a new line to the end of the output.

–p Sends output to a coprocess.

–R Is similar to **–r**, except that **print** treats all subsequent options (except **–n**) as arguments rather than as options.

–r Ignores escape conventions.

–s Appends the output to the command history file rather than sending it to standard output.

–u[*descriptor***]**
Redirects the output to the file corresponding to the single digit file *descriptor*. The default file descriptor is `1`.

Usage Note

print is a built-in shell command.

Localization

print uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

`0` Successful completion
`1` Failure due to any of the following:
- Incorrect *descriptor* specified with **–u**
- Nonexistent coprocess

`2` Failure due to an incorrect command-line option

Messages

Possible error messages include:

Cannot print on file descriptor ...
You tried to print on a file descriptor that was not opened for writing.

History not available
You specified the **–s** option to write into a history file, but you are not now using a history file.

Portability

print is an extension to POSIX.2 and XPG.

Related Information

echo, **fc**, **read**, **sh**

---

printenv — Display the values of environment variables

Format

printenv [**name**]

tcsh shell: **printenv** [**name**]

Description

The **printenv** command displays the values of environment variables. If the *name* argument is specified, only the value associated with *name* is printed. If it is not specified, printenv displays the current environment variables, one *name=value* pair per line.

If a *name* argument is specified but is not defined in the environment variable, **printenv** returns exit status `1`; otherwise it returns status `0`.

printenv in the tcsh shell

In the tcsh shell, **printenv** prints the names and values of all environment variables or, with *name*, the value of the environment variable named. See "tcsh — Invoke a C shell" on page 556.

Options

There are no options.

Example

To find the current setting of the **HOME** environment variable, enter:

```
printenv HOME
```

Usage Notes

1. Only one *name* argument can be specified.
2. **printenv SOMENAME** is equivalent to **echo $SOMENAME** for exported variables.
3. **printenv** without any arguments is functionally equivalent to **env** without any arguments.

Exit Values

`0` Successful completion

`1` Failure due to one of the following:
- More than one environment variable was specified
- An option was specified (**printenv** has no options)

Portability

printenv is compatible with the AIX **printenv** utility.

Related Information

env, **tcsh**

---

printf — Write formatted output

Format

printf *format* [*argument ...*]

Description

printf writes the *argument* operands to standard output, formatted according to the *format* operand.

format is a format string composed of conversion specifications that convert and add the next *argument* to the output. *format* can contain backslash-escape sequences. These conversions are similar to those used by the ANSI C standard. Conversion specifications have the form:

`%`[*flag*] [*width*]
[*precision*] [*char*]

where *flag* is one of the following:

– Left-justifies the field; default is right justification.

\+ Always prefixes a signed value with a sign (+ or –).

`space` Reserves a character position at the start of the string for the minus sign (for negative numbers) or a space (for positive numbers). If both `space` and – appear as flags, the `space` flag is ignored.

`#` Prefixes octal values with `0` and hexadecimal values with `0x` or `0X`. For floating-point values, this causes the decimal point always to be displayed even if no characters follow it.

`0` Pads numeric values with leading zeros. If both `0` and – appear as flags, the `0` flag is ignored.

width is the minimum field width of the output field. If the converted value is shorter than the minimum width, **printf** pads it with spaces or zeros.

In a string, *precision* is the maximum number of bytes to be printed from the string; in a number, the precision is the number of digits to be printed to right of the decimal point in a floating-point value. *width* or *precision* can be specified as `*`, in which case the value is read from the next argument, which must be an integer. For example:

```
printf "%*.*d\n" 20 10 200
```

is equivalent to:

```
printf "%20.10d\n" 200
```

The conversion character *char* is one of the following:

b A string that may contain a backslash-escape sequence. Valid escape sequences are described in
c Single character of an integer value; the first character of a string.
d Decimal integer.
e,E
Floating point (scientific notation).
f,F
Floating point.
g,G
The shorter of e and f (suppresses nonsignificant zeros).
i Decimal integer.
o Unsigned octal integer.
s String.
u Unsigned decimal integer.
x,X
Unsigned hexadecimal integer.

When there are more arguments than positions in *format*, the *format* string is applied again to the remaining arguments. When there are fewer arguments than there are positions in the *format* string, **printf** fills the remaining positions with null strings (character fields) or zeros (numeric fields).

Caution

The POSIX.2 **printf** facility (like the C language **printf()** on which it is based), does not accommodate doublebyte characters gracefully when using **%c** conversion, or either of **%b** or **%s** conversions with a specified precision. Use these features cautiously when you have doublebyte characters in the character set.

In a doublebyte environment, normal backslash-escape characters are handled correctly—**printf** shifts state as required—but octal and hexadecimal escape characters do not change state. This is significant in a shift-lock environment. For example, if an octal escape character contains the shift-in character, it is the user's responsibility to ensure that there is also a shift-out character. Further, an octal or hexadecimal backslash escape character that comes immediately after a doublebyte character may or may not be processed in the shifted state.

For more information on doublebyte character environments, see “Using the Doublebyte Character Set (DBCS)” on page 7.

Localization

printf uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_NUMERIC**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Values

`0` Successful completion

`>0`
The number of failures due to any of the following:

- Missing format specifications
- Arguments supplied for a *format* string that does not accept them (that is, that has no `%`s)
- Incorrect integer argument
- Incorrect floating-point argument

Portability

POSIX.2, X/Open Portability Guide, UNIX System V.

The **%F** format and the handling of `*` as a width or precision argument are extensions of the POSIX standard.

Related Information

echo, **print**

---

ps — Return the status of a process

Format

ps [**–Aacdefjlm**] [**–G** *idlist*] [**–g** *grouplist*] [**–n** *name*] [**–o** *format*] ... [**–p** *proclist*] [**–s** *idlist*] [**–t** *termlist*] [**–U**|**u** *uidlist*]

Description

ps displays status information about processes, and optionally, the threads running under each process.

By default, for each process associated with the user's terminal, **ps** will display the process id (PID), TTY, processor time used (TIME), and name of the command (COMM).

The **–a**, **–A**, and **–e** options can be used to show information associated with all available or accessible processes on the system. However, these options can only show information for those processes the user has appropriate privileges to access. The **–g**, **–G**, **–p**, **–s**, **–t**, **–u**, and **–U** options can be used to select specific processes by process id, terminal id, and user name.

The **–f**, **–j**, and **–l** options can be used to display additional status fields using predefined formats. The **–o** format option allows the user to select specific status fields and to define the format in which these fields are displayed.

ps will display information for each thread associated with a process when the **–m** and/or **–o THREAD** options are used. Output lines for thread information immediately follow the output line for the parent process. Since the default behavior of **ps** displays process status fields only, to provide meaningful thread output, the **–o** option is used to specify thread specific status fields. There are some conditions,

such as when the process is in a terminating or zombie state, where thread data cannot be captured. In these cases, a single thread output line will be displayed showing a ? in the thread output fields.

Options

ps accepts several options. When a description says that **ps** lists "all processes", it means all the processes on the system, provided that you have appropriate privileges.

The fields `pcpu`, `nice`, `pri`, `addr`, and `wchan` are unsupported and will always display a dash.

–A Displays information on all available processes. You can specify **–A**, **–a**, and **–e** in any combination; however, **–a** overrides both **–A** and **–e**.

–a Displays information on all processes associated with terminals. You can specify **–A**, **–a**, and **–e** in any combination; however, **–a** overrides both **–A** and **–e**.

–c Displays more detailed information about processes for the **–f** and **–l** options. **–c** is accepted but not currently implemented.

–d Displays information for all processes except group leaders.

–e Displays information on all accessible processes. You can specify **–A**, **–a**, and **–e** in any combination; however, **–a** overrides both **–A** and **–e**.

–f Displays information as if the user specified:

```
–oruser=UID –opid,ppid,pcpu=C –ostime,tty=TTY –oatime,args=CMD
```

–G *grouplist*
Displays information on processes with real group ID numbers in *grouplist*. Separate numbers in *grouplist* with either blanks or commas.

–g *idlist*
Displays information on processes with process ID numbers in *idlist*. Separate the numbers in *idlist* with either blanks or commas.

–j Displays information as if the user specified:

```
-o pid,sid,pgid=PGRP -o tty=TTY -o atime,args
```

–l Displays information as if the user had specified:

```
–oflags,state,ruid=UID –opid,ppid,pcpu=C –opri,nice,addr,vsz=SZ
–owchan,tty=TTY –oatime,comm=CMD
```

–m Displays thread status information. Output lines for thread status immediately follow the output line for the parent process. Process-only status fields will contain dashes for thread output lines. Since the default behavior of **ps** is to display process-only status fields, to provide meaningful thread output, the **–o** option should be used to specify thread supported status fields. If **–o THREAD** is used, **–m** is assumed.

–n *name*
Specifies the name of the executable file containing the kernel symbol table.

–o *format*
Displays information according to the given *format* specifications. If **–o** is not used, the default format is the same as specifying:

```
–o pid,tty=TTY –o atime,comm
```

See “Format Specifications” on page 460.

–p *proclist*
Displays information for processes with process ID numbers in *proclist*. Separate numbers in *proclist* with either blanks or commas.

–s *idlist*
Displays information for processes with session ID numbers in *idlist*. Separate the numbers in *idlist* with commas.

–t *termlist*
Displays information for processes with terminals in *termlist*. You denote terminals in *termlist* with either the filename of the device (for example, `tty04`). Or, if the filename begins with `tty`, you can simply specify the characters following `tty`. For example, `tty04` and `04` both denote the same terminal. Terminals in *termlist* are separated by either blanks or commas.

–U *userlist*
Displays information for processes with user IDs in *userlist*. Items in *userlist* can be user ID numbers or login names, and are separated by commas.

–u *userlist*
Displays information for processes with user IDs in *userlist*. Items in *userlist* can be user ID numbers or login names, and are separated by commas.

Format Specifications

Using the **–o** option, the user can define the status fields that will be displayed and their column headings. If you do not specify the **–o** option, **ps** displays the information as though you specified:

```
-o pid,tty=TTY -o atime,comm
```

The *format* specification is a list of status field names separated with blanks or commas. However, if the list of names is separated by blanks, the list must be contained in single quotes. Below you'll find a list of status field names recognized by **ps**.

Multiple **–o** format specifications can be provided and, in the case where user-specified column headings are defined, these specifications may be necessary.

The first line of **ps** output contains column headings for each status field. Each status field has a default heading which can be overridden by the user by specifying `=newheading` after the status field. When a new heading is specified, it must be the last field given on the **–o** option. To specify additional fields, it is necessary to use additional **–o** statements.

For example, if you wish to display the process id (pid), real user name (ruser), and command name (comm), but change the heading for the real user name from the default of (`RUSER`) to `WHO`, use:

```
-o pid,ruser=WHO -o comm
```

An additional **–o** is required when `comm` is specified because the last argument must be user-specified headings (in this case ruser=`WHO`).

If you specify `=` with no heading, **ps** displays that column without a heading. If none of the columns have a heading, **ps** displays no heading line.

In a doublebyte locale, user-defined headings may contain multibyte (doublebyte) characters.

The following list shows the names that **ps** recognizes. The list is separated into three groups:

process only These are fields which only display meaningful data for process output lines. For thread output lines, a dash is shown in these fields.

thread only These are fields which only display meaningful data for thread output lines. For process output lines, a dash is shown in these fields.

processes and threads
These are fields that apply to both processes and threads. For example, `state` is meaningful because both processes and threads have a state that can be determined for them.

At the end of each description, we put the default column heading inside square brackets.

Process Only

`addr` Displays the address of the process. This field is currently not supported and will display a dash. [ADDR]

`args` Displays the command that is running, with all its arguments. [COMMAND]

`atime` Displays the amount of processor time that the process has used since it began running. Time is displayed in the format ss:nn where ss is the number of seconds and nn is the hundreths of seconds. [TIME]

`comm` Displays the name of the command that is running without its arguments. This string is padded on the right if necessary. [COMMAND]

`etime` Displays the amount of real time that has elapsed since the process began running. **ps** shows the time in the form:

[[*dd*]*hh*]*mmss*

where *dd* is the number of days, *hh* is the number of hours, *mm* is the number of minutes, and *ss* is the number of seconds. [ELAPSED]

`gid` Displays the effective group ID of the process. [EGID]

`group` Displays the effective group ID of the process, as a group name if possible and as a decimal group ID if not. [GROUP]

`jobname` Displays the OS/390 jobname. [JOBNAME]

`nice` Displays the nice value (urgency) of the process as a decimal value. This field is currently not supported and will display a dash. [NI]

`pcpu` Displays a percentage value giving the ratio of processor time used to processor time available. This field is currently not supported and will display a dash. [%CPU]

`pgid` Displays the process group ID as a decimal value. [PGID]

`pid` Displays the process ID as a decimal value. Decimal *pid*s are reported with default actions. [XPID]

`ppid` Displays the parent process ID as a decimal value. [PPID]

pri Displays the process priority. This field is currently not supported and will display a dash. [PRI]

rgid Displays the real group ID of the process. [GID]

rgroup Displays the real group ID of the process, as a group name if possible and as a decimal group ID if not. [RGROUP]

ruid Displays the real user ID of the process. [UID]

ruser Displays the real user ID of the process, as a user name if possible and as a decimal user ID otherwise. [RUSER]

sid Displays the session ID of the process. [SID]

stime Displays the start time of the process. [STIME]

thdcnt Displays the total number of threads. [THCNT]

time Displays the amount of processor time that the process has used since it began running. **ps** displays this time in form similar to that used by etime. [TIME]

tty Displays the name of the controlling terminal (if any). [TT]

uid Displays the effective user ID of the process. [EUID]

user Displays the effective user ID of the process, as a user name if possible and as a decimal user ID otherwise. [USER]

vsz Displays the amount of (virtual) memory that the process is using, as a decimal number of kilobytes. [VSZ]

wchan Displays the channel upon which the process is waiting. This field is currently not supported and will display a dash. [WCHAN]

xasid Displays the address space id as a hexadecimal value (Note: a non-hex asid is not supported). [ASID]

xpgid Displays the process group ID as a hexadecimal value. [XPGID]

xpid Displays the process ID as a hexadecimal value. [XPID]

xppid Displays the parent process ID as a hexadecimal value. [XPPID]

Thread Only:

lpid Displays the latch pad waited for. [lpid]

lsyscall Displays the last five syscalls. This is a 20 character string consisting of five four character syscalls with no delimiting characters between them. From left-to-right the syscalls are ordered from most recent to oldest. In the following example of lsyscall output, 1WAT is the most recent syscall: 1WAT1SPM1SPM1SPM1TSP. [LASTSYSC]

semnum Displays the semaphore number of the semaphore the thread is in a wait state for. (Note: a semaphore number is only available when the thread is in a semaphore wait state (state field value equals d), otherwise, a dash will be displayed). [SNUM]

semval Displays the semaphore value of the semaphore the thread is in a wait state for. (Note: a semaphore value is only available when the thread is in a semaphore wait state (state field value equals D), otherwise, a dash will be displayed). [SVAL]

`sigmask` Displays the signal pending mask as a hexidecimal value. [SIGMASK]

`syscall` Displays the current syscall (for example, `1frk` for fork). [SYSC]

`tagdata` Displays the tag assigned to the thread using pthread_tag_np(). If a tag was not assigned, a dash will be displayed. [TAGDATA]

`wtime` Displays waiting time in the form *fffffldd"hh"mm:ssffl.xx"* where *dd* is the number of days, *hh* is the number of hours, *mm* is the number of minutes, and *ss* is the number of seconds. [WTIME]

`xtcbaddr` Displays the tcb address as a hexidecimal value. A non-hex tcb address is not supported. [TCBADDR]

`xstid` Displays the short thread id as a hexidecimal value. This is the low order word (the sequential value) of the thread id. A non-hex short thread ID is not supported. [STID]

`xtid` Displays thread id as a hexidecimal value. A non-hex thread ID is not supported. [TID]

Processes and Threads:

`flags` Displays the state field values using a hexadecimal representation. `flags` is the four byte value determined when a bit is set to one for each corresponding state that is active. Below is the state-to-state bit mapping for the currently defined state values:

```
 Byte    0        1        2        3
 Bits 11111111 11111111 11111111 11010000
      -------- -------- -------- --------
state ABCDEFG   JK  NO   RS UVWX YZ 1
```

For example, if a thread or process had a state field value of `1W`, then the following bits would be set:

```
 Byte    0        1        2        3
 Bits 00000000 00000000 00000010 00010000
      -------- -------- -------- --------
state
```

Which when represented as a hexadecimal value would be `210.[F]`

`state` Displays the process state. [STATE] Various values can be printed in this field:

1 A single task using assembler callable services.
A Message queue receive wait.
B Message queue send wait.
C Communication system kernel wait.
D Semaphore operation wait.
E Quiesce frozen.
F File system kernel wait.
G MVS Pause wait.
J Pthread created.
K Other kernel wait (for example, pause or sigsuspend).
N Medium weight thread.
O Asynchronous thread.
R Running (not kernel wait).
S Sleeping.
U Initial process thread.

V Thread is detached.
W Waiting for a child (wait or waitpid function is running).
X Creating a new process (fork function is running).
Y MVS wait.
Z Canceled and parent has not performed wait (*Z* for zombie).

THREAD THREAD [THREAD] is a synonym for specifying the following fields:

```
-m -o ruser=UID -o pid,ppid,xstid,state=STATE -o atime,syscall,args=CMD
```

The following is an example of how this output will appear:

```
    UID        PID       PPID      STID   STATE  TIME   SYSC   CMD
WELLIE8    67108867  15099496         -   1W     0:25   -      sh -L
     -           -         -  00000002   W      0:17   1WAT   -
WELLIE8  1073741830   67108867        -   1Y     0:00   -      ./ps -o THREAD
     -           -         -  00000000   Y      0:00   1GTH   -
```

Environment Variables

ps uses the following environment variable:

COLUMNS
Contains the maximum number of columns to display on one line.

Localization

ps uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_TIME**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion
1 Failure due to the inability to open the process table
2 Failure due to any of the following:
- Unknown command-line option
- Missing *format* string after **–o**
- Missing lists after other options
- Too many arguments on the command line

Portability

POSIX.2 User Portability Extension, X/Open Portability Guide.

The **–c**, **–d**, **–e**, **–f**, **–g**, **–j**, **–l**, **–m**, **–n**, **–s**, and **–u** options are extensions of the POSIX standard.

Related Information

jobs, **kill**

---

pwd — Return the working directory name

Format

pwd

Description

pwd displays the absolute pathname of the working directory to standard output.

If the current working directory is a symbolic link to another directory, the pathname displayed depends on the setting of the shell's logical flag. See **set** for more information.

Usage Note

pwd is a built-in shell command and is also a separate utility.

Localization

pwd uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion
1 Inability to determine the working directory

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Information

set, **sh**

---

r — Process a command history list

Format

r [*old=new*] [*specifier*]

Description

r is an alias for **fc –s**. Like **fc –s**, **r** reenters exactly one command without going through an editor. If a command specifier is given, **r** selects the command to reenter; otherwise, **r** uses the last command entered. To perform a simple substitution on the command before reentry, use a parameter of the form *old=new*. The string *new* replaces the first occurence of string *old*. **r** displays the (possibly modified) command before reentering it.

See "fc — Process a command history list" on page 250 for more information.

Related Information

fc, **history**, **sh**

---

read — Read a line from standard input

Format

read [**–prs**] [**–u**[*d*]] [*variable***?***prompt*] [*variable ...*]

Description

When you call **read** without options, it reads one line from the standard input, breaks the line into fields, and assigns the fields to each *variable* in order.

To determine where to break the line into fields, **read** uses the built-in variable **IFS** (which stands for *internal field separator*). Encountering any of the characters in **IFS** means the end of one field and the beginning of the next. The default value of **IFS** is blank, tab, and newline.

In general, a single **IFS** character marks the end of one field and the beginning of the next. For example, if **IFS** is colon (`:`), **read** considers the input `a::b` to have three fields: `a`, an empty field, and `b`. However, if **IFS** contains blanks, tabs or escaped newlines, **read** considers a sequence of multiple blanks, tabs, or escaped newlines to be a single field separator. For example, `"a  b"` has two fields, even though there are several blanks between the `a` and `b`.

The *n*th *variable* in the command line is assigned the *n*th field. If there are more input fields than there are variables, the last variable is assigned all the unassigned fields. If there are more variables than fields, the extra variables are assigned the null string (`""`).

The environment variable **REPLY** is assigned the input when no variables are given. The exit status of **read** is `0`, unless it encounters the end of the file.

Options

–p Receives input from a coprocess.

–r Treats input as raw data, ignoring escape conventions. For example, **read –r** does not interpret a final backslash (\) as a line continuation character, but as part of the input.

–s Adds input to the command history file as well as to the variables specified with *variable*.

–u[*d***]**
Reads input from the single-digit file descriptor *d,* rather than from the standard input. The default file descriptor is `0`.

When the first variable parameter has the form:

*variable***?***prompt*

it defines a prompt for input. If the shell is interactive, **read** sends the prompt to the file descriptor `d` if it is open for write and is a terminal device. The default file descriptor for the prompt is `2`.

Examples

```
IFS=':'
while read name junk junk1 junk2 junk3
do
  echo $name
done </samples/comics.lst
```

provides a list of comic names from the sample **comics.lst** file.

Environment Variables

read uses the following environment variables:

IFS Contains a string of characters to be used as internal field separators.

PS2 Contains the prompt string that an interactive shell uses when it reads a line ending with a backslash and you did not specify the **–r** option, or if a here-document is not terminated after you enter a newline.

REPLY
Contains the input (including separators) if you did not specify any variables. The ability of omitting the variable from the command and using the environment variable **REPLY** is an extension.

Localization

read uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Usage Note

read is a built-in shell command.

Exit Values

`0` Successful completion
`1` Failure due to any of the following:
- End-of-file on input
- Incorrect *variable*
- Incorrect descriptor specified after **–u**
- Missing coprocess

`2` Incorrect command-line argument

Messages

Possible error messages include:

Cannot read on file descriptor ...
You tried to read a file descriptor that was not opened for reading.

Portability

POSIX.2, X/Open Portability Guide.

The **–p**, **–s**, and **–u** options are extensions of the POSIX :epsc. standard.

Related Information

continue, **fc**, **print**, **sh**

---

readonly — Mark a variable as read-only

Format

readonly [**–p**] [*name*[**=***value*] ...]

Description

readonly prevents subsequent changes in the value of any of the *name* arguments. Parameters of the form:

name=value

assign *value* to *name* as well as marking *name* read-only. If **readonly** is called without arguments, it lists, with appropriate quoting, the names you have set as read-only in the following format:

Variable="*value*"

Options

–p Displays *export name=value* pairs that, when read by a shell, ensures the read-only status and values of variables. The shell formats the output so it is suitable for reentry to the shell as commands that achieve the same attribute-setting results.

Because it is not possible to change a read-only variable, you cannot source the output unless you go to a new shell.

Usage Note

readonly is a special built-in shell command.

Localization

readonly uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion
1 An attempt to give read-only status to a variable that is already read-only
2 Failure due to incorrect command-line argument

Portability

POSIX.2, X/Open Portability Guide.

The behavior given for calling **readonly** with no arguments is an extension of the POSIX standard.

Related Information

alias, **sh**, **typeset**

---

renice — Change priorities of a running process

Format

renice [**–n** *increment*] [**–gl–p –u**] *ID ...*
renice *priority* [**–p**] *pid ...* [**–g** *pgrp ...*] [**–p** *pid ...*] [**–u** *user ...*]
renice *priority* **–g** *pgrp ...* [**–g** *pgrp ...*] [**–p** *pid ...*] [**–u** *user ...*]
renice *priority* **–u** *user ...* [**–g** *pgrp ...*] [**–p** *pid ...*] [**–u** *user ...*]

Description

renice changes the priority of one or more running processes. Normal users can change only the priority of processes that have the same real or effective user ID as the real or effective user ID of the process that calls **renice**. Privileged users can set the priority of any process.

You can specify the new *priority* as a decimal integer, with higher values indicating more urgent priority. The range of priorities is site-specific, and you may require appropriate privileges for some priority values.

When you change the priority of a process group, the priority of all processes in that group are changed.

If the string **--** appears in the arguments, **renice** does *not* interpret it as the end of command-line arguments. This is an exception to the usual POSIX syntax rules.

Options

–g Treats all following *ID*s (or just *pgrp*s in the obsolescent versions) as process group IDs.

–n *increment*
Adjusts the system scheduling priority of the specified processes by *increment*. Positive *increment*s lower the priority while negative *increment*s result in a higher priority.

Note: Negative *increment*s may require appropriate privileges.

–p Treats all following *ID*s (or just *pid*s in the obsolescent versions) as process IDs.

–u Treats all following *ID*s (or just *user*s in the obsolescent versions) as either user names or numeric user IDs.

priority
A number that indicates an absolute priority value (higher numbers reflect higher priorities).

If no **–p**, **–g**, or **–u** option appears on the command line, **renice** assumes **–p**.

Localization

renice uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion

1 Failure due to one of the following:

- Incorrect command-line argument
- The wrong number of command-line arguments
- A *priority* that is outside the range
- An incorrect *priority* argument
- An incorrect *ID* argument
- Missing arguments following one of the options

2 Failure because the system does not recognize the *ID* in a **–u** option

Portability

POSIX.2 User Portability Extension, UNIX systems.

POSIX considers all but the first form of the **renice** command to be obsolescent.

Related Information

nice

---

return — Return from a shell function or . (dot) script

Format

return [*expression*]

Description

return returns from a shell function or **.** (dot) script. The exit status is the value of *expression*. The default value of *expression* is the exit status of the last command run.

Usage Note

return is a special built-in shell command.

Localization

renice uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

The current function or script returns the value of *expression*. If no *expression* is given, the exit status is the exit status of the last command run.

Portability

POSIX.2, X/Open Portability Guide.

Related Information

exit, **sh**

---

rlogind — Validate rlogin requests

Format

rlogind [**–a**] [**–d**] [**–l**] [**–m**] [**–n**]

Description

The **rlogind** program is the server for the remote login command **rlogin** commonly found on UNIX systems. It validates the remote login request and verifies the password of the target user. It starts an OS/390 shell for the user and handles translation between ASCII and EBCDIC code pages as data flows between the workstation and the shell.

The **rlogind** program is given control via an **execl()** issued by the **inetd** daemon.

Note

rlogind should always be invoked from **inetd** through the **/etc/inetd.conf** file. It should not be invoked from the shell. **inetd** sets up certain files and sockets needed by **rlogind**. Invoking **rlogind** directly gives unpredictable results.

Options

–a Specifies that the requester's Internet address be checked against the local **gethostbyname()** file. This option has no effect because the rlogin program never uses the **.rhosts** file for authentication.

–d Specifies that the debugging option be enabled. Informational messages on the rlogin process is written to the system log.

–l Specifies that the **.rhosts** file for authentication not be used. This option has no effect because the rlogin program never uses the **.rhosts** file for authentication.

–m Specifies that multiprocessing support in the user's address space be enabled. Using the **–m** option uses fewer system resources and provides faster performance for the end user.

If you do not specify **–m**, each rlogin request causes two MVS address spaces to be consumed. The first address space is the rlogind code, which provides the user connection to the socket, and the second is the user's shell. In this mode, all shell functions behave in a manner conformant to the standards.

If you specify **–m**, the rlogin process and the shell process share the same address space using OS/390 Unix System Services support for multiple processes in an address space. Using **–m** has the potential of doubling the number of users supported via rlogin.

Note: If you issued **rlogind** with the **–m** option, the shell process cannot execute a setuid program that replaces the shell. This causes functions like **newgrp** to fail. In this situation, you may want to create a secondary shell that runs in its own address space.

–n Specifies that the transport-level keep-alive messages be disabled. The messages are enabled by default.

Usage Notes

1. The **rlogind** program normally translates all error and warning messages to ASCII and then sends them to the originating terminal.

   However, when the C runtime library writes error messages, the **rlogind** program cannot intercept them to translate the messages to ASCII. Therefore, these messages are written to the file **/tmp/rlogind.stderr** or **/tmp/rlogind2.stderr**.

   These two files must be predefined in **/tmp**, and owned by the superuser (UID 0). The files should have permissions of rw–rw–rw or rw—w—w–. In addition, the sticky bit must be set for the **/tmp** directory so that these files (and other files in **/tmp**) cannot be removed except by the files' owners or the superuser.

2. **rlogind** is not affected by the locale information specified in locale-related environment variables.

Related Information

inetd

---

rm — Remove a directory entry

Format

rm [**–fiRrv**] *file ...*

Description

rm removes files (provided that it is a valid pathname). If you specify either **.** or **..** as the final component of the pathname for a *file*, **rm** displays an error message and goes to the next file. If a file does not have write permission set, **rm** asks you if you are sure you want to delete the file; type the yes expression defined in **LC_MESSAGES** (the English expression is typically `y` or `yes`) if you really want it deleted.

Note: A file can be removed by any user who has write permission to the directory containing the file, unless that directory has its sticky bit turned on. If the file is in a directory whose sticky bit is turned on, only the file owner, the owner of the directory, or a superuser can remove the file.

Options

–f Deletes read-only files immediately without asking for confirmation. When you specify this option and a file does not exist, **rm** does not display an error message and does not modify the exit status. If you specify both **–f** and **–i**, **rm** uses the option that appears last on the command line. If no files are specified, **rm –f** will not issue an error.

–i Prompts you for confirmation before deleting each file. If you specify both **–f** and **–i**, **rm** uses the option that appears last on the command line.

–R Recursively removes the entire directory structure if *file* is a directory.

–r Is equivalent to **–R**.

–v Displays a list of files that were removed.

Localization

rm uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion

1 Failure due to any of the following:

- Inability to remove a file
- Attempt to remove directory without specifying **–r** or **–R**
- Inability to find file information when using **–r** or **–R**
- Inability to read directory when using **–r** or **–R**

2 Failure due to any of the following:

- Incorrect command-line option
- No file was specified

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Information

cp, **mv**, **rmdir**

---

rmdir — Remove a directory

Format

rmdir [**–p**] *directory ...*

Description

rmdir removes each requested *directory*. Each directory must be empty for **rmdir** to be successful.

Options

–p Removes all intermediate components. For example:

```
rmdir -p abc/def/ghi
```

is equivalent to:

```
rmdir abc/def/ghi
rmdir abc/def
rmdir abc
```

Localization

rmdir uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion
1 Failure because *directory* is not a directory, or because it still contains files or subdirectories
2 Failure because of an incorrect command-line option, or no *directory* names specified

Messages

Possible error messages include:

Nonempty directory
Files or other directories are found under the directory to be removed. Use **rm –r** to remove the directory.

No such directory
The requested directory does not exist or is otherwise inaccessible.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Information

mkdir, **rm**

---

runcat — Pipe output from mkcatdefs to gencat

Format

runcat *CatalogName SourceFile* [*CatalogFile*]

Description

runcat invokes the **mkcatdefs** command and pipes the message catalog source data (the output from **mkcatdefs**) to the **gencat** utility.

The file specified by the *SourceFile* parameter contains the message text with your symbolic identifiers. The **Created by ActiveSystems 10/25/96 Entity not defined.** program uses the *CatalogName* parameter to generate the name of the symbolic definition file by adding `.h` to the end of the *CatalogName* value, and to generate the symbolic name for the catalog file by adding **MF_** to the beginning of the *CatalogName* value. The definition file must be included in your application program. The symbolic name for the catalog file can be used in the library functions (such as the **catopen** subroutine). *SourceFile* cannot be **stdin**.

The *CatalogFile* parameter is the name of the catalog file created by the **gencat** command. If you do not specify this parameter, the **gencat** command names the catalog file by adding `.cat` to the end of the *CatalogName* value. This filename can also be used in the **catopen** subroutine.

Examples

To generate a catalog named **test.cat** from the message source file **test.msg**, enter:

```
runcat test test.msg
```

Related Information

dspcat, **dspmsg**, **gencat**, **mkcatdefs**

---

sed — Start the sed noninteractive stream editor

Format

sed [**–En**] [*script*] [*file ...*]
sed [**–En**] [**–e** *script*] .... [**–f** *scriptfile*] .... [*file ...*]

Description

The **sed** command applies a set of editing subcommands contained in *script* to each argument input *file*.

If more than one *file* is specified, they are concatenated and treated as a single large file. **script** is the arguments of all **–e** and **–f** options and the contents of all *scriptfile*s. You can specify multiple **–e** and **–f** options; commands are added to *script* in the order specified.

If you did not specify *file*, **sed** reads the standard input.

sed reads each input line into a special area known as the *pattern buffer*. Certain subcommands [**gGhHx**] use a second area called the *hold buffer*. By default, after each pass through the script, **sed** writes the final contents of the *pattern buffer* to the standard output.

Options

–E Uses extended regular expressions. Normally, **sed** uses basic regular expressions. For more information, see Appendix C.

–e *script*
Adds the editing subcommands *script* to the end of the script.

–f *scriptfile*
Adds the subcommands in the file *scriptfile* (one command per line) to the end of the script.

–n Suppresses all output except that generated by explicit subcommands in the **sed** script [**acilnpPr**]

If you need only one *script* argument, you can omit the **–e** and use the first form of the command.

sed subcommands are similar to those of the interactive text editor **ed**, except that **sed** subcommands necessarily view the input text as a stream rather than as a directly addressable file.

Each line of a **sed** script contains up to two addresses, a single letter command, possible command modifiers, and a terminating newline. The newline is optional in script strings typed on the command line.

[*addr*[**,***addr*]]**.** *command* [*modifiers*]

Subcommands

sed subcommands necessarily view the input text as a stream rather than as a directly addressable file. Script subcommands can begin with zero, one, or two addresses, as in **ed**.

- Zero-address subcommands refer to every input line.
- One-address subcommands select only those lines matching that address.
- Two-address subcommands select those input line ranges starting with a match on the first address up to an input line matching the second address, inclusive.

Permissible addressing constructions are:

n The number *n* matches only the *n*th input line.

$ This address matches the last input line.

/*regexp*/
This address selects an input line matching the specified regular expression *regexp*. If you do not want to use slash (/) characters around the regular expression, use a different character but put a backslash (\) before the first one. For example, if you want to use % to enclose the regular expression, write \%*regexp*%.

Each line of a script contains up to two addresses, a single-letter subcommand, possible subcommand modifiers, and an ending newline. The newline is optional in script strings entered on the command line.

The following **sed** subcommand summary shows the subcommands with the maximum number of legitimate addresses. A subcommand can be given fewer than the number of addresses specified, but not more.

aa\
*A*ppends subsequent text lines from the script to the standard output. **sed** writes the text after completing all other script operations for that line and before reading the next record. Text lines are ended by the first line that does not end with a backslash (\). **sed** does not treat the \ characters on the end of lines as part of the text.

a,bb **[***label***]**
*B*ranches to **:***label*. If you omit *label*, **sed** branches to the end of the script.

a,bc\
*C*hanges the addressed lines by deleting the contents of the pattern buffer (input line) and sending subsequent text (similar to the *a* command) to the standard output. When you specify two addresses, **sed** delays text output until the final line in the range of addresses; otherwise, the behavior would surprise many users. The rest of the script is skipped for each addressed line except the last.

a,bd
*D*eletes the contents of the pattern buffer (input line) and restarts the script with the next input line.

a,bD
*D*eletes the pattern buffer only up to and including the first newline. Then it restarts the script from the beginning and applies it to the text left in the pattern buffer.

a,bg
*G*rabs a copy of the text in the hold buffer and places it in the pattern buffer, overwriting the original contents.

a,bG
*G*rabs a copy of the text in the hold buffer and appends it to the end of the pattern buffer after appending a newline.

a,bh
*H*olds a copy of the text in the pattern buffer by placing it in the hold buffer, overwriting its original contents.

a,bH
*H*olds a copy of the text in the pattern buffer by appending it to the end of the hold buffer after appending a newline.

ai\
*I*nserts text. This subcommand is similar to the **a** subcommand, except that its text is output immediately.

a,bl
*L*ists the pattern buffer (input line) to the standard output so that nonprintable characters are visible. The end-of-line is represented by $, and the characters \\, \a, \b, \f, \r, \t, and \v are printed as escape sequences. Each byte of a nonprintable doublebyte character appears as an escape sequence or as a 3-digit octal number. This subcommand is analogous to the **l** subcommand in **ed**.

sed folds long lines to suit the output device, indicating the point of folding with a backslash (\).

a,bn
Prints the pattern space on standard output if the default printing of the pattern space is not suppressed (because of the **-n** option). The *n*ext line of input is then read, and the processing of the line continues from the location of the **n** command in the script.

a,bN
Appends the *n*ext line of input to the end of the pattern buffer, using a new line to separate the appended material from the original. The current line number changes.

a,bp
*P*rints the text in the pattern buffer to the standard output. The **–n** option does not disable this form of output. If you do not use **–n**, the pattern buffer is printed twice.

a,bP
Operates like the **p** subcommand, except that it prints the text in the pattern buffer only up to and including the first newline character.

aq
*Q*uits **sed**, skipping the rest of the script and reading no more input lines.

ar *file*
*R*eads text from *file* and writes it to the standard output before reading the next input line. The timing of this operation is the same as for the **a** subcommand. If *file* does not exist or cannot be read, **sed** treats it as an empty file.

a,bs/*reg*/ *sub*/**[gp***n***] [w***file***]**
*S*ubstitutes the new text string *sub* for text matching the regular expression, *reg*. Normally, the **s** subcommand replaces only the first such matching string in each input line. You can use any single printable character other than space or newline instead of the slash (/) to delimit *reg* and *sub*. The delimiter itself may appear as a literal character in *reg* or *sub* if you precede it with a backslash (\). You can omit the trailing delimiter.

If an ampersand (&) appears in *sub*, **sed** replaces it with the string matching *reg*. For more information, see Appendix C. A \n in *reg* matches an embedded newline in the pattern buffer (resulting,for example, from an **N** subcommand). The subcommand can be followed by a combination of the following:

n Substitutes only the *n*th occurrence of *regexp.*

g Replaces all nonoverlapping occurrences of *regexp* rather than the default first occurrence.

p Executes the print (**p**) subcommand only if a successful substitution occurs.

w *file*
*W*rites the contents of the pattern buffer to the end of *file*, if a substitution occurs.

a,bt **[***label***]**
Branches to the indicated *label* if a successful substitution has occurred since either reading the last input line or running the last **t** subcommand. If you do not specify *label*, **sed** branches to the end of the script.

a,bw *file*
*W*rites the text in the pattern buffer to the end of *file*.

a,bx
E*x*changes the text in the hold buffer with that in the pattern buffer.

a,by/*set1*/*set2*/
Transliterates any input character occurring in *set1* to the corresponding element of *set2*. The sets must be the same length. You can use any character other than backslash or newline instead of the slash to delimit the strings.

a,b{
Groups all commands until the next matching } subcommand, so that **sed** runs the entire group only if the { subcommand is selected by its address(es).

:*label*
Designates a *label*, which can be the destination of a **b**or **t** subcommand.

a,b!*cmd*
Runs the specified *cmd* only if the addresses do *not* select the **!** subcommand.

Treats the script line as a comment unless it is the first line in the script. Including the first line in a script as #n is equivalent to specifying **–n** on the command line. An empty script line is also treated as a comment.

a=
Writes the decimal value of the current line number to the standard output.

Example

This filter switches desserts in a menu:

```
sed 's/cake\(ic\)*/cookies/g'
```

Environment Variable

sed uses the following environment variable:

COLUMNS Contains the width of the screen in columns. If set, **sed** uses this value to fold long lines on output. Otherwise, **sed** uses a default screen width of 80.

Localization

sed uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion

1 Failure due to any of the following:

- Missing script
- Too many script arguments
- Too few arguments
- Unknown option
- Inability to open script file
- No noncomment subcommand
- Label not found in script
- Unknown subcommand
- Nesting **!** subcommand not permitted
- No \ at end of subcommand
- End-of-file in subcommand
- No label in subcommand
- Badly formed filename
- Inability to open file
- Insufficient memory to compile subcommand
- Bad regular expression delimiter
- No remembered regular expression
- Regular expression error
- Insufficient memory for buffers
- **y** subcommand not followed by a printable character as separator
- The strings are not the same length
- Nonmatching { and } subcommands
- Garbage after command
- Too many addresses for command
- Newline or end-of-file found in pattern

- Input line too long
- Pattern space overflow during **G** subcommand
- Hold space overflow during **H** subcommand
- Inability to chain subcommand

Messages

The error messages are output only if **h** or **H** subcommands are used after **sed** outputs `?`. Possible error messages include:

badly formed filename for *command* **command**
The given subcommand required a filename, but its operand did not have the syntax of a filename.

Cannot nest ! command
A **!** subcommand cannot contain a **!** subcommand of its own.

subcommand **command needs a label**
The specified subcommand required a label, but you did not supply one.

must have at least one (noncomment) command
The input to **sed** must contain at least one active subcommand (that is, a subcommand that is not a comment).

No remembered regular expression
You issued a subcommand that tried to use a remembered regular expression—for example, `s//abc`. However, there is no remembered regular expression yet. To correct this, change the subcommand to use an explicit regular expression.

Limits

sed allows a limit of 1024 bytes per line and 28 000 lines per file. It does not allow the NUL character. The maximum length of a global command is 256 characters, including newlines.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The **–E** option is an extension of the POSIX standard and is unique to this version of **sed**.

Related Information

awk, **diff**, **ed**, **grep**, **vi**

See Appendix C for more information about **regexp**.

---

set — Set or unset command options and positional parameters

Format

set [±**abCefhiKkLmnpstuvx–**] [±**o**[*flag*]] [±**A***name*][*parameter ...*]

tcsh shell:

1. **set** [**-r**]
2. **set** [**-r**] *name ...*
3. **set** [**-r**] *name=word ...*
4. **set** [**-r**] [**-fl-l**] *name=(wordlist*) *...*
5. **set** *name*[*index*]=*word ...*

Description

Calling **set** without arguments displays the names and values of all shell variables, sorted by name, in the following format:

```
Variable="value"
```

The quoting allows the output to be reinput to the shell using the built-in command **eval**. Arguments of the form **–***option* set each shell flag specified as an option. Similarly, arguments of the form **+***option* turn off each of the shell flags specified as an option. (Contrary to what you might expect, **–** means *on,* and + means *off.*)

Note: All of the **set** options except ±**A**, **–s**, **–**, and **—** are shell flags. Shell flags can also be set on the **sh** command line at invocation.

set in the tcsh shell

tcsh shell: See format section above to view the forms described below.

1. The first form of the command prints the value of all shell variables. Variables which contain more than a single word print as a parenthesized word list.

   Variables which are read-only will only be displayed by using the **-r** option. For forms 2, 3 and 4, if **-r** is specified, the value is set to read-only.
2. The second form sets *name* to the null string.
3. The third form sets *name* to the single *word*.
4. The fourth form sets *name* to the list of words in *wordlist*. In all cases the value is command and filename expanded. If **-f** or **-l** is specified, **set** only unique words keeping their order. **-f** prefers the first occurrence of a word, and **-l** the last.
5. The fifth form sets the *index*'th component of *name* to *word*; this component must already exist.

These arguments can be repeated to set and/or make read-only multiple variables in a single set command. However, variable expansion happens for all arguments before any setting occurs. Also, '=' can be adjacent to both name and word or separated from both by whitespace, but cannot be adjacent to only one or the other. For example:

```
set -r name=word and set -r name = word
```

are allowed, but

```
set -r name= word and set -r name =word
```

are not allowed.

See “tcsh — Invoke a C shell” on page 556.

Options

–a Sets all subsequently defined variables for export.

–b Notifies you when background jobs finish running.

–C Prevents the output redirection operator > from overwriting an existing file. Use the alternate operator >| to force an overwrite.

–e Tells a noninteractive shell to execute the ERR trap and then exit. This flag is disabled when reading profiles.

–f Disables pathname generation.

–h Makes all commands use tracked aliases.

See page 494 for an explanation of tracked aliases.)

–i Makes the shell interactive.

–K Tells the shell to use KornShell-compatible behavior in any case where the POSIX.2 behavior is different from the behavior specified by the KornShell. For more details, see the **let** and **trap** command descriptions.

–k Allows assignment parameters anywhere on the command line and still includes them in the environment of the command.

–L Makes the shell a login shell. Setting this flag is effective only at shell invocation.

–m Runs each background job in a separate process group and reports on each as they complete.

–n Tells a noninteractive shell to read commands but not run them.

–o *flag*

Sets a shell *flag*. If you do not specify *flag*, this option lists all shell flags that are currently set. *flag* can be one of the following:

allexport Is the same as the **–a** option.

errexit Is the same as the **–e** option.

bgnice Runs background jobs at a lower priority.

emacs Specifies **emacs**- style inline editor for command entry. See **shedit** for information about the **emacs**editing mode.

gmacs Specifies **gmacs**- style inline editor for command entry. See **shedit** for information about the **gmacs** editing mode.

ignoreeof Tells the shell not to exit when an end-of-file character is entered.

interactive Is the same as the **–i** option.

keyword Is the same as the **–k** option.

korn Is the same as the **–K** option.

logical Specifies that **cd, pwd** and the **PWD** variable use logical pathnames in directories with symbolic links. If this flag is not set, these built-ins and **PWD** use physical directory pathnames. For example, assume **/usr/spool** is a symbolic link to **/var/spool**, and that it is your current directory. If **logical** is not

set, **PWD** has the value **/var/spool**, and **cd** changes the current directory to **/var**. If **logical** is set, **PWD** has the value **/usr/spool** and **cd** changes the current directory to **/usr**.

login Is the same as the **–L** option of **sh**.

markdirs Adds a trailing slash (/) to filename-generated directories.

monitor Is the same as the **–m** option.

noclobber Is the same as the **–C** option.

noexec Is the same as the **–n** option.

noglob Is the same as the **–f** option.

nolog Does not record function definitions in the history file.

notify Is the same as the **–b** option.

nounset Is the same as the **–u** option.

privileged Is the same as the **–p** option.

trackall Is the same as the **–h** option.

verbose Is the same as the **–v** option.

xtrace Is the same as the **–x** option.

vi Specifies **vi**- style inline editor. See **shedit** for information about the **vi**editing mode.

warnstopped
Tells the shell to issue a warning, but not to exit, when there are stopped jobs.

–p Resets the **PATH** variable to the default value, disables processing of **$HOME/.profile**, and ignores the value of the **ENV** variable.

–s Sorts the positional parameters.

–t Exits after reading and running one command.

–u Tells the shell to issue an error message if an unset parameter is used in a substitution.

–v Prints shell input lines as they are read.

–x Prints commands and their arguments as they run.

Other options:

– Turns off the **–v** and **–x** options. Also, parameters that follow this option do not set shell flags, but are assigned to positional parameters (see **sh**).

— Specifies that parameters following this option do not set shell flags, but are assigned to positional parameters.

+A *name*
Assigns the parameter list to the elements of *name*, starting at *name*[0].

–A *name*
Unsets *name* and then assigns the parameter list to the elements of *name* starting at *name*[0].

Usage Notes

set is a special built-in shell command.

Localization

set uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion
1 Failure due to an incorrect command-line argument
2 Failure resulting in a usage message, usually due to a missing argument

Portability

POSIX.2, X/Open Portability Guide.

Several shell flags are extensions of the POSIX standard: **bgnice**, **ignoreeof**, **keyword**, **markdirs**, **monitor**, **noglob**, **nolog**, **privileged**, and **trackall** are extensions of the POSIX standard, along with the shell flags ±**A**, ±**h**, ±**k**, ±**p**, ±**s**, and ±**t**.

Related Information

alias, **eval**, **export**, **sh**, **shedit**, **tcsh**, **trap**, **typeset**

---

sh — Invoke a shell

Format

[**r**]**sh** [±**abCefhiKkLmnprtuvx**] [±**o** *option*] [*cmd_file* [*argument ...*]]
[**r**]**sh –S** [±**abCefhiKkLmnprtuvx**] [±**o** *option*] [*cmd_file* [*argument ...*]]
[**r**]**sh –c** *cmdstring* [±**abCefhiKkLmnprtuvx**] [±**o** *option*] [*cmd_name* [*argument ...*]]
[**r**]**sh –s** [±**abCefhikLmnprtuvx**] [±**o** *option*] [*argument ...*]

Description

sh contains the following sections and subsections:

- Options and invocation
- Options
- Command syntax
- Reserved-Word Commands
- Command execution
- Quoting
- Directory substitution
- Parameter substitution
- Arithmetic substitution
- Command substitution

- File descriptors and redirection
- Filename generation
- Variables
- Shell execution environments
- Built-in commands
- Examples
- Shell Variables
- Files
- Localization
- Exit Values
- Messages

Subsections dealing with substitution and interpretation of input appear in the order in which the shell performs those substitutions and interpretations.

Much of what the shell can do is provided through such built-in commands as **cd** and **alias**.

Options and Invocation

The OS/390 UNIX System Services shell, based on the KornShell, is upward-compatible with the Bourne shell.

Normally you invoke the shell by logging in. You can also invoke the shell by typing an explicit **sh** command. Some people find it useful to copy the **sh** file into a file named **rsh**. If you invoke the shell under the name **rsh**, the shell operates in *restricted* mode. This mode is described in connection with **–r**.

If you invoke the shell with a name that begins with the – character, it is a *login shell*. (You can also get a login shell if you invoke the shell with the **–L** option.) A login shell begins by running the file **/etc/profile**. It then runs **$HOME/.profile** using the **.** command (see **dot**). If **HOME** is not set, the shell searches the working directory for:

```
.profile
```

and runs this file with the **.** command if it exists. You do not get an error message if any of these files cannot be found.

You can use these profile files to customize your session with **sh**. For example, your profile files can set options, create aliases, or define functions and variables.

If there is at least one argument on the **sh** command line, **sh** takes the first argument as the name of a shell script to run. (The exception to this is when **–s** is used.) Any additional arguments are assigned to the positional parameters; usually, these serve as arguments to the shell script. See “Parameter Substitution” on page 496 for information about positional parameters. Also see **set** for information about changing these parameters.

If **sh** finds the **ENV** environment variable set when it begins running (after profile processing), **sh** runs the file named by the expansion of the value of this variable.

Options

The shell accepts the following options on the command line:

–c *cmdstring*
: Runs *cmdstring* as if it were an input line to the shell and then exits. This is used by programs (for example, editors) that call the shell for a single command. **sh** assigns arguments after *cmdstring* to the positional parameters. If you specify *cmd_name,* special parameter 0 is set to this string for use when running the commands in *cmdstring*.

–i Invokes an interactive shell, as opposed to running a script. With **–i**, the shell catches and ignores keyboard interrupts. Without **–i**, an interrupt ends the shell. For shells that read from the terminal, **–i** is the default.

–K Specifies KornShell-compatible behavior where the POSIX.2 behavior is different from the behavior specified by the KornShell. Without -K, the shell defaults to POSIX.2 behavior.

–L Makes the shell a *login shell,* as described earlier. (A login shell is an interactive shell.)

–r Invokes a restricted shell. (As noted earlier, you can also invoke a restricted shell by using the name **rsh**). In a restricted shell, you cannot do the following:

- Use the **cd** command
- Change the values of the variables **ENV**, **PATH**, or **SHELL**
- Use > or >> to redirect output; or specify command names containing /

These restrictions do not apply during execution of your *profile* files.

–s Reads commands from standard input (**stdin**) and assigns all arguments to the positional parameters. Normally, if there is at least one argument to the shell, the first such argument is the name of a file to be run.

–S Searches the directories in the environment variable **PATH** for a file *cmd_file* that contains shell commands. The current working directory is not searched before **PATH**.

If you do not give either the **–c** or **–s** option, but you do specify *cmd_file*, the shell takes it as the name of a file that contains commands to be run. Special parameter 0 is set to this name.

If *cmd_file* contains a slash, the shell attempts to read that filename. If *cmd_file* does not contain a slash, the following can occur:

- If **–S** is specified, the shell searches for the file in **PATH**.
- If **–S** is not specified, the shell searches for the file in the current working directory, then in **PATH**.

In addition to these options, you can use any valid option to the **set** command (including **–o** *option*) as a command-line option to **sh**. See **set** for details.

Command Syntax

The shell implements a sophisticated programming language that gives you complete control over the execution and combination of individual commands. When the shell scans its input, it always treats the following characters specially:

```
;   &   (   )   <   >   |   '   \   "
space   tab   newline
```

If you want to use any of these characters inside an actual argument, you must quote the argument (so that the shell does not use the special meanings of the characters). See “Quoting” on page 495 for more information.

A *simple command* is a list of *argument*s separated by characters in the **IFS** environment variable (the default value of **IFS** has blank, tabs, and newlines).

When a word is preceded by an unescaped pound sign (#), the remainder of the line is treated as a *comment,* and the shell discards input up to but not including the next newline. When a command starts with a defined alias, **sh** replaces the alias with its definition (see **alias**).

A *reserved-word command* starts with a *reserved word* (for example, **if**, **while**, or **for**). Reserved-word commands provide flow of control operations for the shell. These are described in “Reserved-Word Commands” on page 489.

A *command* can be any of the following:

command:

simple command
reserved-word command
(*command*)
command |*command*
command &&*command*
command ||*command*
command &*command*
command &
command |&
command ;*command*
command ;
command<newline>

The following is the order of precedence of the preceding operators. The highest priority operators are listed first, and operators on the same line have equal priority.

```
()
|
&&     ||
&      |&       ;      <newline>
```

The meaning of these operations is as follows:

(*command*)
: Runs *command* in a child shell. The current shell invokes a second shell, and this second shell actually runs *command*. In this way, *command* runs in a completely separate execution environment; it can change working directories, change variables, open files, and so on without affecting the first shell. The child shell's environment begins as a copy of the current environment,

so the value of the **ENV** environment variable is not run when a child shell starts.

| Creates a pipe between the two *command*s that the | operator connects. The standard output is the first *command* becomes the standard input of the second *command*. A series of commands connected by pipes is called a *pipeline*. The exit status is that of the last command in the pipeline.

&& Is the logical AND operator. The shell runs the second *command* if and only if the first *command* returns a true (zero) exit status.

|| This is the logical OR operator. The shell runs the second *command* if and only if the first *command* returns a false (nonzero) exit status.

& Runs the *command* that precedes it asynchronously. The shell just starts the *command* running and then immediately goes on take new input, before the *command* finishes execution. On systems where asynchronous execution is not possible, this operation is effectively equivalent to **;**.

|& Runs the *command* that precedes it as a co-process. The *command* runs asynchronously, as with the & operator, but the command's standard input and standard output are connected to the shell by pipes. The shell sends input to *command*'s standard input with the **print –p** command, and reads from *command*'s standard output with the **read –p** command. The *command* should not buffer its output. Because of this and other limitations, co-processes should be designed to be used as co-processes. On systems where asynchronous execution is not possible, co-processes are not supported.

; Is the sequential execution operator. The second *command* is run only after the first *command* has completed.

newline
The unescaped newline is equivalent to the **;** operator.

Reserved-Word Commands

The shell contains a rich set of *reserved-word commands,* which provide flow of control and let you create compound commands. In the following list, a *command* can also be a sequence of *command*s separated by newlines. Square brackets (**[ ]**) indicate optional portions of commands, and are included as part of the command syntax except in the case of **[[** *test_expr* **]]**, where square brackets are part of the command.

! The exclamation point is the logical NOT command. When its operand is false (nonzero), this command returns true (zero). When its operand is true (zero), this command returns false (nonzero).

{*command***;}**
Enclosing a command in braces is similar to the (*command*) construct, except that the shell runs the *command* in the same environment rather than under a child shell. { and } are reserved words to the shell. To make it possible for the shell to recognize these symbols, you must put a blank or newline after the {, and a semicolon or newline before the }.

[[*test_expr* **]]**
The double-square-bracket command (**[[** *test_expr* **]]**) is a command that returns an exit status indicating whether the *test_expr* (test expression) is true or false.

Word-splitting and wildcard expansion (filename expansion or globbing) are NOT done within **[[ ]]**. This makes quoting less necessary than when you use the **test** (or **[ ]**) command. Alias expansion is also not done within **[[ ]]**.

The following primitives are used in (**[[** *test_expr* **]]**). Spaces or tabs are required to separate operators from operands.

–a file True if file exists (**–e** is recommended to avoid confusion with the **test** command syntax)

–b file True if file is a block special file (block special files are not supported in OS/390)

–c file True if file is a character special file

–d file True if file is a directory

–e file True if file exists

–f file True if file is an ordinary file

–g file True if the set-group-ID attribute of file is on

–G file True if file group owner is the effective group id

–h file True if file is a symbolic link

–k file True if file has the "sticky" bit on

–L file True if file is a symbolic link

–n string
True if the length of the string is greater than zero

–o option
True if shell option is on

–O file True if file owner is the effective user id

–p file True if file is a FIFO (named pipe)

–r file True if file is readable (checks permission bits and access control)

–s file True if size of the file is nonzero

–S file True if file is a socket

–t fd True if the numeric file descriptor fd is open and associated with a terminal

–u file True if the set-user-ID attribute of file is on

–w file True if file is writable (checks permission bits and access control)

–x file True if file is executable (checks permission bits and access control)

–z string
True if length of the string is zero

string True if string is not a null string

string = pattern
True if string matches pattern (== is recommended to avoid confusion with the **test** command syntax)

string == **pattern**
True if string matches pattern. Quote pattern to treat it as a string. See **Patterns** below.

string1 != pattern
True if string does not match patterns. See **Patterns** below.

string1 < **string2**
True if string1 comes before string2 in the collation order defined in the current locale

string1 > **string2**
True if string1 comes after string2 in the collation order defined in the current locale

exp1 –eq exp2
True if arithmetic expression exp1 and exp2 are equal

exp1 –ge exp2
True if arithmetic expression exp1 is greater than or equal to exp2

exp1 –gt exp2
True if arithmetic expression exp1 is greater than exp2

exp1 –le exp2
True if arithmetic expression exp1 is less than or equal to exp2

exp1 –lt exp2
True if arithmetic expression exp1 is less than exp2

exp1 –ne exp2
True if arithmetic expression exp1 is not equal to exp2

file1 –nt file2
True if file1 is newer than file2

file1 –ot file2
True if file1 is older than file2

file1 –ef file2
True if file1 is a hard link or symbolic link to file2 (this is different than the test command which only tests for hard links on OS/390)

(test_expr)
Grouping to override normal precedence; true if test_expr is true

! test_expr
Logical negation; true if test_expr is false

test_expr1 && test_expr2
Logical AND; true if both test_expr1 and test_expr2 are true

test_expr || **test_expr2**
Logical OR; true if either test_expr1 or test_expr2 is true

Patterns: Patterns tested in double-square-bracket conditions are composed of special characters and regular characters. Patterns follow the rules given in “Filename Generation” on page 503, except that the period (.) and the slash (/) are not treated specially. Note that pattern matching is similar to regular expression processing, but different in syntax.

case *word* **in** *[(][pattern[|pattern] &...* **)***command* **;;***] ... [(][pattern[|pattern] ...* **)***command* **;;** *] ...* **esac**

The **case** statement is similar to the **switch** statement of the C programming language or the **case** statement of Pascal. If the given *word* matches any one of the patterns separated by "or" bar (|) characters, **sh** runs the corresponding *command*. The .patterns should follow the rules given in "Filename Generation" on page 503, except that the period (**.**) and slash (/) are not treated specially. Patterns are matched in the order they are given, so more inclusive patterns should be mentioned later. You must use the double semicolon (**;;**) to delimit *command* and introduce the next *pattern*.

for *variable* **[in** *word ...***]** **do** *command* **done**

The **for** statement sets *variable* to each *word* argument in turn, and runs the set of *command*s once for each setting of *variable*. If you omit the **in** *word* part, **sh** sets *variable* to each positional parameter. You can divert the flow of control within the loop with the **break** or **continue** statements.

function *variable* {
    command
    ...
}

*variable***()** {
    command
    ...
}

Either one of these forms defines a **function** named *variable*, the body of which consists of the sequence of *command*s. You invoke a function just like any other command; when you actually call the function, **sh** saves the current positional parameters. The function's command-line arguments then replaces these parameters until the function finishes. **sh** also saves the current `ERR` and `EXIT` traps, as well as any flags manipulated by `EXIT` with the **set** command; these are restored when the function finishes. The function ends either by falling off the end of the code of the function body, or by reaching a **return** statement. If the function uses **typeset** to declare any variables in the function body, the variables are local to the function.

if *command* **then** *command* **[elif** *command* **then** *command] ...* **[else** *command]* **fi**

In the **if** statement, if the first (leftmost) *command* succeeds (returns a zero exit status), **sh** runs the *command* following **then**. Otherwise, **sh** runs the *command* (if any) following the **elif** (which is short for "else if"); if that succeeds, **sh** runs the *command* following the next **then**. If neither case succeeds, **sh** runs the *command* following the **else** (if any).

select *variable* **[in** *word ...***]** **do** *commands* **done**

The **select** statement can handle menulike interactions with the user. Its syntax is like the **for** statement. Each *word* is printed on the standard error file, one per line, with an accompanying number. If you omit the "**in** *word ...*" part, **sh** uses the positional parameters. **sh** then displays the value of the variable **PS3** to prompt the user to enter a numerical reply. If

the reply is an empty line, **sh** displays the menu again; otherwise, **sh** assigns the input line to the variable **REPLY**, sets *variable* to the *word* selected, and then runs the *commands*. **sh** does this over and over until the loop is ended by an interrupt, an end-of-file, or an explicit **break** statement in the *commands*.

until *command1* **do** *command2* **done**
The **until** statement runs *command1* and tests its exit status for success (zero) or failure (nonzero). If *command1* succeeds, the loop ends; otherwise, **sh** runs *command2* and then goes back to run and test *command1* again. **break** and **continue** commands in the *commands* can affect the operation of the loop.

while *command1* **do** *command2* **done**
The **while** statement works similarly to the **until** statement. However, the loop ends whenever *command1* is unsuccessful (nonzero exit status).

Shell reserved words are recognized only when they are the unquoted first token of a command. This lets you pass these reserved words as arguments to commands run from the shell. The full list of reserved words is:

```
!           done        function        while
[[          elif        if
{           else        select
}           esac        then
case        fi          time
do          for         until
```

Command Execution

Before running a *simple command,* the shell processes the command line, performing expansion, assignments, and redirection.

First, **sh** examines the command line and divides it into a series of *tokens,* which are either *operators* or *words*. An operator is either a control operator, which is described in “Command Syntax” on page 488. Or it can be a redirection operator, described in “File Descriptors and Redirection” on page 501. A word is any token that is not an operator.

Next, the shell expands words in the following order:

1. **sh** performs directory substitution.
2. **sh** performs parameter substitution, command substitution, or arithmetic substitution, as appropriate, in the order that the words appear on the command line, expanding each word to a *field* (see the appropriate sections).
3. **sh** scans each field produced in step 2 for unquoted characters from the **IFS** environment variable and further subdivides this field into one or more new fields.
4. **sh** expands any aliases to their definitions.
5. **sh** performs pathname expansion on each unquoted field from step 3.
6. **sh** removes all quote mechanisms (\, ', and ") that were present in the original word unless they have themselves been quoted.

The shell considers the first field of the expanded result to be a command.

The expanded simple command can contain variable assignments and redirections. Variable assignments affect the current execution environment. After expansion, the shell handles all redirection constructs, and the command, if one was found, it performs the redirection in a child shell environment (see “Shell Execution Environments” on page 504).

When a simple command contains a command name, variable assignments in the command affect only the execution of that command.

After the shell has expanded all appropriate arguments in a simple command, but before it performs filename generation, it examines the command name (if the command has one). **sh** first checks the names against currently defined aliases (see the **alias** command) and functions (see **function** under “Reserved-Word Commands” on page 489), and finally against the set of *built-in commands*: commands that the shell can run directly without searching for program files. Built-in commands are described in “Built-in Commands” on page 505 .

The **autoload** command, an alias of **typeset –fu**, identifies functions which are not yet defined. The first time an **autoload** function is called within the shell, the shell will search directories in the **FPATH** shell variable for a file with the same name as the function. If a matching file is found, it is assumed to contain the function definition of the same name. The file is read and executed in the current shell environment, storing the function in the shell's memory for subsequent execution. (Multiple function definitions may be contained in the same file. When the file is processed by the shell, all the functions will be defined. Every function definition in the file should be a link name to the file.)

If the command is a built-in or function, the shell executes it.

If the command name is not a function or a built-in command, the OS/390 shell looks for a program file or script file that contains an executable version of that command. The shell uses the following procedure to locate the program file:

- If the command name typed to the shell has slash (/) characters in its name, the command is taken to be a full pathname (absolute or relative). The shell tries to execute the contents of that file.
- Otherwise, the shell performs a path search. To do this, the shell obtains the value of the **PATH** environment variable. The value should be a list of directory names. **sh** searches under each directory for a file, the name of which matches the command name. If the **FPATH** shell variable is set, the shell will search the **PATH** and **FPATH** directories. If a file with a name matching the command name is found in the same directory in both **PATH** and **FPATH**, or if a matching file is found only in **FPATH**, this file will be read and executed in the current shell environment (defining the functions contained in the file). The shell will then execute the function matching the command name. This allows users to use **FPATH** for locating functions without the need to identify every function with the **autoload** command.

  If **FPATH** is not set, or the command is not found in **FPATH**, the shell executes the first matching file found in the **PATH** directories.

Command names can be marked as *tracked aliases.* The first time you run a command with a tracked alias, the shell does a normal **PATH** search. If the search is successful, the shell remembers the file that it finds. The next time you run a command with the same name, **sh** immediately runs the file found on the last

PATH search; there is no new search. This speeds up the time that it takes the shell to find the appropriate file.

The **set –h** command tells the shell that all commands should be treated as tracked aliases. See **alias** and **set** for more information.

Quoting

To let you override the special meaning of certain *words* or special characters, the shell provides several quoting mechanisms. In general, you can turn off the special meaning of any character by putting a backslash (\) in front of the character. This is called *escaping* the character.

For example, you can tell the shell to disregard the special meaning of the newline character by putting a backslash at the very end of a line. The shell ignores the escaped newline, and joins the next line of input to the end of the current line. In this way, you can enter long lines in a convenient and readable fashion.

Escaping characters by putting a backslash in front of them is the most direct way of telling the shell to disregard special meanings. However, it can be awkward and confusing if you have several characters to escape.

As an alternative, you can put arguments in various types of quotes. Different quote characters have different “strengths.” The single-quote characters are the strongest. When you enclose a command-line argument in single-quote characters, the shell disregards the special meanings of everything inside the single quotes. For example:

```
 echo
'*'
```

Double-quote characters are weaker. Inside double quotes, the shell performs command substitutions of the form:

$(*command*)

or

command

The shell does not perform such substitutions when they appear inside single quotes. In addition, the shell performs parameter substitutions of the form:

$(*command*)

or

command

$parameter

when they are inside the double quotes but not when they are inside single quotes. See “Parameter Substitution” on page 496. You can use the backslash to escape another character when they appear inside double quotes, but inside single quotes the shell ignores this special meaning.

The shell treats internal field separator characters (that is, characters in the value of the **IFS** variable) literally inside quoted arguments, whether they're quoted with double quotes or single quotes. This means that a quoted argument is considered a single entity, even if it contains **IFS** characters.

Quoting can override the special meanings of reserved words and aliases. For example, in:

```
"time" program
```

the quotes around **time** tell the shell not to interpret **time** as a shell reserved word. Instead, **sh** does a normal command search for a command named **time**.

You must always quote the following characters if you want **sh** to interpret them literally:

```
|   &   ;   <   >   (   )   $    '   "       \
<space>  <tab>  <newline>
```

The following characters need to be quoted in certain contexts if they are to be interpreted literally:

```
*   ?   [   #   %   =
~
```

Directory Substitution

When a word begins with an unquoted tilde (~), **sh** tries to perform directory substitution on the word. **sh** obtains all characters from the tilde (~) to the first slash (/) and uses this as a *user name*. **sh** looks for this name in the user profile, the file that contains information on all the system's users. If **sh** finds a matching name, it replaces ~name with the name of the user's *home directory,* as given in the matching RACF user profile entry.

For example, if you specify a filename as:

```
~jsmith/file
```

sh would look up jsmith's home directory and put that directory name in place of the **~jsmith** construct.

If you specify a ~without an accompanying name, **sh** replaces the ~with the current value of your **HOME** variable. For example:

```
echo ~
```

displays the name of your home directory. Similarly, **sh** replaces the construct ~+ with the value of the **PWD** variable (the name of the your working directory), and replaces the tilde hyphen (~–) with the value of **OLDPWD** (the name of your previous working directory). In variable assignments, tilde expansion is also performed after colons (:).

Parameter Substitution

The shell uses three types of parameters: positional parameters, special parameters, and variables. A positional parameter is represented with either a single digit (except 0) or one or more digits in curly braces. For example, 7 and {15} are both valid representations of positional parameters. Positional parameters are assigned values from the command line when you invoke **sh**.

A special parameter is represented with one of the following characters:

```
*    @    #    ?    !    -    $     0
```

The values to which special parameters expand are listed in the following paragraphs.

Variables are named parameters. For details on naming and declaring variables, see “Variables” on page 504.

The simplest way to use a parameter in a command line is to enter a dollar sign ($) followed by the name of the parameter. For example, if you enter the command:

```
echo $x
```

sh replaces $x with the value of the parameter *x* and then displays the results (because **echo** displays its arguments). Other ways to expand parameters are shown in the following paragraphs.

The following parameters are built in to the shell:

$1, $2, ... $9
Expands to the *d* positional parameter (where *d* is the single digit following the $). If there is no such parameter, $*d* expands to a null string.

$0 Expands to the name of the shell, the shell script, or a value assigned when you invoked the shell.

$# Expands to the number of positional parameters.

$@** Expands to the complete list of positional parameters. If **$@ is quoted, the result is separate arguments, each quoted. This means that:

```
"$@"
```

is equivalent to:

```
"$1" "$2" ...
```

$*** Expands to the complete list of positional parameters. If **$* is quoted, the result is concatenated into a single argument, with parameters separated by the first character of the value of **IFS** (see “Variables” on page 504). For example, if the first character of **IFS** is a blank, then:

```
"$*"
```

is equivalent to:

```
"$1 $2 ..."
```

$– Expands to all options that are in effect from previous calls to the **set** command and from options on the **sh** command line.

$? Expands to the exit status of the last command run.

$$ Expands to the current process number of the original parent shell.

$! Expands to the process number of the last asynchronous command.

These constructs are called *parameters* of the shell. They include the positional parameters, but are not restricted to the positional parameters.

We have already mentioned that you can expand a parameter by putting a $ in front of the parameter name. More sophisticated ways to expand parameters are:

${*parameter***}**
Expands any parameter.

${*number***}**
Expands to the positional parameter with the given number. (Remember that if you just enter $*d* to refer to the *d*th positional parameter, *d* can only be a single digit; with brace brackets, *number* can be greater than 9.) Since braces

mark the beginning and end of the name, you can have a letter or digit immediately following the expression.

${*variable***[***arithmetic expression***]}**
Expands to the value of an element in an array named *variable*. The *arithmetic expression* gives the subscript of the array. (See "Arithmetic Substitution" on page 499.)

${*variable* **[*]}**
Expands to all the elements in the array *variable*, separated by the first character of the value of **$IFS**.

${***variable* **[@]$}
When unquoted, is the same as ${ *variable*[*]} When quoted as "${*variable* [@]$} ," it expands to all the elements in the array *variable*, with each element quoted individually.

${#*parameter***}**
Expands to the number of characters in the value of the given *parameter*.

${#} Expands to the number of positional parameters.

${# *}
Expands to the number of positional parameters.

${#@}
Expands to the number of positional parameters.

${#*variable* **[*]}**
Expands to the number of elements in the array named *variable*. Elements that do not have assigned values do not count. For example, if you only assign values to elements `0` and `4`, the number of elements is `2`. Elements `1` through `3` do not count.

${*parameter***:-***word***}**
Expands to the value of *parameter* if it is defined and has a nonempty value; otherwise, it expands *word.* This means that you can use *word* as a default value if the parameter isn't defined.

${*parameter***-***word***}**
Is similar to the preceding construct, except that the parameter is expanded if defined, even if the value is empty.

${*variable***:=***word***}**
Expands *word* with parameter expansion and assigns the result to *variable*, provided that *variable* is not defined or has an empty value. The result is the expansion of *variable*, whether or not *word* was expanded.

${*variable***=***word***}**
Is similar to the preceding construct, except that the *variable* must be undefined (it cannot just be null) for *word* to be expanded.

${*parameter***:?***word***}**
Expands to the value of *parameter* provided that it is defined and non-empty. If *parameter* isn't defined or is null, **sh** expands and displays *word* as a message. If *word* is empty, **sh** displays a default message. After a noninteractive shell has displayed a message, it ends.

${*parameter***?***word***}**
Is similar to the preceding construct, except that **sh** displays *word* only if *parameter* is undefined.

${*parameter***:+***word***}**
Expands to *word*, provided that *parameter* is defined and nonempty.

${*parameter***+***word***}**
Expands to *word*, provided that *parameter* is defined.

${*parameter***#***pattern***}**
Attempts to match *pattern* against the value of the specified *parameter*. The *pattern* is the same as a case *pattern*. **sh** searches for the shortest prefix of the value of *parameter* that matches *pattern*. If **sh** finds no match, the previous construct expands to the value of *parameter*; otherwise, the portion of the value that matched *pattern* is deleted from the expansion.

${*parameter***##***pattern***}**
Is similar to the preceding construct, except that **sh** deletes the longest part that matches *pattern* if it finds such a match.

${*parameter***%***pattern***}**
Searches for the shortest suffix of the value of *parameter* matching *pattern* and deletes the matching string from the expansion.

${*parameter***%%***pattern***}**
Is similar to the preceding construct, except that **sh** deletes the longest part that matches *pattern* if it finds such a match.

Arithmetic Substitution

Arithmetic substitution is available with the syntax:

$((*arithmetic expression*))

or:

$[*arithmetic expression*]

This sequence is replaced with the value of *arithmetic expression*. Arithmetic expressions consist of expanded variables, numeric constants, and operators. Numeric constants have the form:

[*base#*]*number*

where the optional *base* is a decimal integer between 2 and 36 inclusive, and *number* is any nonnegative number in the given base. The default base is 10. Undefined variables evaluate to zero.

The operators are listed in decreasing order of precedence in Figure 12. Operators sharing a heading have the same precedence. Evaluation within a precedence group is from left to right, except for the assignment operator, which evaluates from right to left.

Figure 12 (Page 1 of 2). Shell Operators (sh command)

Unary Operators	
–	Unary minus
!	Logical negation
+ ~	Identity, bitwise negation
Multiplicative Operators	
* / %	Multiplication, division, remainder
Additive Operators	

Figure 12 (Page 2 of 2). Shell Operators (sh command)	
Unary Operators	
+ –	Addition, subtraction
Bitwise Shift Operators	
<< >>	Bitwise shift right, bitwise shift left
Relational Operators	
< >	Less than, greater than
<= >=	Less than or equal, greater than or equal
== !=	Equal to, not equal to
Bitwise AND Operator	
&	AND
Bitwise Exclusive OR Operator	
^	Exclusive OR
Bitwise Inclusive OR Operator	
|	Inclusive OR
Logical AND Operator	
&&	Logical AND
Logical OR Operator	
||	Logical OR
Conditional Operator	
? :	If-else
Assignment Operator	
= *= /= %= += –= <<= >>= &= ^= |=	Assignment

You do not need the $(( )) syntax to enclose an arithmetic expression in these situations:

- In assignment to an integer variable. (See **typeset**.)
- As an argument to the following built-in shell commands:

  break exit return continue let shift
- When used as arguments in test numeric comparisons (**–eq**, **–ge**, **–gt**, **–le**, **–lt**, and **–ne**). See **test**.

Command Substitution

In *command substitution,* **sh** uses the expansion of the standard output of one command in the command line for a second command. There are two syntaxes.

The first syntax (called *backquoting*) surrounds a command with grave accents `, as in:

```
ls `cat list`
```

To process this command line, **sh** first runs the **cat** command and collects its standard output. The shell then breaks this output into arguments and puts the result into the command line of the **ls** command. The previous command therefore lists the attributes of all files, the names of which are contained in the file **list**.

This syntax is easy to type, but is not useful if you want to put one command substitution inside another (*nesting* command substitutions). A more useful syntax is:

`$(`*command*`)`

as in:

```
ed $(grep –f –l function $(find . –name '*.c'))
```

This command uses **find** to search the current directory and its subdirectories to find all files, the names of which end in **.c**. It then uses **grep –f** to search each such file for those that contain the string `function`. Finally, it calls **ed** to edit each such file.

There is a historical inconsistency in the backquoting syntax. A backslash (\) within a backquoted command is interpreted differently depending on its context. Backslashes are interpreted literally unless they precede a dollar sign ($), grave accent ( ), or another backslash (\). In these cases, the leading backslash becomes an escape character to force the literal interpretation of the $, , or \. Consequently, the command:

```
echo '\$x'
```

issued at system level produces the output:

```
\$x
```

whereas the same command nested in a backquoted syntax:

```
echo  echo '\$x'
```

produces the output:

```
$x
```

We recommend the `$(`*command*`)` syntax for command substitutions.

sh performs command substitutions as if a new copy of the shell is invoked to run the command. This affects the behavior of `$-` (standing for the list of options passed to the shell). If a command substitution contains `$-`, the expansion of `$-` does not include the **–i** option, since the command is being run by a noninteractive shell.

File Descriptors and Redirection

The shell sometimes refers to files using *file descriptors*. A file descriptor is a number in the range `0` to `9`. It may have any number of digits. For example, the file descriptors `001` and `01` are identical to file descriptor `1`. Various operations (for example, **exec**) can associate a file descriptor with a particular file.

Some file descriptors are set up at the time the shell starts up. These are the standard input/output streams:

- Standard input (file descriptor 0)
- Standard output (file descriptor 1)
- Standard error (file descriptor 2)

Commands running under the shell can use these descriptors and streams too. When a command runs under the shell, the streams are normally associated with your terminal. However, you can *redirect* these file descriptors to associate them with other files (so that I/O on the stream takes place on the associated file instead of your terminal). In fact, the shell lets you redirect the I/O streams associated with file descriptors `0` through `9`, using the following command-line constructs.

number<*file*
: Uses *file* for input on the file descriptor, the number of which is *number*. If you omit *number*, as in <*file*, the default is `0`; this redirects the standard input.

number>*file*
: Uses *file* for output on the file descriptor, the number of which is *number*. If you omit *number*, as in >*file*, the default is `1`; this redirects the standard output. The shell creates the file if it does not already exist. The redirection fails if the file already exists and **noclobber** is set (see **set**).

number>|*file*
: Is similar to *number*>*file* but if *file* already exists, the output written to the file overwrites its current contents.

number<> *file*
: Uses *file* for input and output with the file descriptor, the number of which is *number*. This is most useful when the file is another terminal or modem line. If you omit *number*, as in <> *file*, the default *number* is zero; this redirects the standard input. Output written to the file overwrites the current contents of the file (if any). The shell creates the file if it does not already exist.

number>>*name*
: Is similar to *number* > *file*, except that output is appended to the current contents of the file (if any).

number<<[–] *name*
: Lets you specify input to a command from your terminal (or from the body of a shell script). This notation is known as a *here-document*. The shell reads from the standard input and feeds that as input to file descriptor *number* until it finds a line that exactly matches the given *name*. If you omit *number*, the default is the standard input. For example, to process the command:

  ```
  cat <<abc >out
  ```

  the shell reads input from the terminal until you enter a line that consists of the word `abc`. This input is passed as the standard input to the **cat** command, which then copies the text to the file **out**.

  If any character of *name* is quoted or escaped, **sh** does not perform substitutions on the input; instead, it performs variable and command substitutions, respecting the usual quoting and escape conventions. If you put – before *name*, **sh** deletes all leading tabs in the *here-document*.

number1<&*number2*
: Makes the input file descriptor *number1* a duplicate of file descriptor *number2*. If you omit *number1*, the default is the standard input (file descriptor 0). For example, <&4 makes the standard input a duplicate of file descriptor 4. In this case, entering input on 4 has the same effect as entering input on standard input (**stdin**).

number1>&*number2*
: Makes the output file descriptor *number2* a duplicate of file descriptor *number2*. If you omit *number2*, the default is the standard output (file

descriptor 1). For example, >&2 makes the standard output a duplicate of file descriptor 2 (the standard error). In this case, writing output on **stdout** has the same effect as writing output on **stderr**.

number<&–
: Closes input descriptor *number*. If you omit *number*, it closes the standard input.

number>&–
: Closes output descriptor *number*. If you omit *number*, it closes the standard output.

Normally, redirection applies only to the command where the redirection construct appears; however, see **exec**.

The order of *redirection* specifications is significant, since an earlier redirection can affect a later one. However, these specifications can be freely intermixed with other command arguments. Since the shell takes care of the redirection, the redirection constructs are not passed to the command itself.

Note: The shell performs the implicit redirections needed for pipelines before performing any explicit redirections.

Filename Generation

The characters * ? [are called *glob characters,* or *wildcard characters*. If an unquoted argument contains one or more glob characters, the shell processes the argument for filename generation. The glob characters are part of *glob patterns,* which represent file and directory names. These patterns are similar to regular expressions, but differ in syntax, since they are intended to match filenames and words (not arbitrary strings). The special constructions that may appear in glob patterns are:

? Matches exactly one character of a filename, except for the separator character / and a **.** at the beginning of a filename. ? only matches an actual filename character and does not match nonexistent characters at the end of the filename. ? is analogous to the metacharacter **.** in regular expressions.

***** Matches zero or more characters in a filename, subject to the same restrictions as ?. * is analogous to the regular expression **.***.

[*chars***]**
: Defines a *class* of characters; the glob pattern matches any single character in the class. A class can contain a range of characters by writing the first character in the range, a dash –, and the last character. For example, [A–Za–z], in the POSIX locale, stands for all the uppercase and lowercase letters. If you want a literal – character (or other glob character) in the class, use the backslash to escape the character, causing it to lose it's special meaning within the pattern expression. If the first character inside the brackets is an exclamation mark (!), the pattern matches any single character that is *not* in the class.

Some sample patterns are:

[!a-f]*.c Matches all **.c** files beginning with something other than the letters from a through f.

/???/?.? Matches all files that are under the root directory in a directory with a three-letter name, and that have a basename containing one character followed by a **.** followed by another single character.

***/*.[chyl]** Matches all **.c**, **.h**, **.y**, and **.l** files in a subdirectory of the working directory.

~mks/*.ksh Matches all shell scripts in the home directory of user `mks`

(see "Directory Substitution" on page 496 for the use of ~).

If no files match the pattern, **sh** leaves the argument untouched. If the **set** option **–f** or **"–o noglob"** is in effect, the shell does not perform filename generation.

Attention: Doublebyte characters in a filename may cause problems. For instance, if you use a doublebyte character in which one of the bytes is a **.** (dot) or / (slash), the file system treats this as part of the pathname.

Variables

The shell maintains variables and can expand them where they are used in command lines; see "Parameter Substitution" on page 496 or details.

A variable name must begin with an uppercase or lowercase letter or an underscore (_). Subsequent characters in the name, if any, can be uppercase or lowercase letters, underscores, or digits `0` through `9`. You can assign a value to a variable with:

variable=value

You can implicitly declare a variable as an array by using a subscript expression when assigning a value, as in:

variable[*arithmetic expression*]*=value*

You can use a subscripted array variable anywhere that the shell allows an ordinary variable. For the syntax of an arithmetic expression, see "Arithmetic Substitution" on page 499. Also see **typeset**, **export**, and **readonly** for details about the attributes of shell variables, and how shell variables can be exported to child processes.

For a list of variables that the shell either sets or understands, see "Shell Variables" on page 507.

Shell Execution Environments

A shell execution environment is the set of conditions affecting most commands run within the shell. It consists of:

- Open files
- The working directory (see **cd**)
- The file creation mask (see **umask**)
- The traps currently set (see **trap**)
- The shell parameters (see **set** and **export**)
- The shell functions currently defined (see "Command Execution" on page 493)
- Options (see **set**)

A child shell environment starts as a duplicate of the shell environment, except that traps caught by the shell are set to default values in the child shell. Since the child

shell environment starts as a duplicate, the value of the **ENV** environment variable is not run. Changes made to a child shell environment do not affect the shell environment.

Command substitutions, commands within parentheses "(*command*)," and commands to be run asynchronously ("*command*&")—all run in child shell environments. Each command in a pipeline "*command*|*command*" runs in a child shell environment.

Shell commands also run in a separate environment that does not affect the shell environment, except for certain built-in commands (for example, **cd** and **umask**) that explicitly alter the shell environment. The environment of a shell command is set up by the shell to include the following:

- Open files, subject to redirection.
- Working directory (see **cd**).
- File creation mask (see **umask**).
- Traps; traps caught by the shell are set to default values and traps ignored by the shell are ignored by the command.
- Variables defined inside the shell and having the `export` attribute.

Built-in Commands

This section lists the commands that are built into the shell. Such commands are built into the shell to increase performance of shell scripts or to access the shell's internal data structures and variables. These internal commands are designed to have semantics indistinguishable from external commands.

:	**chmod**	**false**	**ln**	**read**	**trap**	**whence**
.	**chown**	**fc**	**login**	**readonly**	**true**	
[	**comm**	**functions**	**ls**	**return**	**type**	
alias	**command**	**getopts**	**mkdir**	**rm**	**typeset**	
autoload	**continue**	**history**	**mv**	**set**	**ulimit**	
basename	**echo**	**integer**	**newgrp**	**shift**	**umask**	
bg	**eval**	**jobs**	**print**	**stop**	**unalias**	
break	**exec**	**kill**	**printf**	**suspend**	**unlink**	
cat	**exit**	**let**	**pwd**	**test**	**unset**	
cd	**export**	**link**	**r**	**times**	**wait**	

POSIX.2 recognizes a subset of these commands as *special* built-ins. Syntax errors in special built-in commands may cause a shell executing that command to terminate, while syntax errors in regular built-in commands will not cause the shell executing that command to terminate. If a special built-in command encountering a syntax error does not terminate the shell, its exit value is nonzero.

Also, shell variable assignments included on shell command lines that invoke special built-in commands remain in effect after the built-in command completes; this is not the case with regular built-in commands or other utilities.

The special built-in commands are:

:	**continue**	**exit**	**readonly**	**shift**	**unset**
.	**eval**	**export**	**return**	**trap**	
break	**exec**	**expr**	**set**	**typeset**	

As well as built-in commands, the shell has a set of predefined aliases:

autoload	**functions**	**history**	**nohup**	**stop**
hash	**integer**	**r**	**suspend**	

See **alias** for details.

Examples

Software distributed over computer networks such as Usenet is often distributed in a form known as a *shell archive*. In essence, a shell archive is a shell script containing the data of one or more files, plus commands to reconstruct the data files and check that the data was sent correctly. The following shows a sample shell archive:

```
# This is a shell archive.
# It contains the one file "frag.ksh"
# To extract contents, type
# sh file
#
if      [ -f frag.ksh ]
then    echo frag.ksh exists: will not overwrite
else
    echo extracting frag.ksh
    sed 's/⅛X//' >frag.ksh <<_EOF_
X# This is frag.ksh
X# Not very interesting, really.
Xecho frag.ksh here!
_EOF_
    if [ " sum frag.ksh|awk '{print $1}' " != 52575 ]
    then        echo frag.ksh damaged in transit
    fi
fi
```

The following is a simple script to produce as much of the Fibonacci sequence as can be calculated in integers:

```
# Print out Fibonacci sequence; start sequence
# with first two positional parameters:
# default 1 1
typeset -i x=${1:-1} y=${2:-1} z
while   [ x -gt 0 ]     # until overflow
do
        echo $x
        let z=y+x x=y y=z
done
```

The following implements the **basename** command as a shell function:

```
# basename command as shell function
function basename {
        case $# in
        1)      ;;
        2)      eval set \${1%$2} ;;
        *)      echo Usage:       $0 pathname '[suffix]     '
                return 1 ;;
        esac
        echo ${1##*/}
        return 0
}
```

Shell Variables

You cannot use doublebyte characters for a shell variable name, but you can use them for shell variable values. Doublebyte characters in filenames and pathnames are treated as singlebyte characters.

Note: Shell variables that are exported are called "Environment Variables", and are made available in the environment of all commands run from the shell. Figure 13 contains a list of built-in shell variables and also includes frequently-used environment variables. For more information on environment variables used by the C-RTL, see the *OS/390 C/C++ Programming Guide*. A list of other environment variables can be found in Appendix J, “Environment Variables” on page 839.

Figure 13 lists frequently-used shell variables and their purposes.

Figure 13 (Page 1 of 4). Built-in Shell Variables (sh command)

Variable	Purpose
_	(Underscore) For every command that is run as a child of the shell, **sh** sets this variable to the full pathname of the executable file and passes this value through the environment to that child process. When processing the **MAILPATH** variable, this variable holds the value of the corresponding mail file.
~	(Tilde) expands to value of the **HOME** directory.
CDPATH	Contains a list of directories for the **cd** command to search. Directory names are separated with colons. **CDPATH** works like the **PATH** variable.
COLUMNS	Used by several commands to define the width of the terminal output device.
EDITOR	Enables the corresponding editing mode (see **set** and **shedit**) when using **vi**, **emacs**, or **gmacs**.
ERRNO	Contains the system error number of the most recently failed system call. The shell sets this variable only for errors that occur in the current environment. Assigning a value of `0` to this variable clears it.
FCEDIT	Contains the name of the default editor for the **fc** command. If this variable is not set, the default is the **ed** command.
FPATH	Contains a list of directories that the system searches to find executable functions. Directories in this list are separated with colons. **sh** searches each directory in the order specified in the list until it finds a matching function. If you want the shell to search the working directory, put a dot (**.**) or a null string in the list of directories (for example, to tell the shell to search the working directory first, start the list with a colon or semicolon).
HISTFILE	Contains the pathname of a file to be used as the history file. When the shell starts, the value of this variable overrides the default history file.
HISTSIZE	Contains the maximum number of commands that the shell keeps in the history file. If this variable contains a valid number when the shell starts, it overrides the default of 127.
HOME	Contains your home directory. This is also the default directory for the **cd** command. The **HOME** variable is set automatically from the RACF user profile when the user logs in.

Figure 13 (Page 2 of 4). Built-in Shell Variables (sh command)

Variable	Purpose
IFS	Contains a series of characters to be used as *internal field separator* characters. Any of these characters can separate arguments in unquoted command substitutions such as `command` or `$(command)`, or in parameter substitutions. In addition, the shell uses these characters to separate values put into variables with the **read** command. Finally, the first character in the value of **IFS** separates the positional parameters in `$*` expansion. By default, **IFS** contains space, tab, and newline.
LANG	Contains the default locale value.
LINENO	Contains the number of the line currently being run by a shell script.
LINES	Used by several commands to define the number of lines on the terminal output device.
LIBPATH	Used to specify the directory to search for a DLL (Dynamic Link Library) filename. If it is not set, the working directory is searched. For more information, see **dlload** in *OS/390 C/C++ Run-Time Library Reference*. **Attention: LIBPATH** can be updated by the **_CEE_ENVFILE** environment variable. For more information on **_CEE_ENVFILE**, see the *OS/390 C/C++ Programming Guide*.
LOCPATH	Tells the **setlocale()** function the name of the directory in the HFS from which to load locale object files. (**localedef** produces locale object files by processing locale source files.)
LOGNAME	Contains the user login name. This is set automatically from the RACF user profile when the user logs in.
MAILCHECK	Contains the number of seconds of elapsed time that must pass before the system checks for mail; the default value is 600 seconds. When using the **MAIL** or **MAILPATH** variables, the shell checks for mail before issuing a prompt.
MAILPATH	Contains a list of mailbox files. This overrides the **MAIL** variable. The mailbox list is separated by colons. If any name is followed by **?***message* or **%***message*, **sh** displays the message if the corresponding file has changed. **sh** performs parameter and command substitution on *message*, and the variable _. (temporarily) expands to the name of the mailbox file. If no **?***message* or **%** *message* is present, the default message is `you have mail in $_`**.**
MANPATH	Contains a list of paths to search for man pages.
MBOX	Contains the pathname of your personal mailbox, usually **$HOME/mbox**, used to store messages that have been read from your system mailbox. This variable is usually set in your **.profile**.
NLSPATH	Specifies where the message catalogs are to be found.
OLDPWD	Contains the name of the directory you were previously working in. The **cd** command sets this variable.
PATH	Contains a list of directories that the system searches to find executable commands. Directories in this list are separated with colons. **sh** searches each directory in the order specified in the list until it finds a matching executable. If you want the shell to search the working directory, put a dot (**.**) or a null string in the list of directories (for example, to tell the shell to search the working directory first, start the list with a colon or semicolon).
PID	Contains the decimal value of the process ID of the parent of the shell. See **ps**.

Figure 13 (Page 3 of 4). Built-in Shell Variables (sh command)

Variable	Purpose
PS1	Contains the primary prompt string used when the shell is interactive. The default value is a dollar sign followed by a space ($). The shell expands parameters before the prompt is printed. A single exclamation mark (!) in the prompt string is replaced by the command number from the history list; see the **fc** command. For a real exclamation mark in the prompt, use !!. This variable is usually set in your **.profile**.
PS2	Contains the secondary prompt, or continuation prompt, used when completing the input of such things as reserved-word commands, quoted strings, and here documents. The default value of this variable is a greater than sign followed by a space (>).
PS3	Contains the prompt string used in connection with the **select** reserved word. The default value is a number sign followed by a question mark and a space (#?).
PS4	Contains the prefix for traced commands with **set -x**. The default value is a plus sign followed by a space (+).
PWD	Contains the name of the working directory. When the shell starts, the working directory name is assigned to **PWD** unless the variable already has a value.
RANDOM	Returns a random integer. Setting this variable sets a new seed for the random number generator.
SECONDS	Contains elapsed time. The value of this variable grows by 1 for each elapsed second of real time. Any value assigned to this variable sets the **SECONDS** counter to that value; initially the shell sets the value to 0.
SHELL	Contains the full pathname of the current shell. It is not set by the shell, but is used by various other commands to invoke the shell. This is set automatically from the RACF user profile when the user logs in.
STEPLIB	Identifies a **STEPLIB** variable to be used in building a process image for running an executable file. A STEPLIB is a set of private libraries used to store a new or test version of an application program, such as a new version of a runtime library. **STEPLIB** can be set to the values CURRENT or NONE or to a list of MVS data set names. The default is CURRENT, which passes on the TASKLIB, STEPLIB, or JOBLIB allocations that are part of the invoker's MVS program search order environment to the process image created for an executable file. The value NONE indicates you do not want a STEPLIB environment for executable files. You can specify up to 255 MVS data set names, separated by colons, as a list of data sets used to build a **STEPLIB** variable. Refer to *OS/390 UNIX System Services Planning* for more information about building a STEPLIB environment.
TMOUT	Contains the number of seconds before user input times out. If user input has not been received within this length of time, the shell ends.
TMPDIR	Is the pathname of the directory being used for temporary files. If it is not set, the OS/390 shell uses **/tmp.**

Figure 13 (Page 4 of 4). Built-in Shell Variables (sh command)	
Variable	**Purpose**
TZ	Contains the system time zone value used for displaying date and time. You can set the **TZ** variable in your **$HOME/.profile** file used during shell startup. The system administrator can also define a **TZ** default for all shell users in the **/etc/profile** file. If you are not in the same time zone, you can set **TZ** yourself. The system administrator can also define **TZ** for the **/etc/init** process in the **/etc/init.options** file.
VISUAL	Overrides the **EDITOR** environment variable in setting **vi**, **emacs**, or **gmacs** editing modes (see **shedit**).

Files

sh_history
The default history storage file.

.profile
The user profile for login shell.

/etc/profile
The systemwide profile for login shells.

/tmp/sh*
Temporary files for here-documents, command substitution, history reexecution, and so on. The default directory **/tmp** can be overridden by setting the shell variable **TMPDIR** to the name of some other directory.

/etc/suid_profile
Used instead of **$HOME/.profile** or **.profile** under the privileged option or when the real and effective UIDs are different.

Localization

sh uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_SYNTAX**

See Appendix F, “Localization” on page 817 for more information.

Exit Values

0 Successful completion

1 Failure due to any of the following:

- The shell was invoked with an incorrect option.
- The shell was invoked to run a shell script and the command.
- A command syntax error.
- A redirection error.
- A variable expansion error.

Otherwise, the exit status of the shell defaults to the exit status of the last command run by the shell. This default can be overridden by explicit use of the **exit** or **return** commands. The exit status of a pipeline is the exit status of the last command in the pipeline.

Messages

Ambiguous redirection
A redirection construct expanded to more than one pathname.

Argument too long
Any single argument to a command is limited in length (see "Limits" on page 512). Command and parameter substitution may exceed this limit.

Cannot restore privileged state
This message occurs only when the implementation of POSIX does not support the *saved IDs* option (*_POSIX_SAVED_IDS*). The message is generated if you tried to use a saved ID feature to return to a privileged state.

File *file* **already exists**
You are attempting to redirect output into an existing file, but you have turned on the **noclobber** option (see the **set** command). If you really want to redirect output into an existing file, use the construct `>|`*filename*, or turn off the option with:

```
set +o noclobber
```

File descriptor number already redirected
You attempted to redirect a file descriptor that was already being redirected in the same command. You can redirect a file descriptor only once.

Hangup
The shell received a *hangup* signal. This signal typically arises when a communication line is disconnected—for example, when a phone connection is cut off.

In base#number: base must be in [2,36]
In a number of the form `base#number`, the value of the base was larger than 36 or less than 2. The only valid range for bases is from 2 through 36.

Invalid subscript
A shell array was indexed with a subscript that was outside the defined bounds.

Illegal instruction
The shell received an illegal instruction signal. This signal typically occurs when a process tries to execute something that is not a valid machine instruction recognized by the hardware.

Misplaced subscript *array name*
The subscript for an array was missing or incorrect.

name **is not an identifier**
You attempted to use a nonalphanumeric *name*.

*name***: readonly variable**
The given *name* is a read-only variable, and cannot be removed or changed (see **readonly**).

*name***: no expansion of unset variable**
The shell is operating with **set –u**, and you used an unset variable in a substitution. For more information, see the **set** command.

No file descriptor available for redirection
When a file descriptor is redirected, the old value is remembered by the shell by a duplication to yet another file descriptor. The total number of file descriptors is limited by the system; hence, the shell may run out, even though your command appears to be using far fewer than the maximum number of descriptors.

Nested aliases
You have more than nine levels of aliases. For example:

```
alias a1=a2 a2=a3 a3=a4 ... a10=command
```

causes this error.

Pipe for coprocess
The shell cannot create a pipe for a coprocess. This may mean that your session or the system as a whole has already set up its maximum number of pipes.

...: restricted
If the shell has been invoked as a restricted shell, certain things are disallowed—for example, the **cd** command, setting **PATH**, and output redirection.

Temporary file error using here document
sh tried to create a temporary file holding the contents of a <<*word* here-document. However, the temporary file could not be created. This may indicate a lack of space on the disk where temporary files are created.

Word after ... expanded to more than one argument
In a context where only one argument was expected, a construct expanded to more than one argument.

Limits

The maximum length of an executable filename, including subdirectories and extensions, is 1023 bytes.

Portability

POSIX.2, X/Open Portability Guide.

The construct $[*arithmetic expression*] is an extension of the POSIX standard.

Related Information

alias, **break**, **cd**, **continue**, **dot**, **echo**, **eval**, **exec**, **exit**, **export**, **fc**, **getopts**, **let**, **print**, **ps**, **pwd**, **read**, **readonly**, **return**, **set**, **shift**, **test**, **time**, **trap**, **true**, **typeset**, **ulimit**, **unalias**, **unset**, **whence**. **shedit**

Appendix I also explains how to set the local time zone with the **TZ** environment variable.

shedit — Interactive command and history editing in the shell

Format

set –o *editmode*
EDITOR=*editprog*
VISUAL= *editprog*

Usage Notes

POSIX uses a number of keys for such things as *erase* and *kill* processing. By default, the shell leaves command-line editing to POSIX, using these familiar editing keys. However, these functions are not particularly powerful or friendly. As an alternative, the shell has built-in facilities for interactive command editing and file name generation that not only aid in composing new commands but also make it easy for you to modify and reexecute previous commands. This capability is distinct from that provided by the **fc** command, which passes previous command lines to a separate program for editing. The built-in facilities mimic the **emacs**, **gmacs**, or **vi** screen editors, and enable the following commands (see **set** and **vi** for details).

```
set –o emacs
set –o gmacs
set –o vi
```

These facilities are also enabled (with the corresponding option set) by assigning a value ending in **vi** to the environment variables **EDITOR** or **VISUAL**. (See **sh**.)

Unlike full-screen editors, shell editing uses a one-line window, extending from the end of the prompt to the next-to-last column. Multiline history entries are displayed with newlines represented as ¼J.

The number of columns on the output device is obtained from the **COLUMNS** environment variable if defined; otherwise, it is assumed to be 80.

A command line that extends into the rightmost column can be scrolled horizontally. If you try to move the cursor beyond the edge of the window, the line is scrolled to approximately center the cursor in the window. The second last column displays a character marking a scrollable line: < indicates extra data off the left, > indicates extra data off the right, and * indicates extra data off both sides.

emacs/gmacs Editing Mode

When the **emacs/gmacs** editing mode has been enabled, ordinary printable characters from the keyboard are entered in the command line and echoed. Various control characters introduce command sequences for such things as moving the cursor, scrolling through the command history, and modifying the current command. The only difference between **emacs** and **gmacs** is in the handling of **Ctrl-T**. (See the description of **Ctrl-T** in 515.)

The command sequences recognized are listed in functional groups. The notation **Meta–** represents **EscK**, followed by the letter. The terminology is historical. Many commands accept an optional preceding count which is entered in decimal as **Meta-***digits*, or as **Ctrl-**, which multiplies the current count (initially 1) by 4.

Cursor Movement

*n***Ctrl-B** Moves the cursor back *n* characters.

*n***Ctrl-F** Moves the cursor forward *n* characters.

Ctrl-A Moves the cursor to beginning of line.

Ctrl-E Moves the cursor to end of line.

*n***Meta–b**
Moves the cursor back to the *n*th previous beginning of word (string of alphanumerics).

*n***Meta–f**
Moves the cursor forward to *n*th beginning of word.

Ctrl–]*c* Moves the cursor forward to next character *c* on current line.

Meta–*space*
Sets mark at cursor position.

Ctrl-@ Sets mark at cursor position.

Ctrl-x Ctrl-X
Exchanges cursor position and mark.

Line Search

These commands display a different history line.

*n***Ctrl-P** Selects the *n*th previous command line from history.

*n***Ctrl-N** Selects the *n*th next command line from history.

Meta–< Selects the earliest command line from history.

Meta–> Selects the latest command line from history.

*n***Ctrl-R***stringEnter*
Selects the *n*th previous command line matching *string*. If *n* is zero, then select the next matching command after the current line.

Text Change

n erase Deletes *n* characters to the left of the cursor. This is the erase character.

*n***Backspace**
Deletes *n* characters to the left of the cursor.

*n***Ctrl-H** Deletes *n* characters to the left of the cursor.

*n***Ctrl-D** Deletes *n* characters to the right of the cursor. If the current line is empty, the shell is ended.

*n***Meta–Ctrl-H**
Deletes to the *n*th beginning of word before the cursor.

*n***Meta–h**
Deletes to the *n*th beginning of word before the cursor.

*n***Meta-d**
Deletes to the *n*th beginning of word after the cursor.

*n***Ctrl-K** Deletes from the cursor to the end of line. If *n* is zero, then deletes from the beginning of line to the cursor.

kill Deletes the entire current line. This is the line kill character.

Ctrl-G Deletes the entire current line.

Ctrl-W Deletes from cursor position to the mark (set with **Meta-***space* or **Ctrl-@**.

Ctrl-T In **emacs** mode, transposes the current character with the previous character and moves the cursor forward. If the cursor is at the end of the line, or in **gmacs** mode, transposes the previous two characters.

Ctrl-Y Restores the last text deleted in **emacs** mode.

Ctrl-C Capitalizes character under cursor.

Ctrl- Capitalizes character under cursor.

Meta-c Capitalizes word to right of cursor.

Meta-l Lowercases word to right of cursor.

Meta-u Uppercases word to right of cursor.

*n***Meta-.** Inserts the *n*th word of the previous command. If *n* is not given or it is zero, inserts the last word of the previous command.

*n***Meta-_**
Inserts the *n*th word of the previous command. If *n* is not given or it is zero, inserts the last word of the previous command.

Meta-* Replaces the current word with the list of files which would match that word with an * appended.

Meta-Esc
Used to complete a pathname. If there is only one existing pathname that matches as much as you've typed, the pathname is completed and a space is added after the complete pathname. If there are several matching pathnames, the shell expands what you've typed by adding all the characters that are common to all matching pathnames.

Meta-= Lists all pathnames matching the current word.

Miscellaneous

Ctrl-J
Executes the current command line.

Ctrl-M
Executes the current command line.

Ctrl-L
Redisplays the current command line.

Ctrl-O
Remembers the next command line, executes the current command line, then selects the remembered line.

Ctrl-U
Multiplies the count on the following command by 4 (for each **Ctrl-U**.

Ctrl-V
Displays the version of the shell.

– Takes the next character literally. Thus, you can enter command and control characters in a command line or search string.

eof Terminates the shell. This is the end-of-file character.

Ctrl-D
Terminates the shell.

Meta–*n*
Enters a count for the following command.

vi Editing Mode

When the **vi** editing facilities have been enabled, the shell is initially in input mode after each new prompt. Keyboard input is normally inserted at the current position in the current command line; the exceptions are the following action keys.

erase Deletes the character to the left of the cursor. This is the erase character.

Backspace
Deletes the character to the left of the cursor.

eof Terminates the shell. This is the end-of-file character.

Ctrl-D Terminates the shell.

Ctrl-W Deletes the word (white-space delimited string) to the left of the cursor.

kill Deletes the current line. This is the line kill character.

Ctrl-X Deletes the current line.

Ctrl-J Deletes the current line.

Ctrl-M Deletes the current line.

Enter Executes the current line.

Esc Switches from input mode to command mode.

If you press the **Esc** key, the shell enters command mode and keyboard input is interpreted as commands to reposition the cursor, scroll through the command history, delete or change text, or reenter input mode. In command mode, input is not echoed; it is acted upon. Many commands take an optional count, *n*, which is entered as a preceding decimal number (not echoed); the command is executed that number of times. Except where otherwise noted, *n* defaults to 1.

Ctrl-V Takes the next character literally; useful for entering any of these action keys as text.

**** Escapes the following action key. If the next character is any action key except **Ctrl-J**, **Ctrl-M**, or **Enter**, the **–** is erased and the escaped character is entered literally. Otherwise, the **–** is entered and the next character is treated normally.

Cursor Movement

These commands reposition the cursor in the command line.

*n***h** Moves back *n* characters.

*n***1** Moves forward *n* characters.

0 Moves to the first character on the line.

Moves to the first nonblank character on the line.

$ Moves to the last character on the line.

*n***w** Moves to the beginning of the *n*the next word (string of alphanumerics, or of nonblank nonalphanumerics).

*n***W** Moves to the beginning of the *n*the next fullword (string of nonblanks).

*n***b** Moves to the *n*the previous beginning of word.

*n***B** Moves to the *n*the previous beginning of fullword.

*n***e** Moves to the *n*the next end of word.

*n***E** Moves to the *n*the next end of fullword.

*n***f***c* Moves to the *n*the next character *c*.

*n***F***c* Moves to the *n*the previous character *c*.

*n***t***c* Moves to the character before the *n*the next character *c*.

*n***T***c* Moves to the character after the *n*the previous character *c*.

*n***;** Repeats the previous **f**, **F**, **t**, or **T** command.

*n***,** Repeats the previous **f**, **F**, **t**, or **T** command, but in the opposite direction.

Line Search

These commands change the current displayed command line.

*n***j** Selects the *n*the next command line from history.

n+ Selects the *n*the next command line from history.

*n***k** Selects the *n*the previous command line from history.

n– Selects the *n*the previous command line from history.

*n***G** Selects the command with history number *n*, or the latest command if *n* is omitted.

*/string***Enter**
Selects the first command line, searching backwards, that matches *string*. If *string* is omitted, the previous search string is used.

*?string***Enter**
Selects the first command line, searching forwards, that matches *string*. If *string* is omitted, the previous search string is used.

n Repeats the last string search ('**/**' or '**?**') command.

N Repeats the last string search, but in the opposite direction.

Text Change

The following commands alter the text in the current command line. Some of these commands operate on a text block, defined by an immediately following cursor movement command. This is designated by *m* (for *movement*) in the text change command. The text block extends from the current cursor position to the new position determined by the movement command.

i Enters input mode, inserting text before the character under the cursor.

I Inserts before the first nonblank on line (**i**).

a Moves the cursor forward one character and enter input mode, appending text after the character originally under the cursor.

A Appends to end of line (**$a**).

*n***d***m* Deletes text block. If *n* is given, it is applied to the *movement*.

dd Deletes entire command line.

D Deletes from cursor to end of line (**d$**).

*n***x** Deletes *n* characters to right of cursor (*n***dl**).

*n***X** Deletes *n* characters to left of cursor (*n***dh**).

*n***c***m* Change text block; deletes block of text and enters input mode. If *n* is given, it is applied to the *movement*.

cc Change entire command line.

s Change entire command line.

*n***s** Change next *n* characters from cursor.

*n***p** Puts back, after the character under the cursor, *n* copies of the last block deleted by a text change command.

*n***P** Puts back, before the character under the cursor, *n* copies of the last block deleted by a text change command.

r*c* Replaces the single character under the cursor with the character *c*, and advances the cursor one position.

R Enters *replace mode*: a special case of input mode in which each character entered overwrites that under the cursor, and advances the cursor one position.

u Undoes the last text change to the current line. This is itself a text change command, and so acts as a toggle for the last change.

U Undoes all changes to the current line.

n **~** Inverts the case of the next *n* characters, advancing the cursor over them.

*n***.** Repeats the last text change command. If *n* is given, it overrides the count originally given with the repeated command.

*n*_ Appends after the character under the cursor, the *n*the argument from the previous command line (default last), and enter input mode.

***** Replaces the current word with the list of file names that matches the word with an ***** appended. If there is no match, an audible alarm sounds and the word is not changed. Otherwise, the cursor is positioned at the end of the list and input mode is entered.

**** Used to complete a pathname. If there is only one existing pathname that matches as much as you've typed, the pathname is completed and a space is added after the complete pathname. If there are several matching pathnames, the shell expands what you've typed by adding all the characters that are common to all matching pathnames.

= Lists all pathnames matching the current word.

Miscellaneous

*n***y***m* Yanks text block into the delete buffer. Does not alter the command line or cursor position, but makes the text block available to subsequent **put** or **p** commands. If *n* is given, it is applied to the movement.

yy Yanks the entire command line.

Y Equivalent to **y$**. Yanks the rest of the line.

Equivalent to **I#Enter**.

*n***v** Executes **fc –e ${VISUAL:–${EDITOR:–vi}}** *n*. If *n* is omitted, the history number of the current line is used.

Ctrl-L Redisplays the current line.

Ctrl-J Executes the current line.

cm Executes the current line.

Enter Executes the current line.

@*letter* Inserts the value of the alias named **_letter**. The symbol *letter* represents a single alphabetic character from the portable character set; implementations may support additional characters as an extension. If the alias **_letter** contains other editing commands, these commands are performed as part of the insertion. If the **_letter** alias is not enabled, this command has no effect.

Limits

Selecting a previous history line for editing while at a secondary prompt (that is, while entering a subsequent line of a new multiline command) yields unexpected results.

Related Information

fc, **set**, **sh**, **vi**

---

shift — Shift positional parameters

Format

shift [*expression*]

tcsh shell: **shift** [*variable*]

Description

shift renames the positional parameters so that *i+n*th positional parameter becomes the *i*th positional parameter, where *n* is the value of the given arithmetic *expression*. If you omit *expression*, the default value is 1. The value of *expression* must be between zero and the number of positional parameters (**$#**), inclusive. The value of **$#** is updated.

shift in the tcsh shell

Without arguments, **shift** discards **argv[1]** and shifts the members of **argv** to the left. It is an error for **argv** not to be set or to have less than one word as value. With *variable*, **shift** performs the same function on *variable*. See “tcsh — Invoke a C shell” on page 556.

Examples

The commands:

```
set a b c d
shift 2
echo $*
```

produce:

```
c d
```

Usage Note

shift is a special built-in shell command.

Localization

shift uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

`0` Successful completion

`1` Failure because the *expression* had a negative value or was greater than the number of positional parameters.

Messages

Possible error messages include:

bad shift count *expr*
: You specified an expression that did not evaluate to a number in the range from `0` to the number of remaining positional parameters.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Allowing an expression, rather than just a number, is an extension found in the OS/390 UNIX System Services shell (a KornShell).

Related Information

set, **sh**, **tcsh**

---

sleep — Suspend execution of a process for an interval of time

Format

sleep *seconds*

Description

sleep continues running until the specified number of *seconds* has elapsed. **sleep** can delay execution of a program or produce periodic execution in conjunction with shell commands.

The *seconds* argument can be either a number of seconds, or a more general time description of the form *n*h*n*m*n*s, with the *n*h, *n*m, and the s being optional.

Example

```
sleep 20h10m
```

sleeps for 20 hours and 10 minutes (or 72600 seconds).

Localization

sleep uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion

2 Failure because you specified no *seconds* value or because *seconds* is an incorrect argument (for example, incorrect format).

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Information

date

---

sort — Start the sort-merge utility

Format

sort [**–cmu**] [**–o** *outfile*] [**–t** *char*] [**–y**[*n*]] [**–z***n*] [**–bdfiMnr**] [**–k** *startpos[,endpos]] ... [file ...]*
sort [**–cmu**] [**–o** *outfile*] [**–t***char*] [**–y***n*] [**–z***n*] [**–bdfiMnr**] [**+***startposition* [**–***endposition*]] ... [*file ...*]

Description

sort implements a full sort-and-merge utility. By default, it sorts according to all the information in the record, in the order given in the record.

sort operates on input files containing records that are separated by the newline character. When you do not specify either the **–c** or **–m** option, **sort** sorts the concatenation of all input files and produces the output on standard output. The following options select particular operations:

–c Checks input files to ensure that they are correctly ordered according to the key position and sort ordering options specified, but does not modify or output the files. This option affects only the exit code.

–m Merges *file*s into one sorted output stream. This option assumes that each input file is correctly ordered according to the other options specified on the command line; you can check this with the **–c** option.

–u Ensures that output records are unique. If two or more input records have equal sort keys, **sort** writes only the first record to the output. When you use **–u** with **–c**, **sort** prints a diagnostic message if the input records have any duplicates.

When you do not specify either the **–c** or the **–m** option, **sort** sorts the concatenation of all input files and produces the output on standard output.

Options

–o *outfile*
Writes output to the file *outfile*. By default, **sort** writes output to the standard output. The output file can be one of the input files. In this case, **sort** makes a copy of the data to allow the (potential) overwriting of the input file.

–t *char*
Indicates that the character *char* separates input fields. When you do not specify the **–t** option, **sort** assumes that any number of white-space (blank or tab) characters separate fields.

–y[*n***]**
Restricts the amount of memory available for sorting to *n* KB of memory (where a KB of memory is 1024 bytes). If *n* is missing, **sort** chooses a reasonable maximum amount of memory for sorting, dependent on the system configuration. **sort** needs at least enough memory to hold five records simultaneously. If you try to request less, **sort** automatically takes enough. When the input files overflow the amount of memory available, **sort** automatically does a polyphase merge (external sorting) algorithm, which is, of necessity, much slower than internal sorting. *n* must be at least 2. *n* has a maximum value of 1024 and a default value of 56.

When you use **–u** with **–c**, **sort** prints a diagnostic message if the input records have any duplicates. Using the **–y** option may therefore improve sorting performance substantially for medium to large input files.

–z*n* Indicates that the longest input record (including the newline character) is *n* bytes in length. By default, record length is limited to LINE_MAX.

The following options control the way in which **sort** does comparisons between records in order to determine the order in which the records are placed on the output. The ordering options apply globally to all sorting keys except those keys for

which you individually specify the ordering option. For more on sorting keys, see "Sorting Keys" on page 523.

–b Skips, for comparison purposes, any leading white space (blank or tab) in any field (or key specification).

–d Uses *dictionary* ordering. With this option, **sort** examines only blanks, uppercase and lowercase letters, and numbers when making comparisons.

–f Converts lowercase letters to uppercase for comparison purposes.

–i Ignores, for comparison purposes, nonprintable characters.

–k [*startpos* **[***endpos***]].**
Specifies a sorting key. For more information, see "Sorting Keys."

–M Assumes that the field contains a month name for comparison purposes. Any leading white space is ignored. If the field starts with the first three letters of a month name in uppercase or lowercase, the comparisons are in month-in-year order. Anything that is not a recognizable month name compares less than `JAN`.

–n Assumes that the field contains an initial numeric value. **sort** sorts first by numeric value and then by the remaining text in the field according to options.

Numeric fields can contain leading optional blanks or optional minus (–) signs. **sort** does not recognize the plus (+) sign.

This option treats a field which contains no digits as if it had a value of zero. If more than one line contains no digits, the lines are sorted alphanumerically.

–r Reverses the order of all comparisons so that **sort** writes output from largest to smallest rather than smallest to largest.

Sorting Keys

By default, **sort** examines entire input records to determine ordering. By specifying *sorting keys* on the command line, you can tell **sort** to restrict its attention to one or more parts of each record.

You can indicate the start of a sorting key with:

`-k` *`m`*`[.`*`n`*`][`*`options`*`]`

where *m* and the optional *n* are positive integers. You can choose *options* from the set **bdfiMnr** (described previously) to specify the way in which **sort** does comparisons for that sorting key. Ordering options set for a key override global ordering options. If you do not specify any *options* for the key, the global ordering options are used.

The number *m* specifies which field in the input record contains the start of the sorting key. The character given with the **–t** option separates input fields; if this option is not specified, spaces or tabs separate the fields. The resulting sort key is from the *m*th field to the end of the record. The number *n* specifies which character in the *m*th field marks the start of the sorting key; if you do not specify *n*, the sorting key starts at the first character of the *m*th field.

If an ending position for a key is not specified, the sorting key extends from the starting position to the end of the input record. You can also specify an ending position for a key, with:

-k *m*[.*n*][*options*],
p[.*q*][*options*]

where *p* and *q* are positive integers, indicating that the sort key ends with the with *q*th character of the *p*th field. If you do not specify *q* or if you specify a value of 0 for *q*, the sorting key ends at the last character of the *p*th field. For example:

```
-k 2.3,4.6
```

defines a sorting key that extends from the third character of the second field to the sixth character of the fourth field. The **b** option applies only the key start or key end for which it is specified;

```
-k 2
```

defines a sorting key that extends from the first character of the second field to the end of the record;

```
-k2 2
```

defines a sorting key that extends from the first character of the second field to the last character of the second field.

sort also supports a historical method of defining the sorting key. Using this method, you indicate the start of the sorting key with:

+*m*[.*n*][*options*]

which is equivalent to:

–k *m*+1[.*n*+1][*options*]

You can also indicate the end of a sorting key with:

–*p*[.*q*][*options*]

which when preceded with +*m*[.*n*] is equivalent to:

–k *m*+1[.*n*+1],*p*.0[*options*]

if q is specified and is zero. Otherwise,

–k *m*+1[.*n*+1],*p*+1[.*q*][*options*]

For example:

```
+1.2 -3.5
```

defines a sorting key with a starting position that **sort** finds by skipping the first two characters of the next field and an ending position that **sort** finds by skipping the first three fields and then the first five characters of the next field. In other words, the sorting key extends from the third character of the second field to the sixth character of the fourth field. This is the same key as defined under the **–k** option, described earlier.

With either syntax, if the end of a sorting key is not a valid position after the beginning key position, the sorting key extends to the end of the input record.

You can specify multiple sort key positions by using several **–k** options or several **+** and **–** options. In this case, **sort** uses the second sorting key only for records where the first sorting keys are equal, the third sorting key only when the first two are equal, and so on. If all key positions compare equal, **sort** determines ordering by using the entire record.

When you specify the **–u** option to determine the uniqueness of output records, **sort** looks only at the sorting keys, not the whole record. (Of course, if you specify no sorting keys, **sort** considers the whole record to be the sorting key.)

Examples

1. To sort an input file having lines consisting of the day of the month, white space, and the month, as in:

```
30 December
23    MAY
25 June
10     June
```

   use the command:

```
sort -k 2M -k 1n
```

2. To merge two dictionaries, with one word per line:

```
sort –m –dfi dict1 dict2 >newdict
```

Environment Variable

sort uses the following environment variable:

TMPDIR Contains the pathname of the directory to be used for temporary files.

File

sort uses the following file:

/tmp/stm*

Temporary files used for merging and **–o** option. You can specify a different directory for temporary files using the **TMPDIR** environment variable.

Localization

sort uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_NUMERIC**
- **LC_TIME**
- **NLSPATH**

The **–M** option works only if **LC_TIME** identifies a locale that contains the same month names as the POSIX locale.

See Appendix F for more information.

Exit Values

0 Successful completion. Also returned if **–c** is specified and the file is in correctly sorted order.

1 Returned if you specified **–c** and the file is not correctly sorted. Also returned to indicate a nonunique record if you specified **–cu**.

2 Failure due to any of the following:

- Missing key description after **–k**
- More than one **–o** option
- Missing *file* name after **–o**
- Missing character after **–t**
- More than one character after **–t**
- Missing *number* with **–y** or **–z**
- *endposition* given before a *startposition*
- Badly formed sort key
- Incorrect command-line option
- Too many key field positions specified
- Insufficient memory
- Inability to open the output file
- Inability to open the input file
- Error in writing to the output file
- Inability to create a temporary file or temporary filename

Messages

Possible error messages include:

Badly formed sort key position *x*
The key position was not specified correctly. Check the format and try again.

File *filename* **is binary**
sort has determined that *filename* is binary because it found a NULL (' ') character in a line.

Insufficient memory for ...
This error normally occurs when you specify very large numbers for **–y** or **–z** and there is not enough memory available for **sort** to satisfy the request.

Line too long: limit *nn* **— truncated**
Any input lines that are longer than the default number of bytes (LINE_MAX) or the number specified with the **–z** option are truncated.

Missing key definition after -k
You specified **–k**, but did not specify a key definition after the **–k**.

No newline at end of file
Any file not ending in a newline character has one added.

Nonunique key in record ...
With the **–c** and **–u** options, a nonunique record was found.

Not ordered properly at ...
With the **–c** option, an incorrect ordering was discovered.

Tempfile error on ...
The named temporary (intermediate) file could not be created. Make sure that you have a directory named **/tmp**, and that this directory has space to create

files. You can change the directory for temporary files using the **TMPDIR** environment variable.

Tempnam() error
sort could not generate a name for a temporary working file. This should almost never happen.

Temporary file error (no space) for ...
Insufficient space was available for a temporary file. Make sure that you have a directory named **/tmp**, and that this directory has space to create files. You can change the directory for temporary files using the **ROOTDIR** and **TMPDIR** environment variables.

Too many key field positions specified
This implementation of **sort** has a limit of 64 key field positions.

Write error (no space) on output
Some error occurred in writing the standard output. Barring write-protected media and the like, this normally occurs when there is insufficient disk space to hold all of the intermediate data.

Portability

POSIX.2, X/Open Portability Guide.

Available on all UNIX systems, with only UNIX System V.2 or later having the full utility described here.

The **–M**, **–y**, and **–z** options are extensions of the POSIX standard.

Related Information

awk, **comm**, **cut**, **join**, **uniq**

The **sortgen awk** script is a useful way to handle complex sorting tasks. It originally appeared in *The AWK Programming Language,* by Aho, Weinberger, and Kernighan. The POSIX.2 standard regards the historical syntax for defining sorting keys as obsolete. Therefore, you should use only the **–k** option in the future.

---

spell — Detect spelling errors in files

Format

spell [**–biluvx**] [**–d** *hashfile*] [**–f** *local*] [**–h** *history*] [**+***local*] [*file ...*]

Description

spell checks for misspelled words in each specified file. If you do not specify a file, it checks the standard input (**stdin**). A list of potentially misspelled words is produced on standard output (**stdout**).

Words are checked against a local word list and then against a hashed word list. The hashed word list included in this distribution contains virtually no proper names or technical terms. It is assumed that you will enter these words into your local word list (or into your machine's word list). Any capitalized word in the hash list must be capitalized in the input document; all other words are matched either capitalized or not. All word forms, including plurals, must be explicitly included in the

hash list. This approach prevents the acceptance of nonsense words that can result from the algorithmic combination of legal roots with legal suffixes or prefixes, a phenomenon common to many other spelling checkers.

Options

–b Uses British spelling (such as “colour” instead of “color”). The dictionary file used is **/usr/lib/hashb** instead of **/usr/lib/hash**.

–d *hashfile*
Uses *hashfile* as the dictionary. *hashfile* is a hash list produced from a list of words using the **-i** option of **spell**. To use a list other than the default **/usr/lib/hash**, the **–d** option must be specified.

–f local
Uses the file **local** as a dictionary of local words, given one word per line. If you do not specify this option, the file **/usr/lib/lwords** is used as the local dictionary.

–h *history*
Appends a history of all misspelled words to the file **history**. This file can be used by a system administrator for dictionary maintenance or generating a local dictionary.

–i Creates a new hash list file or add words to an existing file, instead of checking for spelling errors. Words to be entered into the dictionary should be specified one per line with no white space on the line. Lines beginning with the **#** character are ignored as comments. Be sure that the words you are entering into the hash list are correctly spelled.

–l Produces a longer form of output. For each misspelled word, **spell** prints three tab-separated columns containing the misspelled word, the line number, and the filename.

–u Forces **spell** to accept any word that is in all uppercase. **spell** assumes that such words are acronyms.

–v Writes to **stdout** all words not literally in the dictionary. This is the default for this implementation because it doesn't apply suffix/prefix rules to derive words.

–x Writes each plausible word stem to **stdout**. Because this implementation of **spell** doesn't derive words, all words are their own word stems.

+ *local*
Uses the file **local** as a dictionary of local words, given one word per line. This is synonymous with **–f**.

Examples

By default, **spell** does not sort the output. This maintains the order and number of occurrences of spelling errors. The following command checks for spelling errors, puts them in dictionary order, removes duplicates, and print them in a multicolumn format:

```
spell file | sort –dfu | c
```

Localization

spell uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion

1 Failure due to any of the following:
- Missing *hashfile* name after **–d**
- Missing *history* filename after **–h**
- Missing **local** filename after **–f**
- Inability to open the **local** file
- Receipt of user interrupt
- An error reading the dictionary file

A spelling mistake is not considered an error.

2 Incorrect command-line option

Files

spell uses the following files:

/usr/lib/hash
The default dictionary file, in hashed form.

/usr/lib/hashb
The British dictionary file, in hashed form.

/usr/lib/lwords
The default location of the local words file. This need not exist.

Limits

Input lines in the text being checked are restricted to a maximum of 100 characters.

Portability

X/Open Portability Guide, UNIX systems.

The **–d**, **–f**, **–h**, **–i**, **–l**, and **–u** options are extensions of the POSIX standard.

Related Information

sort, **vi**

split — Split a file into manageable pieces

Format

split [**–a** *n*] [**–l** *n*] [*file* [*prefix*]]
split –b *n*[**bkm**] [**–a** *n*] [*file* [*prefix*]]
split [**–***n*] [**–a** *n*] *file* [*prefix*]

Description

split breaks a file up into a set of files. It starts a new file every time it has copied 1000 lines.

split names the files that it creates as a prefix followed by a suffix. `x` is the prefix unless you specify a different *prefix* on the command line. Unless altered by the following options, the suffix begins as `aa` and is incremented with each new file. By default, therefore, the first file is `xaa` followed by `xab`, and so on.

Options

–a *n*
Uses a suffix *n* letters long. The default is two.

–b *n***[bkm]**
Splits the file every *n* units. The default unit size is bytes. When you follow *n* with **b**, **k**, or **m**, **split** uses a corresponding unit size of 512 bytes, 1K (1024 bytes), or 1 megabyte (1 048 576 bytes).

–l *n* Splits the file every *n* lines.

–*n* Is an obsolescent version of the **–l** option.

If the *file* is – (dash) or if no file is specified, **split** reads the standard input (**stdin**).

Localization

split uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

`0` Successful completion
`1` Failure due to any of the following:
- Error opening input or output file
- Missing number after **–a**
- Invalid **–a** option
- Missing byte count after **–b**
- Invalid byte count specification
- Invalid count specification
- Unknown option

- Out of memory for binary split buffer
- Read error on input file
- Write error on output file
- Too many names generated

Portability

POSIX.2 User Portability Extension, X/Open Portability Guide, UNIX systems,

The **b** suffix of the **–b** option is an extension to the POSIX.2 standard.

Related Information

csplit

---

stop — Suspend a process or job

Format

stop [*pid ...*] [*job—identifier ...*]

tcsh shell: **stop** *%job|pid ...*

Description

stop is an alias for **kill –STOP**. Like **kill –STOP**, **stop** sends a **SIGSTOP** to the process you specify.

See “kill — End a process or job, or send it a signal” on page 301 for more information.

stop in the tcsh shell

In the tcsh shell, **stop** stops the specified jobs or processes which are executing in the background. *job* may be a number, a string, '', %, + or - . There is no default *job*. Specifying stop alone does not stop the current job. See “tcsh — Invoke a C shell” on page 556.

Options

job-identifier

Is the job identifier reported by the shell when a process is started with **&**. It is one way to identify a process. It is also reported by the **jobs** command. When using the job identifier with the **stop** command, the job identifier must be prefaced with a percent (%) sign. For example, if the job identifier is 2, the **stop** command would be entered as follows:

```
stop %2
```

pid Is the process ID that the shell reports when a process is started with **&**. You can also find it using the **ps** command. The *pid* argument is a number that may be specified as octal, decimal, or hex. Process IDs are reported in decimal. **stop** supports negative values for *pid*.

If *pid* is negative but not –1, the signal is sent to all processes whose process group ID is equal to the absolute value of *pid*. The negative *pid* is specified in this way:

```
stop — –nn
```

where *nn* is the process group ID and may have a range of 2 to 7 digits (*nn* to *nnnnnnn*).

```
stop — –9812753
```

The format must include the — before the *–nn* in order to specify the process group ID.

If *pid* is `0`, the signal is sent to all processes in the process group of the invoker.

The process to be killed must belong to the current user, unless he or she is the superuser.

Related Information

kill, **jobs**, **sh**, **suspend**, **tcsh**

---

strings — Display printable strings in binary files

Format

strings [**–aopxz**] [**–n** *number*] [**–t** *format*] [*file ...*]
strings [–] [**–opxz**] [**–t** *format*] [*–number*] [*file ...*]

Description

If the command line specifies a filename of –, **strings** reads the standard input. **strings** finds pieces of information in binary files. It is frequently used for looking through executable files to uncover items such as copyright notices, error messages, and undocumented feature.

The command displays strings of printable characters that are at least four characters in length. Strings must be terminated by a NUL character or by a newline.

Options

–a This option has no effect in the OS/390 environment. The entire file is examined, regardless of whether or not this option is specified.

–n *number*
Displays strings of printable characters that are at least *number* characters in length. If you do not specify the **–n** option, strings will act as if **–n 4** had been specified.

–o For each string, displays as an octal value its offset in bytes from the beginning of the file. This is the same as **–t** *o*.

–t *format*
For each string, displays its offset in bytes from the beginning of the file. The base of the offset is set to decimal, octal, or hexadecimal by specifying *format* as *d*, *o*, or *x*, respectively.

–x For each string, displays as a decimal value its offset in bytes from the beginning of the file. This is the same as **–t** *x*.

–z Ignores the POSIX definition of a string and searches for any group of printable characters greater than four in length.

– Is the obsolescent version of **–a**.

–*number*
Is the obsolescent version of **–n** *number*.

Localization

strings uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion
1 Failure due to any of the following:
- Incorrect command-line option
- Insufficient memory

Portability

POSIX.2 User Portability Extension, X/Open Portability Guide.

A Berkeley addition to most UNIX systems. Most Berkeley versions do not require the terminating NUL or newline.

The **–o** and **–x** options are extensions to the POSIX standard.

---

strip — Remove unnecessary information from an executable file

Format

strip *file*

Description

strip removes symbolic information (and any other information not required for execution) from an executable file in order to conserve disk space for production (that is, already debugged) programs. **strip** does not modify the contents of any executable file—it is functionally equivalent to **touch** *file*.

Localization

strip uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion

1 Failure due to any of the following:

- *file* does not exist or could not be opened.
- The user does not have write permission for *file*
- An error occurred while reading *file*
- *file* is not an executable file
- *file* is executable, but appears corrupted

2 No *file* was specified on the command line

Messages

Possible error messages include:

executable file *file***: No such file or directory**
The input file does not exist. Check that the filename was entered correctly and that it exists.

file *file1***: Not an executable file**
strip only operates on executable files.

Write permission required to strip *file*
The user does not have write permission on the file.

executable file *file***: Permission denied**
The user does not have read permission on the file.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

---

stty — Set or display terminal options

Format

stty [**–ag**] [*operand*]

Description

stty sets or reports the terminal I/O characteristics for the standard input device. **stty**, entered without options or operands, reports only the terminal I/O characteristics that differ from the defaults. **stty**, entered with *operand*s enables, disables, or selects the full range of terminal I/O characteristics.

The **stty** command affects whichever line discipline is in effect for your terminal. If you are using the OS/390 Communications Server, then **stty** affects the line discipline on the RISC System/6000 where the OS/390 Communications Server is running. If you are not using OS/390 Communications Server, then **stty** affects the line discipline for OS/390.

Options

This command supports the following options:

–a Displays all of the terminal I/O characteristics.

–g Displays all of the terminal I/O characteristics in a format that can be used as input to the **stty** command.

The **–a** option gives you a clear readable description, whereas the **–g** option enables you to save and restore the terminal I/O characteristics.

stty entered with *operand*s enables, disables, or selects the full range of terminal I/O characteristics.

Control Mode Operands

The valid operands for setting control modes are:

parenb Enable parity generation and detection. Not valid for OS/390 line discipline. **–parenb** is always used. See "Usage Notes" on page 539.

–parenb Disable parity generation and detection.

parodd Select odd parity. Not valid for OS/390 line discipline. **–parodd** is always used. See "Usage Notes" on page 539.

–parodd Select even parity.

cs5 Select character size CS5. Not valid for OS/390 line discipline. CS8 is always used. See "Usage Notes" on page 539.

cs6 Select character size CS6. Not valid for OS/390 line discipline. CS8 is always used. See "Usage Notes" on page 539.

cs7 Select character size CS7. Not valid for OS/390 line discipline. CS8 is always used. See "Usage Notes" on page 539.

cs8 Select character size CS8.

number Set the input and output baud rates to *number*. A *number* of zero hangs up the modem line.

ispeed *number*
Set the input baud rate to *number*. Not valid for OS/390 line discipline. No special processing of zero is done. See "Usage Notes" on page 539.

ospeed *number*
Set the output baud rate to *number*. Not valid for OS/390 line discipline. No special processing of zero is done. See "Usage Notes" on page 539.

hupcl Hang up the modem line on the last close.

–hupcl Do not hang up the modem line on the last close.

hup Hang up the modem line on the last close.

–hup Do not hang up the modem line on the last close.

cstopb Use two stop bits per character. Not valid for OS/390 line discipline. **–cstopb** is always used. See "Usage Notes" on page 539.

–cstopb Use one stop bit per character.

cread Enable the receiver.

–cread Disable the receiver. Not valid for OS/390 line discipline. **cread** is always used. See “Usage Notes” on page 539.

clocal Assume a line without modem control.

–clocal Assume a line with modem control.

columns number
Set number of columns to number.

This should only be used if rlogin or telnet client does not support window size, or you are having trouble getting the correct size.

row number
Set number of rows to number.

This should only be used if rlogin or telnet client does not support window size, or you are having trouble getting the correct screen size.

Input Mode Operands

The valid operands for setting input modes are:

ignbrk Ignore break on input.

–ignbrk Do not ignore break on input.

brkint Signal INTR on break.

–brkint Do not signal INTR on break.

ignpar Ignore parity errors.

–ignpar Do not ignore parity errors.

parmrk Mark parity errors.

–parmrk Do not mark parity errors.

inpck Enable input parity checking.

–inpck Disable input parity checking.

istrip Strip input characters to seven bits. This feature is required by the standards but IBM strongly recomments that you not use the setting for non-OCS sessions. It will make it impossible to send EBCDIC alphanumeric characters to your shell session and you will have to take extreme measures to terminate the session.

–istrip Do not strip input characters to seven bits. This is the default and should not be changed.

inlcr Map newline to carriage return on input.

–inlcr Do not map newline to carriage return on input.

igncr Ignore carriage return on input.

–igncr Do not ignore carriage return on input.

icrnl Map carriage return to newline on input.

–icrnl Do not map carriage return to newline on input.

iuclc Map uppercase alphabetic characters to lowercase on input.

–iuclc Do not map uppercase alphabetic characters to lowercase on input.

ixon Enable START/STOP output control.

–ixon Disable START/STOP output control.

ixany Allow any character to restart input.

–ixany Do not allow any character to restart input.

ixoff Ask the system to send START/STOP characters to regulate the size of the input queue.

–ixoff Ask the system not to send START/STOP characters to regulate the size of the input queue.

Output Mode Operands

The valid operands for setting output modes are:

onlcr Converts newline characters to newline-carriage return sequences.

–onlcr Newline characters are displayed as newlines only.

opost Postprocess output.

–opost Do not postprocess output. Ignore all other output modes.

olcuc Map lowercase alphabetic characters to uppercase on output.

–olcuc Do not map lowercase alphabetic characters to uppercase on output.

ocrnl Map CR to NL on output.

–ocrnl Do not map CR to NL on output.

onocr Do not output CR at column 0.

–onocr Output CR at column 0.

onlret The terminal newline key performs the CR function.

–onlret The terminal newline key does not perform the CR function.

ofill Use fill characters for delays.

–ofill Use timing for delays.

ofdel Fill characters are DELs.

–ofdel Fill characters are NULs.

cr0 Sets the style of delay for CRs (CRDLY) to CR0.

cr1 Sets the style of delay for CRs (CRDLY) to CR1.

cr2 Sets the style of delay for CRs (CRDLY) to CR2.

cr3 Sets the style of delay for CRs (CRDLY) to CR3.

nl0 Select the style of delay for NL (NDLY) to NL0.

nl1 Select the style of delay for NL (NLDLY) to NL1.

tab0 Sets the style of delay for horizontal tabs (TABDLY) to TAB0.

tab1 Sets the style of delay for horizontal tabs (TABDLY) to TAB1.

tab2 Sets the style of delay for horizontal tabs (TABDLY) to TAB2.

tab3 Sets the style of delay for horizontal tabs (TABDLY) to TAB3.

bs0 Select the style of delay for backspace (BSDLY) to BS0.

bs1 Select the style of delay for backspace (BSDLY) to BS1.

ff0 Select the style of delay for form feeds (FFDLY) to FF0.

ff1 Select the style of delay for form feeds (FFDLY) to FF1.

vt0 Select the style of delay for vertical tabs (VTDLY) to VT0.

vt1 Select the style of delay for vertical tabs (VTDLY) to VT1.

Local Mode Operands

The valid operands for setting local modes are:

isig Enable character checking against the special control characters INTR, QUIT and SUSP.

–isig Disable character checking against the special control characters INTR, QUIT and SUSP.

icanon Enable canonical input mode.

–icanon Disable canonical input mode.

xcase Set canonical uppercase or lowercase presentation.

–xcase Do not set canonical uppercase or lowercase presentation.

iexten Enable any custom special control characters.

–iexten Disable any custom special control characters.

echo Echo every character typed.

–echo Do not echo every character typed.

echoe Enable the ERASE character to visibly erase the latest character.

–echoe Do not enable the ERASE character to visibly erase the latest character.

echok Echo newline after a KILL character.

–echok Do not echo newline after a KILL character.

echonl Echo newline (even when **echo** is disabled).

–echonl Do not echo newline when **echo** is disabled.

noflsh Disable flush after INTR, QUIT, and SUSP.

–noflsh Enable flush after INTR, QUIT, and SUSP.

tostop Send the **SIGTOU** signal for background output.

–tostop Do not send the **SIGTOU** signal for background output.

Control Character Operands

In a doublebyte environment, the *char* parameter to these operands must be a narrow (singlebyte) character.

The valid operands for assigning special control characters are:

min *number* Set `min` to *number*.
time *number* Set `time` to *number*.
eof *char* Set end of file character to *char*.
eol *char* Set end of line character to *char*.
erase *char* Set ERASE character to *char*.
intr *char* Set INTR character to *char*.
kill *char* Set KILL character to *char*.

quit *char* Set QUIT character to *char*.
susp *char* Set SUSP character to *char*.
start *char* Set START character to *char*.
stop *char* Set STOP character to *char*.

Combination Mode Operands

The valid operands for setting combination modes are:

saved-settings
Set the terminal I/O characteristics to the saved settings produced by the **–g** option.

evenp Enable **parenb** and `cs7`; disable **parodd**. Not valid for OS/390 line discipline. See "Usage Notes."

parity Enable **parenb** and `cs7`; disable **parodd**. Not valid for OS/390 line discipline. See "Usage Notes."

oddp Enable **parenb**, `cs7` and **parodd**. Not valid for OS/390 line discipline. See "Usage Notes."

–parity Disable **parenb** and set **cs8**. Not valid for OS/390 line discipline. See "Usage Notes."

–evenp Disable **parenb** and set **cs8**. Not valid for OS/390 line discipline. See "Usage Notes."

–oddp Disable **parenb** and set **cs8**. Not valid for OS/390 line discipline. See "Usage Notes."

raw Enable raw input and output.

–raw or cooked
Disable raw input and output.

nl Enable **icrnl**.

–nl Disable **icrnl**; unset **inlcr** and **igncr**.

lcase Set **xcase**, **iuclc**, and **olcuc**.

–lcase Disable **xcase**, **iuclc**, and **olcuc**.

LCASE Equivalent to **lcase**.

–LCASE Equivalent to **–lcase**.

tabs Perserve tabs when printing.

–tabs or tab8
Expand to spaces when printing.

ek Reset ERASE and KILL characters to system defaults.

sane Reset all modes to reasonable values.

Usage Notes

1. **stty** will operate successfully even if it is unable to perform one or more actions in a group of requested actions. For example, if a valid OS/390 operand is requested with an invalid one, **stty** will operate successfully because it can perform the valid operand. The valid operand will then be satisfied.
2. If **stty** is only used with invalid OS/390 operands or invalid operands in combination with valid operands that have already been satisfied, **stty** will fail.

Localization

stty uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion
1 Failure due to any of the following:
- Error setting termios attributes
- Unknown mode
- Missing number after option
- Argument out of range
- Bad number after option
- Internal error
- Error reading termios attributes
- Missing character after option
- Badly formed argument option character
- Missing speed after **ispeed** or **ospeed**
- Bad speed argument

Portability

POSIX.2, UNIX System V.

---

su — Change the user ID associated with a session

Format

su [**–s**][*userid*]

Description

su starts a new shell and lets you operate in it with the privileges of a superuser or another user.

If you do not specify a user ID, **su** changes your authorization to that of the superuser. If you specify a user ID, **su** changes your authorization to that of the specified user ID. The new environment is built and then a new session is initiated. The new session is run as a child shell of the shell issuing the **su** command.

su performs these functions:

- **Obtains your user profile information.** After validating that you have an OMVS segment in the user profile, the OMVS segment information is obtained.
- **Verifies authorization.** If a user ID is not specified, you must have the appropriate authorization to obtain superuser authority. You must be permitted to the

BPX.SUPERUSER RACF facility class. For more information, see *OS/390 UNIX System Services Planning*.

If a user ID is specified, and you do not have read access to the SURROGAT facility class profile, BPX.SRV.uuuuuuuu (where uuuuuuuu is the MVS userid associated with the target UID), you must enter the target user's password when prompted. If a user ID is specified, and you have read access to the SURROGAT facility class profile for the target user, you can use the **–s** option, or hit enter at the password. For more information see *OS/390 UNIX System Services Planning*.

If a user ID is specified, the group ID is changed to that of the specified user's default group GID.

If a user ID is specified, the supplementary group list is changed to that of the specified user.

- **Changes the user ID.** Your user ID may be changed to either the specified user ID or the superuser's user ID (UID 0).
  - When a user ID is specified, your MVS identity changes to the specified user ID, changing your access authority for MVS data sets in addition to changing to the new user's UID.
  - When a user ID is not specified, your MVS identity remains the same. This maintains your access authority to MVS data sets, while gaining superuser authority.
  - If you are already running under UID 0 and BPX.DAEMON is defined, issuing **su** with no commands will result in your uid being switched to BPX.ROOT. If BPX.DAEMON is not defined, and you issue **su** with no arguments while running under UID 0, your UID will remain set to 0. In both cases, access to the BPX.SUPERUSER facility class will not be checked.
- **Sets up the shell environment.** The OMVS segment of your user profile is used to set up the shell environment. The environment is set up to be as similar as possible to the environment of the shell issuing the **su** command. Information is obtained from the OMVS segment if available. If not, defaults are used. If the value for the initial program (shell) is not available, a default value of **/bin/sh** is used.
- **Executes the new shell.** The new shell is initialized to run as a child process of the shell issuing the **su** command. If the **su** command is run from a restricted shell (such as a shell that was started with the **–r** option), you will exit from the restricted shell and leave the protection of the trusted environment.

  Notes:

  1. The new shell is always run in a new address space, even if you have _BPX_SHAREAS=YES set.
  2. If you use the OMVS interface when running a shell created by **su**, any attempt to execute TSO commands (PF6) results in the command running back in your TSO address space. When these TSO commands run, they run with your TSO identity, not the identity specified by **su**.

     However, if you are not using the OMVS interface (e.g. you rlogin or telnet into the shell), you cannot use PF6 to execute a TSO command, and, as a result, there will be no TSO address space or identity. The alternative solution is to use **tso –t** which allows you to run a TSO/E command with the current identity set by **su**.

To restore the previous session, enter **exit** or press <EscChar-D> (where EscChar is normally the cent sign). If you use rlogin or telnet to enter the shell, you hold down the Ctrl key while you press D. This action ends the child shell initiated by the **su** command and returns you to the previous shell, user ID, and environment. See *OS/390 UNIX System Services User's Guide* for more information about exiting the shell environment.

Options

–s
Does not prompt for password. Use this option if you have read access to the SURROGATE facility class profile, BPX.SRV.uuuuuuuu (where uuuuuuuu is the MVS userid associated with the target UID).

Usage Notes

1. The new shell inherits the standard file descriptors from the **su** command, so commands can be piped to the **stdin** of the new shell and run under the new user.
2. If the OMVS NOECHO option is in effect, your password will be displayed.
3. Because **su** starts a new interactive shell, it should not be used from a batch interface such as BPXBATCH, unless you provide the commands to be executed under superuser via **stdin** to the **su** command.

Exit Values

0 The command completed successfully
1 The user is not authorized to obtain superuser authority
2 Failure due to any of the following:
- Unable to execute the shell
- The OMVS segment of the user's profile cannot be found
- Unable to set up the superuser environment

3 Failure due to any of the following:
- Incorrect command syntax
- Unable to open the message catalog

Messages

Possible error messages include:

User not authorized to obtain superuser authority
The user ID issuing the **su** command does not have the proper authorization to switch to superuser. Verify authorization with the system programmer.

Unable to set up the user environment. Processing terminates.
The environment variables required by the shell have not been set up. Processing terminates. Contact the system programmer.

Unable to open the message catalog.
The message catalog cannot be opened. Processing continues with the default messages being used. Verify that the message catalog exists in the file system. Contact the system programmer.

Unable to execute the shell.
The initial program (shell) was not run. Verify that the initial program (shell) exists on this system and that the user has permission to execute it.

The RACF profile for this user does not contain an OMVS segment.
During **su** command processing, the OMVS segment of the user's TSO/E profile could not be obtained. Contact the system programmer.

Command loaded from an unauthorized library. Processing terminates.
The **su** command must reside in an authorized library in order to check the password of a specified user ID. Contact the system programmer.

User not authorized to switch to -"%s-".-n"
The setuid to the specified user's UID failed because the user is not authorized. Contact the system programmer.

Unable to switch to -"%s-", due to an error in the OMVS segment of the RACF profile.-n"
The setuid to the specified user's UID failed. Processing terminates. Contact the system administrator.

User is not a surrogate of –"%s–".–n"
User not defined to have appropriate permission to the SURROGAT facility class profile for the new user. Either no password was entered or the **–s** option was used. Processing terminates. Contact the system administrator.

Limits

Only users who have RACF access permission to the superuser class can use **su** without specifying the user ID.

Portability

None. This command is an extension that comes with OS/390 UNIX services.

Related Information

sh, **ISHELL**

---

sum — Compute checksum and block count for file

Format

sum [**–ciprt**] [*file...*]

Description

sum calculates a checksum for each input file. It also displays the number of 512-byte blocks in each file. If you do not specify any files, or if you specify – as the filename, **sum** reads standard input (**stdin**). The checksum is useful as a quick way to compare a file or files that have been moved from one system to another to ensure that data has not been lost.

sum differs from **cksum** only in the format of the output. The output of **sum** has the form:

```
checksum    blockcount    filename
```

where *blockcount* is the number of 512-byte blocks in the file.

sum can calculate checksums in a variety of ways. The default checksum algorithm produces a 16-bit unsigned integer resulting from the arithmetic addition of each input byte. This checksum algorithm is not sensitive to byte order.

Options

–c Uses a standard 16-bit Cyclical Redundancy Check (CRC-16).

–i Uses the CCITT standard Cyclic Redundancy Check (CRC-CCITT). Data communications network protocols often use a cyclic redundancy check to ensure proper transmission. This algorithm is more likely to produce a different sum for inputs which differ only in byte order.

–p Uses the POSIX.2 checksum algorithm.

–r Enables the use of an alternate checksum algorithm which has the advantage of being sensitive to byte order.

–t Produces a line containing the total number of blocks of data read, as well as the checksum of the concatenation of the input files.

Localization

sum uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion

1 Failure due to any of the following:

- Inability to open input file
- Error reading the input file

2 Unknown command-line option

Portability

The default checksum algorithm is compatible with UNIX System V.2 and later. The **–r** algorithm is also available on UNIX System V.2 and is the default algorithm for Berkeley and Version 7. The **–c**, **–i**, and **–t** options are not available under UNIX.

Related Information

cmp, **cksum**, **diff**, **ls**, **wc**

---

suspend — Send a SIGSTOP to the current shell

Format

suspend

tcsh shell: same as above

Description

suspend is an alias for **stop $$**, where **stop** is an alias of **kill –STOP** and **$$** expands to the current process of the shell. **suspend** sends a **SIGSTOP** to the current shell.

See “kill — End a process or job, or send it a signal” on page 301 for more information.

suspend in the tcsh shell

suspend causes the tcsh shell to stop in its tracks, much as if it had been sent a stop signal with ^Z. See “tcsh — Invoke a C shell” on page 556.

Related Information

kill, **sh**, **tcsh**

---

sysvar — Display static system symbols

Format

sysvar *var*

Description

The **sysvar** command will allow users to obtain substitution text for system variables that may be defined in IEASYMxx or in the system IPL parameters. The substitution text will be printed to standard out. This could be used to substitute system variables in shell variables. For example:

```
system_name=$(sysvar SYSNAME)
```

Exit Values

0	Successful completion
1	Failure because *var* is not a valid system variable
2	Failure because no *var* was specified

---

tabs — Set tab stops

Format

tabs [**+m**[*margin*]] [**–T** *term*] [**–** *number*]
tabs [**+m**[*margin*]] [**–T** *term*] **–t** *tablist*
tabs [**+m**[*margin*]] [**–T** *term*] *num1*[,*num2,...*]
tabs [**+m**[*margin*]] [**–T** *term*] *tabspec*

Description

tabs sends a series of characters to the standard output, designed to clear the terminal hardware's tab stops and then set new ones. The characters that are sent depend on the type of terminal you are using.

The first column of your terminal screen is column 1. If you set a tab stop at position *N* and then tab to that position, the next character displayed on the screen appears in column *N+1* of the line (that is, after the tab stop).

tabs may not be able to set the tab stops on some types of terminals. In this situation, it issues an error message and then exits with a status greater than zero. **tabs** with no arguments sets tab stops every 8 positions.

Options

+m[*margin***]**
Sets the left margin to *margin*. It defaults to 10 if you do not specify a value. All tab positions are relative to the left margin. To find the actual tab positions, you add the value of *margin* to each tab position.

–T *type*
Indicates the type of terminal you have. The *term* argument is a site-specific name for your terminal type.

If you do not specify **–T**, **tabs** looks for an environment variable named **TERM** and uses its value for *type*. If **TERM** is not defined, **tabs** assumes a default terminal type.

–t *tablist*
Sets tab stops as specified by *tablist*. *tablist* consists of one or more positive decimal integers, separated by commas; the numbers in the list should be in strictly increasing order.

If only one number *N* is given, tabs are set every *N* columns. If more than one number is given, tabs are set at those column numbers.

*num1***[,***num2,...***]**
Sets tab stops to the given numbers. The numbers in the list should be positive decimal integers in strictly increasing order. Except for the first number, any number in the list may be preceded by a plus sign (+), in which case the number is considered to be an increment on the previous setting rather than a column position. For example,

```
tabs 4,8,12
tabs 4,+4,+4
```

are equivalent.

tabspec
Can be one of **–a**, **–a2**, **–c**, **–c2**, **–c3**, **–f**, **–p**, **–s** or **–u** and sets tab stops at these positions:

–a 1,10,16,36,72
–a2 1,10,16,40,72
–c 1,8,12,16,20,55
–c2 1,6,10,14,49
–c3 1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67
–f 1,7,11,15,19,23
–p 1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61

–s 1,10,55
–u 1,12,20,44

Each *tabspec* is designed for a particular programming language. Assembler uses **–a**,**–a2**, and **–u**. COBOL uses **–c**, **–c2**, and **–c3**. FORTRAN, PL/I, and SNOBOL use **–f**, **–p**, and **–s**, respectively.

–number
Sets tab stops every *number* positions along the line. *number* must be a single-digit decimal number. If *number* is zero (**–0**), **tabs** clears all the tab stops and does not set new ones.

Environment Variables

tabs uses the following environment variables:

TERM Contains the name of your terminal.
TERMINFO
Contains the pathname of the terminfo database.

Localization

tabs uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion
1 Missing definition in the terminfo database
2 Usage error
3 Unknown terminal or cannot find the terminfo database
4 Illegal tabs
5 An error occurred

Portability

POSIX.2 User Portability Extension, X/Open Portability Guide, UNIX systems.

The **+m**, **–t**, and *tabspec* arguments are all extensions to the POSIX standard.

The **–t** argument is an extension to the X/Open standard.

Related Information

stty

tail — Display the last part of a file

Format

tail [**–f**] [**–bcklmn** [±]*number*] [*file*]
tail [**–f**] [±*number* [**bcklmn**]] [*file*]

Description

tail without options displays the last ten lines of *file*. This is useful for seeing the most recent entries in log files and any file where new information is added on the end.

The **tail** command is used with text files. To make a binary file input to the **tail** command, use the **–c** option. If a binary file is input without the **–c** option being specified, the entire file is sent to the screen.

Options

±*number*
Is either of the following:

+*number*
Skips to line *number* and then displays the rest of the file. For example, `+100` prints from line 100 to the end of the file.

–*number*
Prints *number* lines from the end of the file. For example, `-20` prints the last 20 lines in the file.

You can precede or follow both **+***number* and **–***number* with one of the following letters to indicate the unit to be used:

- **b**–blocks
- **c**–bytes
- **k**–kilobytes
- **l** or **n**–lines
- **m**–megabytes

The default unit is lines.

–f Monitors a file as it grows. Every two seconds, **tail** wakes up and prints any new data at the end of the file. This option is ignored if **tail** read from the standard input, and standard input is a pipe.

Localization

tail uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion
1 Failure due to any of the following:
- Insufficient memory
- Write error on the standard output (**stdout**)
- Badly formed line or character count
- Missing number after an option
- Error reopening a file descriptor

2 Failure due to an unknown command-line option

Messages

Possible error messages include:

Badly formed line/character count *string*
In an option of the form **–n** *number* or **–***number*, the *number* was not a valid number.

Reopening file descriptor *number*
–f was used to follow a file as it grew. **tail** closed the file associated with the given file descriptor *number* and then tried to open it 2 seconds later. At this point, **tail** found it could not reopen the file for reading, and therefore could not follow the file any longer.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The POSIX standard does not include the use of **b**, **k**, or **m** as either options or suffixes. **–l** is an extension of the traditional implementation of **tail**.

Related Information

cat, **head**, **more**

---

talk — Talk to another user

Format

talk *address* [*terminal*]

Description

talk lets you begin a two-way conversation with someone else logged in to the system.

Options

address
Indicates the user with whom you want to talk. The most common form of *address* is the person's user name (as given by the **who** command), but other formats may be supported.

terminal
An optional identifier for use when the other user is logged in on more than one terminal. The format of the *terminal* identifier is the same as given by **who**.

Environment Variables

talk uses the following environment variables:

TERM	Contains the name of your terminal.
TERMINFO	Contains the pathname of the terminfo database.

Localization

talk uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Usage Notes

1. When you issue a **talk** command to request a conversation with another user, the other user receives a message of the form:

   ```
   Message from name
   talk: connection requested by your_address
   talk: respond with: talk your_address
   ```

   To set up the connection, your intended recipient must issue the system command

   ```
   talk your_address
   ```

   which establishes the two-way connection. After this connection has been established, both of you can type simultaneously. **talk** displays incoming messages from the other person in one part of the screen and outgoing messages in another part of the screen.

   Some terminals may not be able to split the screen into parts in this way. Depending on the terminal type, **talk** may try to simulate this effect. However, it may not be possible for both users to enter messages simultaneously. **talk** determines terminal type by looking for an environment variable named **TERM**. If this variable exists, **talk** uses its value as a site-specific name giving a terminal type. If **TERM** doesn't exist, **talk** assumes a default type.

2. The character-erase and line-kill characters work as usual. Typing <Ctrl-L> refreshes both parts of the screen (for example, if some unusual character messes up the display).

3. The interrupt character (for example, <Ctrl-C>) terminates your **talk** session and breaks the connection. When one side breaks the connection, **talk** notifies the other side and exits.

4. The **mesg** command lets you refuse **talk** sessions. With:

   ```
   mesg n
   ```

you can tell the system that you don't want to be interrupted by **talk** requests. If people try to establish a **talk** session with you, they are denied immediately; you are not informed about such requests. For more details, see **mesg**.

Exit Values

The following exit status values are possible:

`0` Successfully established and completed a transmission

`>0`
An error occurred, or you are trying to use **talk** on a terminal that cannot handle the way **talk** uses the screen

Portability

POSIX.2 User Portability Extension, X/Open Portability Guide, UNIX systems.

Related Information

mail, **mesg**, **who**, **write**

---

tar — Manipulate the tar archive files to copy or back up a file

Format

tar –c[*#s*]**bvwlzOUX**] **–f** *file* [**–V** *volpat*] [*tarfile*] [*blocksize*] [**–C** *pathname*]
tar –r[*#s*]**bvwlzOUX**] **–f** *file* [**–V** *volpat*] [*tarfile*] [*blocksize*]
tar –t[*#s*]**bvzEOUX**] **–f** *file* [**–V** *volpat*] [*tarfile*] [*blocksize*] [**–C** *pathname*]
tar –x[*#s*]**bvwpmozOUX**] **–f** *file* [**–V** *volpat*] [*tarfile*] [*blocksize*]

Description

tar reads, writes and lists archive files. An *archive file* is a single file containing one or more files and/or directories. Archive files can be HFS files or MVS datasets. A file stored inside an archive is called a *component file*; similarly, a directory stored inside an archive is called a *component directory*.

Note: MVS datasets cannot be specified for component files.

Included with each component file and directory is recorded information such as owner and group name, permission bits, file attributes, and modification time.

You can therefore use a single archive file to transfer a directory structure from one machine to another, or to back up or restore groups of files and directories.

Archives created by **tar** are interchangeable with those created with the **pax** utility. Both utilities can read and create archives in the default format of the other (USTAR for **pax** and TAR for **tar**). To save and restore external attributes, the USTAR format (**-U**) must be used. In general, the USTAR format records the most information and is recommended. Archives are generally named with suffixes such as .pax or .tar (or pax.Z and tar.Z for compressed files), but this is not required.

Note: In order to preserve information such as extended attributes, external links, and links whose targets exceed 100 characters, the USTAR format must be used. See the **-U** option for selecting the USTAR format. The **-O** and **-X**

options and “OS/390 Extended USTAR Support” on page 441 contain information about enabling and disabling USTAR support.

You cannot use **tar** unless you specify **–f**

Options

The four forms of the command shown in the syntax represent the main functions of **tar** as follows:

–c Creates an archive. This command writes each named file into a newly created archive. Directories recursively include all components. Under the USTAR (**–U**) option, **tar** records directories and other special files in the tape archive; otherwise, it ignores such files. If **–** appears in place of any filename, **tar** reads the standard input for a list of files one per line. This allows other commands to generate lists of files for **tar** to archive.

–r Writes the named files to the end of the archive. It is possible to have more than one copy of a file in a tape archive using this method. To use this form of the command with a tape, it must be possible to backspace the tape.

Note: You cannot specify both the **–r** and the **–z** option at the same time.

–t Displays a table of contents. This displays the names of all the files in the archive, one per line. If you specify one or more files on the command line, **tar** prints only those filenames. The verbose (**–v**) and (for USTAR formatted archives) the **–E** options can be used to show the attributes and extended attributes of each component.

–x Extracts files from an archive. **tar** extracts each named file to a file of the same name. If you did not specify any files on the command line, all files in the archive are extracted. This extraction restores all file system attributes as controlled by other options.

You must specify one of the preceding basic options as the first character of an option string. You can add other characters to the option string. Unlike with other commands, you must give options as a single string; for example, you might specify **–tv**, but you cannot separate them, as in “**–t –**.” You can omit the leading dash **–** if you want. Other possible options in the option string are:

b Sets the number of 512-byte blocks used for tape archive read/write operations to *blocksize*. The *blocksize* argument must be specified, and *blocksize* can be specified only when **b** is in the option string. When reading from the tape archive, **tar** automatically determines the blocking factor by trying to read the largest permitted blocking factor and using the actual number read to be the *blocksize*. For UNIX compatibility, the largest valid block size is 20 blocks; in USTAR mode, it is 60 blocks.

–C *pathname*
Is an unusual option because it is specified in the middle of your file list. When **tar** encounters a **–C** *pathname* option while archiving files, it changes the working directory (for **tar** only) to *pathname* and treats all following entries in your file list (including another **–C**) as being relative to *pathname*.

–E Same as verbose (**–v**) output, but additionally displays extended attributes. See “Output” on page 554 for more information.

f You must specify **f**. The **f** option uses the file *tapefile* for the tape archive rather than using the default. The *tapefile* argument must be specified, and *tapefile* can be specified only when **f** is in the option string. The *tapefile* argument must precede the *blocksize* argument if both are present. If *tapefile* is the character –, the standard input is used for reading archives, and the standard output is used for writing archives.

#s *#s is not supported by*. The default archive filename used by **tar** is **/dev/mt/0m**. This option is the least general way to override this default. For a more general method, see the **f** option. The filename generated by this option has the form **/dev/mt/***#s*. The *#* can be any digit between 0 and 7, inclusive, to select the tape unit. The density selector *s* can be l (low), m (medium), or h (high).

l Complains if all links are not resolved when adding files to the tape archive.

m Does not restore a file's modification time stamp when extracting it from an archive. The default behavior is to restore the time stamp from information contained in the archive.

o When writing files to an archive, does not record owner and modes of directories in the archive. If this is specified when extracting from an existing **ar** archive, **tar** does not restore any owner and group information in the archive. The default is to record this information when creating a **tar** archive, and to restore it when extracting from the archive.

–O For USTAR formatted files, tells **tar** to disable extended USTAR support. When reading or listing an archive, ignore any extended support encoded within the archive. Should an archive contain OS/390 special header files, these will be displayed or restored. This option has no affect for non-USTAR formatted archives.

OS/390 extended USTAR support is implemented in a manner which should be tolerated by older OS/390 versions of **pax** and **tar** utilities or POSIX-compliant versions on non-OS/390 UNIX platforms. So, the **-O** option is not typically needed, however, it can be used to insure portability of archives when they will be processed by these other versions. If **-O** is specified, attributes recognized by the OS/390 extended USTAR support such as external links and extended attributes will not be restored.

For more information, see “OS/390 Extended USTAR Support” on page 441.

–X For USTAR formatted archives, **–X** enables extended USTAR support. **–X** is the default. This option has no affect for non-USTAR formatted archives. **tar –X** functions in the following manner:

- During archive writing, **-X** causes **tar** to preserve extended USTAR information.
- During archive listing, **-X** causes **tar** to display extended USTAR information.
- During archive reading, **-X** enables **tar** to restore extended USTAR information

Note: To restore certain information, the user must also have the appropriate privileges and have specified the corresponding options. For example, you must specify **–p** to restore extended attributes.

For more information on extended attributes, see “OS/390 Extended USTAR Support” on page 441.

p When extracting, restores the three high-order file permission bits, exactly as in the archive. They indicate the set-user-ID, set-group-ID, and sticky bit. For USTAR formatted archives, **p** also restores, if present, extended attributes.

Note: If **-O** is specified, it overrides **-p** for extended attributes- they will not be restored.

To use **-p** on UNIX systems, you must have appropriate privileges; **tar** restores the modes restored exactly as in the archive and ignores the UMASK.

U When creating a new tape archive with the **–c** option, forces **tar** to use the USTAR format. The default format used when creating a new archive is the original UNIX **tar** format. When you do not specify **–c**, **tar** can deduce whether the tape archive is in USTAR format by reading it, so you can use **U** to suppress a warning about USTAR format.

In order to save external links and extended attributes, the USTAR format must be used.

v Displays each filename, along with the appropriate action key letter as it processes the archive. With the **–t** form of the command, this option gives more detail about each archive member being listed and shows information about the members in the same format used by the **ls –l** command. See “Output” for more information.

–V *volpat*
Provides automatic multivolume support. **tar** writes output to files—the names of which are formatted with *volpat*. Any occurrence of # in *volpat* is replaced by the current volume number. When you invoke **tar** with this option, it prompts for the first number in the archive set, and waits for you to type the number and a carriage return before proceeding with the operation. **tar** issues the same sort of message when a write error or read error occurs on the archive; this kind of error means that **tar** has reached the end of the volume and should go on to a new one.

w Is used to confirm each operation, such as replacing or extracting. **tar** displays the operation and the file involved. You can then confirm whether you want the operation to take place. Typing in an answer that begins with “y” tells **tar** to do the operation; anything else tells **tar** to go on to the next operation.

z Reads or writes, or both reads and writes, the tape archive by first passing through a compression algorithm compatible with that of **compress**.

Note: You cannot specify both the **–r** option and the **–z** option at the same time.

Output

When the **–v** or **–E** option is used with **–t** (table of contents), **tar** produces a verbose table of contents for the archive. The output for **–v** is equivalent to the output from the **ls –l** command with following exceptions:

- The following notation is used to represent hard, symbolic, and external links:

  ```
  hlink external link to origfile
  ```

  indicates that hlink is a hard link of origfile.

  ```
  slink symbolic link to origfile
  ```

  indicates that slink is a symbolic link to origfile.

```
elink external link to ORIG.FILE
```

indicates that elink is an external link to ORIG.FILE.

- For symbolic and external links, **pax** output always shows a filesize of 0

Refer to the description of **ls** for an explanation of the **ls –v**.

The output from the **–E** option has the same format as **–v** and additionally displays a column showing the extended attributes:

a	Program runs APF-authorized if linked AC=1
p	Program is considered program-controlled
s	Program runs in a shared address space
l	Program is loaded from the shared library region
	Note: l is a lower-case L, not an upper-case i.
–	attribute not set

The format of the **tar –E** output is variable in length and will be extended as necessary to display additional file characteristics that are not supported by **tar –v** (**ls –l**).

Examples

1. The following command takes a directory and places it in an archive in compressed format:

   ```
   tar –cvzf archive directory
   ```

2. To identify all files that have been changed in the last week (7 days), and to archive them to the **/tmp/posix/testpgm** file, enter:

   ```
   find /tmp/posix/testpgm –type f –mtime –7 | tar –cvf testpgm.tar –
   ```

   –type –f tells **find** to select only files. This avoids duplicate input to **tar**.

Localization

tar uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion

1 Failure due to any of the following:

- Incorrect option
- Incorrect command-line arguments
- Out of memory
- Compression error
- Failure on extraction
- Failure on creation

Usage Notes

1. Use the **pax** command if you need to use multibyte patterns when searching for filenames.
2. The POSIX 1003.1 standard defines formats for **pax** and **tar** archives that limit the length of the target of a link file to 100 characters or less.

   Note: In the case of a hard link, the target is the first occurrence of the hard link which is archived. Subsequent hard links refer to the first instance.

   Beginning with OS/390 Release 9, **pax** and **tar** provide extended USTAR support that allows these links to be preserved when creating an archive and restored when reading an archive. See “OS/390 Extended USTAR Support” on page 441 for more information.
3. On the OS/390 system, superuser privileges or read access to the appropriate FACILITY classes are required to create character special files, to restore user and group names, and to set certain extended attributes.
4. Pathnames in the tape archive are normally restricted to a maximum length of 100 bytes. However, in USTAR mode, pathnames can be up to 255 bytes long.
5. When transferring archives between OS/390 and other UNIX systems, note the following:
   a. File transfers (for example, using OPUT/OGET or ftp put/get) must be done using binary or image format. This is true, even for archives consisting only of text files.
   b. You may need to convert text files from EBCDIC to ASCII (or some other character set). You can use the **iconv** utility to convert files before or after archiving. When text files are being created or extracted, you can use the **pax –o** option to convert them.

Portability

4.2BSD

The **–U** option is an extension to provide POSIX USTAR format compatibility. The **–p** option is an common extension on BSD UNIX systems that is not available on UNIX System V systems. The **–O** and**–X** options are also extensions.

Related Information

cpio, **pax**

Also see the **pax** file format description in Appendix H for more information.

---

tcsh — Invoke a C shell

Format

tcsh [**–bcdeFfimnqstvVxX**]
tcsh –l

Note: **–l** is a lowercase L, not an uppercase i.

Description

tcsh contains the following sections and subsections:

- Options and invocation
- Options
- Editing
- Command syntax
- Substitutions
- Command Execution
- Features
- Jobs
- Status Reporting
- Automatic, Periodic, and Time Events
- Native Language System Report
- Signal Handling
- Built-in Commands
- Shell and Environment Variables
- Files
- Problems and Limitations

Options and Invocation

The tcsh shell is an enhanced but completely compatible version of the Berkeley UNIX C shell, csh. It is a command language interpreter usable both as an interactive login shell and a shell script command processor. It includes a command-line editor, programmable word completion, spelling correction, a history mechanism, job control, and a C-like syntax.

You can invoke the shell by typing an explicit **tcsh** command. A login shell can also be specified by invoking the shell with the **–l** option as the only argument.

A login shell begins by executing commands from the system files **/etc/csh.cshrc** and **/etc/csh.login**. It then executes commands from files in the user's home directory: first **~/.tcshrc**, then **~/.history** (or the value of the **histfile** shell variable), then **~/.login**, and finally **~/.cshdirs** (or the value of the **dirsfile** shell variable). The shell reads **/etc/csh.login** after **/etc/csh.cshrc**.

Non-login shells read only **/etc/csh.cshrc** and **~/.tcshrc** or **~/.cshrc** on invocation.

Commands like **stty**, which need be run only once per login, usually go in the user's **~/.login** file.

In the normal case, the shell begins reading commands from the terminal, prompting with >. The shell repeatedly reads a line of command input, breaks it into words, places it on the command history list, and then parses and executes each command in the line. See “Command Execution” on page 576.

A user can log out of a tcsh shell session by typing ^D, **logout**, or **login** on an empty line (see **ignoreeof** shell variable), or via the shell's autologout mechanism. When a login shell terminates, it sets the **logout** shell variable to *normal* or *automatic* as appropriate, then executes commands from the files **/etc/csh.logout** and **~/.logout**.

Note: The names of the system login and logout files vary from system to system for compatibility with different csh variants; see “tcsh Files” on page 599.

Options

If the first argument (argument 0) to the tcsh shell is - (hyphen), then it is a login shell. You can also specify the login shell by invoking the tcsh shell with the **–l** as the only argument.

The OS/390 UNIX System Services tcsh shell accepts the following options on the command line:

–b Forces a break from option processing, causing any further shell arguments to be treated as non-option arguments. The remaining arguments will not be interpreted as shell options. This may be used to pass options to a shell script without confusion or possible subterfuge.

–c Reads and executes commands stored in the command shell (this option must be present and must be a single arugment). Any remaining arguments are placed in the **argv** shell variable.

–d Loads the directory stack from **~/.cshdirs** as described under “Options and Invocation” on page 557, whether or not it is a login shell.

–e Terminates shell if any invoked command terminates abnormally or yields a non-zero exit status.

–i Invokes an interactive shell and prompts for its top-level input, even if it appears to not be a terminal. Shells are interactive without this option if their inputs and outputs are terminals.

–l Invokes a login shell. Only applicable if **–l** is the only option specified.

Note: **–l** is a lower-case L not an upper-case i.

–m Loads **~/.tcshrc** even if it does not belong to the effective user.

–n Parses commands but does not execute them. This aids in debugging shell scripts.

–q Accepts SIGQUIT and behaves when it is used under a debugger. Job control is disabled.

–s Take command input from the standard input.

–t Reads and executes a single line of input. A \ (backslash) may be used to escape the newline at the end of this line and continue onto another line.

–v Sets the **verbose** shell variable so command input is echoed after history substitution.

–x Sets the **echo** shell variable so commands are echoed immediately before execution.

–V Sets the **verbose** shell variable even before executing **~/.tcshrc**.

–X Is to **–x** as **–V** is to **–v**.

After processing of option arguments, if arguments remain but none of the **–c**, **–i**, **–s**, or **–t** were given, the first argument is taken as the name of a file of commands, or script , to be executed. The shell opens this file and saves its name for possible resubstitution by $0. Since many systems use shells whose shell scripts are not compatible with this shell, the tcsh shell uses such a **standard** shell to execute a script whose character is not a #, that is, which does not start with a comment.

Remaining arguments are placed in the **argv** shell variable.

tcsh shell Editing

In this section, we first describe the Command-Line Editor. We then discuss Completion and Listing and Spelling Correction which describe two sets of functionality that are implemented as editor commands but which deserve their own treatment. Finally, the Editor Commands section lists and describes the editor commands specific to the tcsh shell and their default bindings.

tcsh shell Command-Line Editor

Command-line input can be edited using key sequences much like those used in GNU Emacs or vi. The editor is active only when the edit shell variable is set, which it is by default in interactive shells. The **bindkey** built-in command can display and change key bindings. Emacs-style key bindings are used by default, but **bindkey** can change the key bindings to vi-style bindings.

The shell always binds the arrow keys to:

down down-history

up up-history

left backward-char

right forward-char

unless doing so would alter another single-character binding. One can set the arrow key escape sequences to the empty string with **settc** to prevent these bindings.

Other key bindings are, for the most part, what Emacs and vi users would expect and can easily be displayed by **bindkey**, so there is no need to list them here. Likewise, **bindkey** can list the editor commands with a short description of each.

Note: Editor commands do not have the same notion of a *word* as does the tcsh shell. The editor delimits words with any non-alphanumeric characters not in the shell variable wordchars, while the tcsh shell recognizes only whitespace and some of the characters with special meanings to it, listed under “Command Syntax” on page 567.

Completion and Listing

The tcsh shell is often able to complete words when given a unique abbreviation. Type part of a word (for example **ls** */usr/lost*) and press the tab key to run the **complete-word** editor command. The shell completes the filename **/usr/lost** to **/usr/lost+found/**, replacing the incomplete word with the complete word in the input buffer. (Note the terminal / (forward slash); completion adds a / to the end of completed directories and a space to the end of other completed words, to speed typing and provide a visual indicator of successful completion. The **addsuffix** shell variable can be unset to prevent this.) If no match is found (for example, **/usr/lost+found** doesn't exist), the terminal bell rings. If the word is already complete (for example, there is a **/usr/lost** on your system, or you were thinking too far ahead and typed the whole thing), a / or space is added to the end if it isn't already there.

Completion works anywhere in the line, not just at the end; completed text pushes the rest of the line to the right. Completion in the middle of a word often results in leftover characters to the right of the cursor which need to be deleted.

Commands and variables can be completed in much the same way. For example, typing **em** *[tab]* would complete 'em' to 'emacs' if **emacs** were the only command

on your system beginning with 'em'. Completion can find a command in any directory in the path or if given a full pathname. Typing **echo** *$ar[tab]* would complete '$ar' to '$argv' if no other variable began with 'ar'.

The shell parses the input buffer to determine whether the word you want to complete should be completed as a filename, command or variable. The first word in the buffer and the first word following ';', 'l', 'l&', '&&' or 'll' is considered to be a command. A word beginning with '$' is considered to be a variable. Anything else is a filename. An empty line is **completed** as a filename.

You can list the possible completions of a word at any time by typing ^D to run the **delete-char-or-list-or-eof** editor command. The tcsh shell lists the possible completions using the **ls-F** built-in and reprints the prompt and unfinished command line, for example:

```
> ls /usr/l['^D]
lbin/ lib/ local/ lost+found/
> ls /usr/l
```

If the **autolist** shell variable is set, the tcsh shell lists the remaining choices (if any) whenever completion fails:

```
> set autolist
> nm /usr/lib/libt[tab]
libtermcap.a@ libtermlib.a@
> nm /usr/lib/libterm
```

If **autolist** is set to *ambiguous*, choices are listed only if multiple matches are possible, and if the completion adds no new characters to the name to be matched.

A filename to be completed can contain variables, your own or others' home directories abbreviated with ~ (tilde; see “Filename Substitution” on page 574) and directory stack entries abbreviated with = (equal; see “Directory Stack Substitution” on page 575). For example:

```
> ls ~k[^D]
kahn kas kellogg
> ls ~ke[tab]
> ls ~kellogg/

or

> set local = /usr/local
> ls $lo[tab]
> ls $local/[^D]
bin/ etc/ lib/ man/ src/
> ls $local/
```

Variables can also be expanded explicitly with the **expand-variables** editor command.

delete-char-or-list-or-eof only lists at the end of the line; in the middle of a line it deletes the character under the cursor and on an empty line it logs one out or, if **ignoreeof** is set, does nothing. M-^D, bound to the editor command **list-choices**, lists completion possibilities anywhere on a line, and **list-choices** (or any one of the related editor commands which do or don't delete, list and/or log out, listed

under **delete-char-or-list-or-eof**) can be bound to ^D with the **bindkey** built-in command if so desired.

The **complete-word-fwd** and **complete-word-back** editor commands (not bound to any keys by default) can be used to cycle up and down through the list of possible completions, replacing the current word with the next or previous word in the list.

The tcsh shell variable **fignore** can be set to a list of suffixes to be ignored by completion. Consider the following:

```
> ls
Makefile condiments.h~ main.o side.c
README main.c meal side.o
condiments.h main.c~
> set fignore = (.o \~)
> emacs ma[^D]
main.c main.c~ main.o
> emacs ma[tab]
> emacs main.c
```

'main.c~' and 'main.o' are ignored by completion (but not listing), because they end in suffixes in **fignore**. \ is needed in front of ~ to prevent it from being expanded to **home** as described under “Filename Substitution” on page 574. **fignore** is ignored if only one completion is possible.

If the **complete** shell variable is set to *enhance*, completion: 1.) ignores case and 2.) considers periods, hyphens and underscores ('.', '-' and '_') to be word separators and hyphens and underscores to be equivalent.

If you had the following files:

```
comp.lang.c comp.lang.perl comp.std.c++
comp.lang.c++ comp.std.c
```

and typed **mail -f** *c.l.c[tab]*, it would be completed to **mail -f** *comp.lang.c*, and ^D would list comp.lang.c and comp.lang.c++. **mail -f** *c..c++[^D]* would list comp.lang.c++ and comp.std.c++. Typing **rm** *a--file[^D]* in the following directory

```
A_silly_file a-hyphenated-file another_silly_file
```

would list all three files, because case is ignored and hyphens and underscores are equivalent. Periods, however, are not equivalent to hyphens or underscores.

Completion and listing are affected by several other tcsh shell variables: **recexact** can be set to complete on the shortest possible unique match, even if more typing might result in a longer match. For example:

```
> ls
fodder foo food foonly
> set recexact
> rm fo[tab]
```

just beeps, because 'fo' could expand to 'fod' or 'foo', but if we type another 'o',

```
> rm foo[tab]
> rm foo
```

the completion completes on 'foo', even though 'food' and 'foonly' also match. **autoexpand** can be set to run the **expand-history** editor command before each completion attempt, and **correct** can be set to complete commands automatically after one hits 'return'. **matchbeep** can be set to make completion beep or not beep

in a variety of situations, and **nobeep** can be set to never beep at all. **nostat** can be set to a list of directories and/or patterns which match directories to prevent the completion mechanism from stat(2)ing those directories.

Note: The completion operation succeeds, but faster. The setting of **nostat** is evident when using the **listflags** variable. For example:

```
>set listflags=x
>ls-F /u/pluto
Dir1/exe1*
>set nostat=(/u/pluto/)
>ls-F /u/pluto
Dir1exe1
>
```

Although, you must be careful when setting **nostat** to keep the trailing / (forward slash).

listmax and **listmaxrows** can be set to limit the number of items and rows (respectively) that are listed without asking first. **recognize_only_executables** can be set to make the shell list only executables when listing commands, but it is quite slow.

Finally, the **complete** built-in command can be used to tell the shell how to complete words other than filenames, commands and variables. Completion and listing do not work on glob-patterns (see “Filename Substitution” on page 574), but the **list-glob** and **expand-glob** editor commands perform equivalent functions for glob-patterns.

Spelling Correction

The tcsh shell can sometimes correct the spelling of filenames, commands and variable names as well as completing and listing them.

Individual words can be spelling-corrected with the **spell-word** editor command (usually bound to M-s and M-S where M = Meta Key or escape (ESC) key) and the entire input buffer with **spell-line** (usually bound to M-$). The **correct** shell variable can be set to 'cmd' to correct the command name or 'all' to correct the entire line each time return is typed.

When spelling correction is invoked in any of these ways and the shell thinks that any part of the command line is misspelled, it prompts with the corrected line:

```
> set correct = cmd
> lz /usr/bin
CORRECT>ls /usr/bin (y|n|e|a)?
```

where one can answer 'y' or space to execute the corrected line, 'e' to leave the uncorrected command in the input buffer, 'a' to abort the command as if ^C had been hit, and anything else to execute the original line unchanged.

Spelling correction recognizes user-defined completions (see the **complete** built-in command). If an input word in a position for which a completion is defined resembles a word in the completion list, spelling correction registers a misspelling and suggests the latter word as a correction. However, if the input word does not match any of the possible completions for that position, spelling correction does not register a misspelling.

Like completion, spelling correction works anywhere in the line, pushing the rest of the line to the right and possibly leaving extra characters to the right of the cursor.

Attention: Spelling correction is not guaranteed to work the way one intends, and is provided mostly as an experimental feature.

Editor Commands

bindkey lists key bindings and **bindkey -l** lists and briefly describes editor commands. Only new or especially interesting editor commands are described here. See **emacs** and **vi** for descriptions of each editor's key bindings.

The character or characters to which each command is bound by default is given in parentheses. ^*character* means a control character and M-*character* a meta character, typed as escape-character on terminals without a meta key. Case counts, but commands which are bound to letters by default are bound to both lower- and uppercase letters for convenience.

complete-word
Completes a word as described under “Completion and Listing” on page 559.

complete-word-back
Like **complete-word-fwd**, but steps up from the end of the list.

complete-word-fwd
Replaces the current word with the first word in the list of possible completions. May be repeated to step down through the list. At the end of the list, beeps and reverts to the incomplete word.

complete-word-raw
Like **complete-word**, but ignores user-defined completions.

copy-prev-word
Copies the previous word in the current line into the input buffer. See also **insert-last-word**.

dabbrev-expand
Expands the current word to the most recent preceding one for which the current is a leading substring, wrapping around the history list (once) if necessary. Repeating **dabbrev-expand** without any intervening typing changes to the next previous word etc., skipping identical matches much like **history-search-backward** does.

delete-char (not bound)
Deletes the character under the cursor. See also **delete-char-or-list-or-eof**.

delete-char-or-eof (not bound)
Does **delete-char** if there is a character under the cursor or **end-of-file** on an empty file. See also **delete-char-or-list-or-eof**.

delete-char-or-list (not bound)
Does **delete-char** if there is a character under the cursor or list-choices at the end of the line. See also **delete-char-or-list-or-eof**.

delete-char-or-list-or-eof (^D)
Does **delete-char** if there is a character under the cursor, **list-choices** at the end of the line or **end-of-file** on an empty line. See also **delete-char-or-eof**, **delete-char-or-list** and **list-or-eof**.

down-history
Like **up-history**, but steps down, stopping at the original input line.

end-of-file
Signals an end of file, causing the tcsh shell to exit unless the **ignoreeof** shell variable is set to prevent this. See also **delete-char-or-list-or-eof**.

expand-history (M-space)
Expands history substitutions in the current word. See “History Substitution” on page 568. See also **magic-space**, **toggle-literal-history**, and the **autoexpand** shell variable.

expand-glob(^X-*)
Expands the glob-pattern to the left of the cursor. For example:

```
>ls test*[^X-*]
```

would expand to

```
>ls test1.c test2.c
```

if those were the only two files in your directory that begin with 'test'. See “Filename Substitution” on page 574.

expand-line (not bound)
Like **expand-history**, but expands history substitutions in each word in the input buffer.

expand-variables (^X-$)
Expands the variable to the left of the cursor. See “Variable Substitution” on page 572.

history-search-backward (M-p, M-P)
Searches backwards through the history list for a command beginning with the current contents of the input buffer up to the cursor and copies it into the input buffer. The search string may be a glob-pattern (see “Filename Substitution” on page 574) containing '*', '?', '[]' or '{}'. **up-history** and **down-history** will proceed from the appropriate point in the history list. Emacs mode only. See also **history-search-forward** and **i-search-back**.

history-search-forward(M-n, M-N)
Like **history-search-backward**, but searches forward.

i-search-back (not bound)
Searches backward like **history-search-backward**, copies the first match into the input buffer with the cursor positioned at the end of the pattern, and prompts with 'bck: ' and the first match. Additional characters may be typed to extend the search. **i-search-back** may be typed to continue searching with the same pattern, wrapping around the history list if necessary, (**i-search-back** must be bound to a single character for this to work) or one of the following special characters may be typed:

^W Appends the rest of the word under the cursor to the search pattern.

delete (or any character bound to backward-delete-char)
Undoes the effect of the last character and deletes a character from the search pattern if appropriate.

^G If the previous search was successful, aborts the entire search. If not, goes back to the last successful search.

escape
Ends the search, leaving the current line in the input buffer.

Any other character not bound to **self-insert-command** terminates the search, leaving the current line in the input buffer, and is then interpreted as

normal input. In particular, a carriage return causes the current line to be executed. Emacs mode only. See also **i-search-fwd** and **history-search-backward**.

i-search-fwd
Like i-search-back, but searches forward.

insert-last-word (M-_)
Inserts the last word of the previous line (!$) into the input buffer. See also **copy-prev-word**.

list-choices (M-D)
Lists completion possibilities as described under “Completion and Listing” on page 559. See also **delete-char-or-list-or-eof**.

list-choices-raw (^X-^D)
Like **list-choices**, but ignores user-defined completions.

list-glob (^X-g, ^X-G)
Lists (via the **ls-F**) matches to the glob-pattern (see “Filename Substitution” on page 574) to the left of the cursor.

list-or-eof (not bound)
Does **list-choices** or **end-of-file** on an empty line. See also **delete-char-or-list-or-eof**.

magic-space (not bound)
Expands history substitutions in the current line, like **expand-history**, and appends a space. **magic-space** is designed to be bound to the spacebar, but is not bound by default.

normalize-command (^X-?)
Searches for the current word in PATH and, if it is found, replaces it with the full path to the executable. Special characters are quoted. Aliases are expanded and quoted but commands within aliases are not. This command is useful with commands which take commands as arguments, for example, **dbx** and **sh -x**.

normalize-path (^X-n, ^X-N)
Expands the current word as described under the *expand* setting of the **symlinks** shell variable.

overwrite-mode (unbound)
Toggles between input and overwrite modes.

run-fg-editor (M-^Z)
Saves the current input line and looks for a stopped job with a name equal to the last component of the file name part of the EDITOR or VISUAL environment variables, or, if neither is set, ed or vi. If such a job is found, it is restarted as if **fg %job** had been typed. This is used to toggle back and forth between an editor and the shell easily. Some people bind this command to ^Z so they can do this even more easily.

run-help (M-h, M-H)
Searches for documentation on the current command, using the same notion of **current command** as the completion routines, and prints it. There is no way to use a pager; **run-help** is designed for short help files. Documentation should be in a file named *command*.help, *command*.1, *command*.6, *command*.8 or *command*, which should be in one of the directories listed in

the HPATH enviroment variable. If there is more than one help file only the first is printed.

self-insert-command (text characters)
In insert mode (the default), inserts the typed character into the input line after the character under the cursor. In overwrite mode, replaces the character under the cursor with the typed character. The input mode is normally preserved between lines, but the **inputmode** shell variable can be set to *insert* or *overwrite* to put the editor in that mode at the beginning of each line. See also **overwrite-mode**.

sequence-lead-in (arrow prefix, meta prefix, ^X)
Indicates that the following characters are part of a multi-key sequence. Binding a command to a multi-key sequence really creates two bindings: the first character to **sequence-lead-in** and the whole sequence to the command. All sequences beginning with a character bound to **sequence-lead-in** are effectively bound to **undefined-key** unless bound to another command.

spell-line (M-$)
Attempts to correct the spelling of each word in the input buffer, like **spell-word**, but ignores words whose first character is one of '-', '!', '^' or '%', or which contain '\', '*' or '?', to avoid problems with switches, substitutions and the like. See “Spelling Correction” on page 562.

spell-word (M-s, M-S)
Attempts to correct the spelling of the current word as described under “Spelling Correction” on page 562. Checks each component of a word which appears to be a pathname.

toggle-literal-history (M-r, M-R)
Expands or unexpands history substitutions in the input buffer. See also **expand-history** and the **autoexpand** shell variable.

undefined-key (any unbound key)
Beeps.

up-history (up-arrow, ^P)
Copies the previous entry in the history list into the input buffer. If **histlit** is set, uses the literal form of the entry. May be repeated to step up through the history list, stopping at the top.

vi-search-back (?)
Prompts with ? for a search string (which may be a glob-pattern, as with **history-search-backward**), searches for it and copies it into the input buffer. The bell rings if no match is found. Hitting return ends the search and leaves the last match in the input buffer. Hitting escape ends the search and executes the match. vi mode only.

vi-search-fwd (/)
Like **vi-search-back**, but searches forward.

which-command (M-?)
Does a **which** (built-in command) on the first word of the input buffer.

Command Syntax

The tcsh shell splits input lines into words at blanks and tabs. The special characters '&', 'l', ';', '<', '>', '(', and ')' and the doubled characters '&&', 'll', '<<' and '>>' are always separate words, whether or not they are surrounded by whitespace.

When the tcsh shell's input is not a terminal, the character '#' is taken to begin a comment. Each # and the rest of the input line on which it appears is discarded before further parsing.

A special character (including a blank or tab) may be prevented from having its special meaning, and possibly made part of another word, by preceding it with a backslash (\) or enclosing it in single ('), double (") or backward (' ` ') quotes. When not otherwise quoted a newline preceded by a \ is equivalent to a blank, but inside quotes this sequence results in a newline.

Furthermore, all substitutions (see “Substitutions”) except history substitution can be prevented by enclosing the strings (or parts of strings) in which they appear with single quotes or by quoting the crucial character(s) (e.g. '$' or '`' for variable substitution or command substitution respectively) with \. (alias substitution is no exception: quoting in any way any character of a word for which an alias has been defined prevents substitution of the alias. The usual way of quoting an alias is to precede it with a backslash.) History substitution is prevented by backslashes but not by single quotes. Strings quoted with double or backward quotes undergo Variable substitution and Command substitution, but other substitutions are prevented.

Text inside single or double quotes becomes a single word (or part of one). Metacharacters in these strings, including blanks and tabs, do not form separate words. Only in one special case (see “Command Substitution” on page 574) can a double-quoted string yield parts of more than one word; single-quoted strings never do. Backward quotes are special: they signal command substitution, which may result in more than one word.

Quoting complex strings, particularly strings which themselves contain quoting characters, can be confusing. Remember that quotes need not be used as they are in human writing! It may be easier to quote not an entire string, but only those parts of the string which need quoting, using different types of quoting to do so if appropriate.

The **backslash_quote** shell variable can be set to make backslashes always quote \, ', and ". This may make complex quoting tasks easier, but it can cause syntax errors in csh (or tcsh) scripts.

Substitutions

This section describes the various transformations the tcsh shell performs on input in the order in which they occur. The section will cover data structures involved and the commands and variables which affect them. Remember that substitutions can be prevented by quoting as described under “Command Syntax.”

History Substitution

Each command, or **event**, input from the terminal is saved in the history list. The previous command is always saved, and the **history** shell variable can be set to a number to save that many commands. The **histdup** shell variable can be set to not save duplicate events or consecutive duplicate events.

Saved commands are numbered sequentially from 1 and stamped with the time. It is not usually necessary to use event numbers, but the current event number can be made part of the prompt by placing an exclamation point (!) in the **prompt** shell variable.

The shell actually saves history in expanded and literal (unexpanded) forms. If the **histlit** shell variable is set, commands that display and store history use the literal form.

The **history** built-in command can print, store in a file, restore and clear the history list at any time, and the **savehist** and **histfile** shell variables can be set to store the history list automatically on logout and restore it on login.

History substitutions introduce words from the history list into the input stream, making it easy to repeat commands, repeat arguments of a previous command in the current command, or fix spelling mistakes in the previous command with little typing and a high degree of confidence.

History substitutions begin with the character !. They may begin anywhere in the input stream, but they do not nest. The ! may be preceded by a \ to prevent its special meaning; for convenience, a ! is passed unchanged when it is followed by a blank, tab, newline, = or (. History substitutions also occur when an input line begins with ^. This special abbreviation will be described later. The characters used to signal history substitution (! and ^ (caret)) can be changed by setting the **histchars** shell variable. Any input line which contains a history substitution is printed before it is executed.

A history substitution may have an **event specification**, which indicates the event from which words are to be taken, a **word designator**, which selects particular words from the chosen event, and/or a **modifier**, which manipulates the selected words.

An event specification can be

n A number, referring to a particular event

–n An offset, referring to the even *n* before the current event

The current event. This should be used carefully in csf, where there is no check for recursion. tcsh allows 10 levels of recursion.

! The previous event (equivalent to -1)

s The most recent event whose first word begins with the string *s*

?*s***?** The most recent event which contains the string *s*. The second ? can be omitted if it is immediately followed by a newline.

For example, consider this bit of someone's history list:

```
9 8:30 nroff -man wumpus.man
10 8:31 cp wumpus.man wumpus.man old
11 8:36 vi wumpus.man
12 8:37 diff wumpus.man.old wumpus.man
```

The commands are shown with their event numbers and time stamps. The current event, which we haven't typed in yet, is event 13. !11 and !-2 refer to event 11. !! refers to the previous event, 12. !! can be abbreviated ! if it is followed by : (colon; described below). !n refers to event 9, which begins with *n*. !?old? also refers to event 12, which contains *old*. Without word designators or modifiers history references simply expand to the entire event, so we might type !cp to redo the copy command or !!|more if the **diff** output scrolled off the top of the screen.

History references may be insulated from the surrounding text with braces if necessary. For example, !vdoc would look for a command beginning with *vdoc*, and, in this example, not find one, but !{v}doc would expand unambiguously to vi wumpus.mandoc. Even in braces, history substitutions do not nest.

While csh expands, for example, !3d to event 3 with the letter d appended to it, tcsh expands it to the last event beginning with 3d; only completely numeric arguments are treated as event numbers. This makes it possible to recall events beginning with numbers. To expand !3d as in csh say !\3d.

To select words from an event we can follow the event specification by a : (colon) and a designator for the desired words. The words of an input line are numbered from 0, the first (usually command) word being 0, the second word (first argument) being 1, etc. The basic word designators are:

0 The first command word

n The *n*th argument

^ The first argument, equivalent to 1

$ The last argument

% The word matched by an ?*s*? search

x-y A range of words

–y Equivalent to 0–y

***** Equivalent to ^–$, but returns nothing if the event contains only 1 word

x* Equivalent to *x-$*

x- Equivalent to *x**, but omitting the last word ($)

Selected words are inserted into the command line separated by single blanks. For example, the **diff** command in the previous example might have been typed as **diff** *!!:1.old !!:1*(using :1 to select the first argument from the previous event) or **diff** *!-2:2 !-2:1*to select and swap the arguments from the **cp** command. If we didn't care about the order of the **diff** we might have said **diff** *!-2:1-2*or simply **diff** *!-2:**. The **cp** command might have been written **cp** *wumpus.man !#:1.old*, using # to refer to the current event. !n:- hurkle.man would reuse the first two words from the **nroff** command to say **nroff -man** *hurkle.man*.

The : separating the event specification from the word designator can be omitted if the argument selector begins with a '^', '$', '*', '%' or '-'. For example, our **diff** command might have been **diff** *!!^.old !!^* or, equivalently, **diff** *!!$.old !!$*.

However, if !! is abbreviated !, an argument selector beginning with - (hypen) will be interpreted as an event specification.

A history reference may have a word designator but no event specification. It then references the previous command. Continuing our **diff** example, we could have said simply **diff** *!^.old !^*or, to get the arguments in the opposite order, just **diff** *!**.

The word or words in a history reference can be edited, or **modified**, by following it with one or more modifiers, each preceded by a : (colon):

h Remove a trailing pathname component, leaving the head.

t Remove all leading pathname components, leaving the tail.

r Remove a filename extension .xxx, leaving the root name.

e Remove all but the extenstion

u Uppercase the first lowercase letter.

l Lowercase the first uppercase letter.

s/l/r Substitute *l* for *r*. *l* is simply a string like *r*, not a regular expression as in the eponymous **ed** command. Any character may be used as the delimiter in place of /; a \ can be used to quote the delimiter inside *l* and *r*. The character & in the *r* is replaced by *l*; \ also quotes &. If *l* is empty (""), the *l* from a previous substitution or the *s* from a previous ?*s*? event specification is used. The trailing delimiter may be omitted if it is immediately followed by a newline.

& Repeat the previous substitution

g Apply the following modifier once to each word.

a Apply the following modifier as many times as possible to a single word. 'a' and 'g' can be used together to apply a modifier globally. In the current implementation, using the 'a' and 's' modifiers together can lead to an infinite loop. For example, :as/f/ff/ will never terminate. This behavior might change in the future.

p Print the new command line but do not execute it.

q Quote the substituted words, preventing further substitutions.

x Like **q**, but break into words at blanks, tabs and newlines.

Modifiers are applied only to the first modifiable word (unless 'g' is used). It is an error for no word to be modifiable.

For example, the **diff** command might have been written as **diff** *wumpus.man.old !#^:r*, using :r to remove *.old* from the first argument on the same line (!#^). We could say **echo** *hello out there*, then **echo** *!*:u* to capitalize 'hello', **echo** *!*:au* to say it out loud, or **echo** *!*:agu* to really shout. We might follow **mail -s** *"I forgot my password" rot* with *!:s/rot/root* to correct the spelling of 'root' (but see “Spelling Correction” on page 562 for a different approach).

There is a special abbreviation for substitutions. ^, when it is the first character on an input line, is equivalent to *!:s^*. Thus, we might have said *^rot^root* to make the spelling correction in the previous example. This is the only history substitution which does not explicitly begin with !.

In csh as such, only one modifier may be applied to each history or variable expansion. In tcsh, more than one may be used, for example

```
% mv wumpus.man /usr/man/man1/wumpus.1
% man !$:t:r
man wumpus
```

In csh, the result would be *wumpus.1:r*. A substitution followed by a colon may need to be insulated from it with braces:

```
> mv a.out /usr/games/wumpus
> setenv PATH !$:h:$PATH
Bad ! modifier: $.
> setenv PATH !{-2$:h}:$PATH
setenv PATH /usr/games:/bin:/usr/bin:.
```

The first attempt would succeed in csh but fails in tcsh, because tcsh expects another modifier after the second colon instead of $.

Finally, history can be accessed through the editor as well as through the substitutions just described. The following commands search for events in the history list and compile them into the input buffer:

- **up-history**
- **down-history**
- **history-search-backward**
- **history-search-forward**
- **i-search-back**
- **i-search-fwd**
- **vi-search-back**
- **vi-search-fwd**
- **copy-prev-word**
- **insert-last-word**

The **toggle-literal-history** editor command switches between the expanded and literal forms of history lines in the input buffer. **expand-history** and **expand-line** expand history substitutions in the current word and in the entire input buffer respectively.

Alias Substitution

The shell maintains a list of aliases which can be set, unset and printed by the **alias** and **unalias** commands. After a command line is parsed into simple commands (see “Command Execution” on page 576) the first word of each command, left-to-right, is checked to see if it has an alias. If so, the first word is replaced by the alias. If the alias contains a history reference, it undergoes history substitution as though the original command were the previous input line. If the alias does not contain a history reference, the argument list is left untouched.

Thus if the alias for **ls** were **ls -l** the command **ls** */usr*would become **ls -l** */usr*, the argument list here being undisturbed. If the alias for **lookup** were **grep** *!^ /etc/passwd* then **lookup bill** would become **grep** *bill /etc/passwd*. Aliases can be used to introduce parser metasyntax. For example, **alias print** *'pr \!* | lpr'* defines a **command** (**print**) which prints its arguments to the line printer.

Alias substitution is repeated until the first word of the command has no alias. If an alias substitution does not change the first word (as in the previous example) it is flagged to prevent a loop. Other loops are detected and cause an error.

Some aliases are referred to by the shell; see “tcsh Built-in Commands” on page 583.

Variable Substitution

The tcsh shell maintains a list of variables, each of which has as value a list of zero or more words. The values of tcsh shell variables can be displayed and changed with the **set** and **unset** commands. The system maintains its own list of "environment" variables. These can be displayed and changed with **printenv**, **setenv** and **unsetenv**.

Variables may be made read-only with **set -r**. Read-only variables may not be modified or unset; attempting to do so will cause an error. Once made read-only, a variable cannot be made writable, so **set -r** should be used with caution. Environment variables cannot be made read-only.

Some variables are set by the tcsh shell or referred to by it. For instance, the **argv** variable is an image of the shell's argument list, and words of this variable's value are referred to in special ways. Some of the variables referred to by the tcsh shell are toggles; the shell does not care what their value is, only whether they are set or not. For instance, the **verbose** variable is a toggle which causes command input to be echoed. The **-v** command line option sets this variable. Special shell variables lists all variables which are referred to by the shell.

Other operations treat variables numerically. The **@** (at) command permits numeric calculations to be performed and the result assigned to a variable. Variable values are, however, always represented as (zero or more) strings. For the purposes of numeric operations, the null string is considered to be zero, and the second and subsequent words of multiword values are ignored.

After the input line is aliased and parsed, and before each command is executed, variable substitution is performed keyed by $ characters. This expansion can be prevented by preceding the $ with a \ except within double quotes (") where it always occurs, and within single quotes (') where it never occurs. Strings quoted by backward quotes or accents (`) are interpreted later (see “Command Substitution” on page 574) so $ substitution does not occur there until later, if at all. A $ is passed unchanged if followed by a blank, tab, or end-of-line.

Input/output redirections are recognized before variable expansion, and are variable expanded separately. Otherwise, the command name and entire argument list are expanded together. It is thus possible for the first (command) word (to this point) to generate more than one word, the first of which becomes the command name, and the rest of which become arguments.

Unless enclosed in double quotes (") or given the :q modifier the results of variable substitution may eventually be command and filename substituted. Within ", a variable whose value consists of multiple words expands to a (portion of a) single word, with the words of the variable's value separated by blanks. When the :q modifier is applied to a substitution the variable will expand to multiple words with each word separated by a blank and quoted to prevent later command or filename substitution.

The following metasequences are provided for introducing variable values into the shell input. Except as noted, it is an error to reference a variable which is not set.

$*name***[***selector***]**

${*name***[***selector***]}**
Substitutes only the selected words from the value of name. The selector is subjected to $ substitution and may consist of a single number or two numbers separated by a - (hyphen). The first word of a variable's value is numbered 1. If the first number of a range is omitted it defaults to 1. If the last member of a range is omitted it defaults to *$#name*. The selector * selects all words. It is not an error for a range to be empty if the second argument is omitted or in range.

$0 Substitutes the name of the file from which command input is being read. An error occurs if the name is not known.

$*number*

${*number***}**
Equivalent to $argv[*number*].

$* Equivalent to $argv, which is equivalent to $argv[*].

The : (colon) modifiers described under "History Substitution" on page 568, except for :p, can be applied to the substitutions above. More than one may be used. Braces may be needed to insulate a variable substitution from a literal colon just as with history substitution; any modifiers must appear within the braces. The following substitutions can not be modified with : modifiers.

$?*name*

${?*name***}**
Substitutes the string `1` if *name* is set, `0` if it is not.

$0 Substitutes the name of the file from which command input is being read. An error occurs if the name is not known.

$?0 Substitutes `1` if the current input filename is known, `0` if it is not. Always `0` in interactive shells.

$#*name* **or ${#***name***}**
Substitutes the number of words in *name*.

**$#** Equivalent to '$#argv'.

$*%name*

${*%name***}**
Substitutes the number of characters in name.

$*%number*

${*%number***}**
Substitutes the number of characters in $argv[number].

$? Equivalent to $status.

$$ Substitutes the (decimal) process number of the (parent) shell.

$! Substitutes the (decimal) process number of the last background process started by this shell.

$< Substitutes a line from the standard input, with no further interpretation thereafter. It can be used to read from the keyboard in a shell script. While csh always quotes $<, as if it were equivalent to $<:q, tcsh does not. Furthermore, when tcsh is waiting for a line to be typed the user may type an interrupt to interrupt the sequence into which the line is to be substituted, but csh does not allow this.

The editor command **expand-variables**, normally bound to ^X-$, can be used to interactively expand individual variables.

Command, Filename and Directory Stack Substitiution

The remaining substitutions are applied selectively to the arguments of tcsh built-in commands. This means that portions of expressions which are not evaluated are not subjected to these expansions. For commands which are not internal to the tcsh shell, the command name is substituted separately from the argument list. This occurs very late, after input-output redirection is performed, and in a child of the main shell.

Command Substitution: Command substitution is indicated by a command enclosed in ' ' '. The output from such a command is broken into separate words at blanks, tabs and newlines, and null words are discarded. The output is variable and command substituted and put in place of the original string.

Command substitutions inside double quotes (") retain blanks and tabs; only newlines force new words. The single final newline does not force a new word in any case. It is thus possible for a command substitution to yield only part of a word, even if the command outputs a complete line.

Filename Substitution: If a word contains any of the characters '*', '?', '[' or '{' or begins with the character '~' it is a candidate for filename substitution, also known as **globbing**. This word is then regarded as a pattern (glob-pattern), and replaced with an alphabetically sorted list of file names which match the pattern.

In matching filenames, the character **.** (period) at the beginning of a filename or immediately following a / (forward slash), as well as the character / must be matched explicitly. The character * matches any string of characters, including the null string. The character ? matches any single character. The sequence [...] matches any one of the characters enclosed. Within [...], a pair of characters separated by - matches any character lexically between the two.

Some glob-patterns can be negated: The sequence [^...] matches any single character not specified by the characters and/or ranges of characters in the braces.

An entire glob-pattern can also be negated with ^:

```
> echo *
bang crash crunch ouch
> echo ^cr*
bang ouch
```

Glob-patterns which do not use '?', '*', or'[]' or which use '{}' or '^' (below) are not negated correctly.

The metanotation **a{b,c,d}e** is a shorthand for abe ace ade. Left-to-right order is preserved: **/usr/source/s1/{oldls,ls}.c** expands to /usr/source/s1/oldls.c **/usr/source/s1/ls.c**. The results of matches are sorted separately at a low level to

preserve this order, such as, like the following example, where **../{memo,*box}** might expand to **../memo ../box ../mbox**. (Note that 'memo' was not sorted with the results of matching '*box'.) It is not an error when this construct expands to files which do not exist, but it is possible to get an error from a command to which the expanded list is passed. This construct may be nested. As a special case the words {, } and {} are passed undisturbed. The character ~ at the beginning of a filename refers to home directories. Standing alone, i.e. ~, it expands to the invoker's home directory as reflected in the value of the home shell variable. When followed by a name consisting of letters, digits and - (hyphen) characters the shell searches for a user with that name and substitutes their home directory; thus **~ken** might expand to **/usr/ken** and **~ken/chmach** to **/usr/ken/chmach**. If the character ~ is followed by a character other than a letter or / or appears elsewhere than at the beginning of a word, it is left undisturbed. A command like **setenv** *MANPATH /usr/man:/usr/local/man:~/lib/man* does not, therefore, do home directory substitution as one might hope. It is an error for a glob-pattern containing '*', '?', '[' or '~', with or without '^', not to match any files. However, only one pattern in a list of glob-patterns must match a file (so that, for example, rm *.a *.c *.o would fail only if there were no files in the current directory ending in '.a', '.c', or '.o'), and if the nonomatch shell variable is set a pattern (or list of patterns) which matches nothing is left unchanged instead of causing an error.

The **noglob** shell variable can be set to prevent filename substitution, and the **expand-glob** editor command, normally bound to ^X-*, can be used to interactively expand individual filename substitutions.

Directory Stack Substitution: The directory stack is a list of directories, numbered from zero, used by the **pushd**, **popd** and **dirs** built-in commands for tcsh. **dirs** can print, store in a file, restore and clear the directory stack at any time, and the **savedirs** and **dirsfile** shell variables can be set to store the directory stack automatically on logout and restore it on login. The **dirstack** shell variable can be examined to see the directory stack and set to put arbitrary directories into the directory stack.

The character = (equal) followed by one or more digits expands to an entry in the directory stack. The special case **=-** expands to the last directory in the stack. For example,

```
> dirs -v
0 /usr/bin
1 /usr/spool/uucp
2 /usr/accts/sys
> echo =1
/usr/spool/uucp
> echo =0/calendar
/usr/bin/calendar
> echo =-
/usr/accts/sys
```

The **noglob** and **nonomatch** shell variables and the **expand-glob** editor command apply to directory stack as well as filename substitutions.

Other Substitutions: There are several more transformations involving filenames, not strictly related to the above but mentioned here for completeness. Any filename may be expanded to a full path when the **symlinks** variable is set to *expand*. Quoting prevents this expansion, and the **normalize-path** editor command does it on demand. The **normalize-command** editor command expands commands in

PATH into full paths on demand. Finally, **cd** and **pushd** interpret - (hyphen) as the old working directory (equivalent to the tcsh shell variable **owd**). This is not a substitution at all, but an abbreviation recognized only by those commands. Nonetheless, it too can be prevented by quoting.

Command Execution

The next three sections describe how the shell executes commands and deals with their input and output.

Simple Commands, Pipelines, and Sequences

A simple command is a sequence of words, the first of which specifies the command to be executed. A series of simple commands joined by '|' characters forms a pipeline. The output of each command in a pipeline is connected to the input of the next.

Simple commands and pipelines may be joined into sequences with ';', and will be executed sequentially. Commands and pipelines can also be joined into sequences with '||' or '&&', indicating, as in the C language, that the second is to be executed only if the first fails or succeeds respectively.

A simple command, pipeline or sequence may be placed in parentheses, '()', to form a simple command, which may in turn be a component of a pipeline or sequence. A command, pipeline or sequence can be executed without waiting for it to terminate by following it with an '&'.

Built-in and Non-Built-in Command Execution

tcsh Built-in commands are executed within the shell. If any component of a pipeline except the last is a built-in command, the pipeline is executed in a subshell.

Parenthesized commands are always executed in a subshell:

```
(cd; pwd); pwd
```

which prints the home directory, leaving you where you were (printing this after the home directory), while

```
cd; pwd
```

leaves you in the home directory. Parenthesized commands are most often used to prevent **cd** from affecting the current shell.

When a command to be executed is found not to be a built-in command the tcsh shell attempts to execute the command via **execve**. Each word in the variable path names a directory in which the tcsh shell will look for the command. If it is given neither a **-c** nor a **-t** option, the shell hashes the names in these directories into an internal table so that it will only try an **execve** in a directory if there is a possibility that the command resides there. This greatly speeds command location when a large number of directories are present in the search path. If this mechanism has been turned off (via **unhash**), if the shell was given a **-c** or **-t** argument or in any case for each directory component of path which does not begin with a /, the shell concatenates the current working directory with the given command name to form a pathname of a file which it then attempts to execute.

If the file has execute permissions but is not an executable to the system (that is, it is neither an executable binary nor a script which specifies its interpreter), then it is assumed to be a file containing shell commands and a new shell is spawned to

read it. The **shell** special alias may be set to specify an interpreter other than the shell itself.

Input or Output

The standard input and standard output of a command may be redirected with the following syntax:

Figure 14. Standard Input/Output Syntax for tcsh Shell

Syntax	Description
< *name*	Open file name (which is first variable, command and filename expanded) as the standard input.
<< *word*	Read the shell input up to a line which is identical to *word*. *word* is not subjected to variable, filename or command substitution, and each input line is compared to *word* before any substitutions are done on this input line. Unless a quoting \, " , ' ' or ' ' ' appears in *word* variable and command substitution is performed on the intervening lines, allowing \ to quote $, \ and ' (single quote). Commands which are substituted have all blanks, tabs, and newlines preserved, except for the final newline which is dropped. The resultant text is placed in an anonymous temporary file which is given to the command as standard input.
> *name* >! *name* >& *name* >&! *name*	The file *name* is used as standard output. If the file does not exist then it is created; if the file exists, its is overwritten and, therefore, the previous contents are lost. If the shell variable **noclobber** is set, then the file must not exist or be a character special file (for example, a terminal or **/dev/null**) or an error results. This helps prevent accidental destruction of files. In this case the ! forms can be used to suppress this check. The forms involving & (ampersand) route the diagnostic output into the specified file as well as the standard output. *name* is expanded in the same way as < input filenames are.
>> *name* >>& *name* >>! *name* >>&! *name*	Like >, but appends output to the end of *name*. If the shell variable *noclobber* is set, then it is an error for the file *not* to exist, unless one of the ! forms is given.

A command receives the environment in which the shell was invoked as modified by the input-output parameters and the presence of the command in a pipeline. Thus, unlike some previous shells, commands run from a file of shell commands have no access to the text of the commands by default; instead they receive the original standard input of the shell. The << mechanism should be used to present inline data. This permits shell command scripts to function as components of pipelines and allows the shell to block read its input. The default standard input for a command run detached is *not* the empty file **/dev/null**, but the original standard input of the shell. If this is a terminal and if the process attempts to read from the terminal, then the process will block and the user will be notified (see “Jobs” on page 580).

Diagnostic output may be directed through a pipe with the standard output. Simply use the form |& instead of just |.

The shell cannot presently redirect diagnostic output without also redirecting standard output, but (command > output-file) >& error-file is often an acceptable

workaround. Either output-file or error-file may be **/dev/tty** to send output to the terminal.

Features

Having described how the tcsh shell accepts, parses and executes command lines, we now turn to a variety of its useful features.

Control Flow

The tcsh shell contains a number of commands which can be used to regulate the flow of control in command files (shell scripts) and (in limited by useful ways) from terminal output. These commands all operate by forcing the shell to reread or skip in its input and, due to the implementation, restrict the placement of some of the commands.

The foreach, switch, and while statements, as well as the if-then-else form of the if statement, require that the major keywords appear in a single simple command on an input line.

If the shell's input is not seekable, the shell buffers up input whenever a loop is being read and performs seeks in this internal buffer to accomplish the rereading implied by the loop . (To the extent that this allows, backward gotos will succeed on non-seekable inputs.)

Expressions

The **if**, **while**, and **exit** built-in commands use expressions with a common syntax. The expressions can include any of the operators described in the next three sections. Note that the **@** built-in command has its own separate syntax.

Logical, Arithmetical and Comparison Operators: These operators are similar to those of C and have the same precendence. They include:

```
|| && | ^ & == != =~ !~ <= >=
< > << >> + - * / % ! ~ ( )
```

Here the precedence increases to the right, '==' '!=' '=~' and '!~', '<=' '>=' '<' and '>', '<<' and '>>', '+' and '-', '*' / and '%' being in groups, at the same level. The '==' '!=' '=~' and '!~' operators compare their arguments as strings; all others operate on numbers. The operators '=~' and '!~' are like '!=' and '==' except that the right hand side is a glob-pattern (see “Filename Substitution” on page 574) against which the left hand operand is matched. This reduces the need for use of the **switch** built-in command in shell scripts when all that is really needed is pattern matching.

Strings which begins with `0` are considered octal numbers. Null or missing arguments are considered `0`. The results of all expressions are strings, which represent decimal numbers. It is important to note that no two components of an expression can appear in the same word; except when adjacent to components of expressions which are syntactically significant to the parser ('$' '|' '<' '>' '(' ')') they should be surrounded by spaces.

Command Exit Status: Commands can be executed in expressions and their exit status returned by enclosing them in braces ({}). Remember that the braces should be separated from the words of the command by spaces. Command executions succeed, returning true, that is, `1`, if the command exits with status `0`, otherwise they fail, returning false (`0`). If more detailed status information is required then the

command should be executed outside of an expression and the status shell variable examined.

File Inquiry Operators: Some of these operators perform true/false tests on files and related objects. They are of the form **-op** file, where op is one of:

r Read access

w Write access

x Execute access

X Executable in the path or shell built-in. For example, **–X ls** and **–X ls-F** are generally true, but **–X /bin/ls** is not.

e Existence

o Ownership

x Zero size

s Non-zero size

f Plain file

d Directory

l Symbolic link

b Block special file

c Character special file

p Named pipe (fifo)

S Socket special file

u Set-user ID bit is set

g Set-group-ID bit is set

k Sticky bit is set

t t *file_descriptor* (which must be a digit) is an open file descriptor for a terminal device

L Applies subsequent operators in a multiple-operator test to a symbolic link instead of to the file to which the link points

file is command and filename expanded and then tested to see if it has the specified relationship to the real user. If file does not exist or is inaccessible or, for the operators indicated by *, if the specified file type does not exist on the current system, then all inquiries return false (0).

These operators may be combined for conciseness: -xy file is equivalent to -x file && -y file. For example, -fx is true (returns 1) for plain executable files, but not for directories.

L may be used in a multiple-operator test to apply subsequent operators to a symbolic link instead of to the file to which the link points. For example, -lLo is true for links owned by the invoking user. Lr, Lw, and Lx are always ture for links and false for non-links. L has a different meaning when it is the last operator in a multiple-operator test.

It is possible but not useful, and sometimes misleading, to combine operators which expect file to be a file with operators which do not (for example, X and t). Following L witha non-file operator can lead to particularly strange results.

Other operators return other information, i.e. not just `0` or `1`. They have the same format as before where op may be one of:

A Last file access time, as the number of seconds since epoch

A: Like **A**, but in timestamp format, that is, 'Fri May 14 16:36:10 1993'

M Last file modification time

M: Like **M**, but in timestamp format

C Last inode modification time

C: Like **C**, but in timestamp format

D Device number

I Inode number

F Composite file identifier, in the form **device : inode**

L The name of the file pointed to by a symbolic link

N Number of (hard) links

P Permissions, in octal, without leading zero

P: Like **P**, with leading zero

P mode
Equivalent to -P mode & file, that is, -P22 file returns `22` if file is writable by group and other, `20` if by group only, and `0` if by neither.

P mode:
Like **P mode**, with leading zero

U Numeric userid

U: Username, or the numeric userid if the username is unknown

G Numeric groupid

G: Groupname, or the numeric groupid if the groupname is unknown

Z Size in bytes

Only one of these operators may appear in a multiple-operator test, and it must be the last. **L** has a different meaning at the end of and elsewhere in a multiple-operator test. Because `0` is a valid return value for many of these operators, they do not return `0` when they fail: most return `-1`, and **F** returns : (colon).

File inquiry operators can also be evaluated with the **filetest** built-in command.

Jobs

The shell associates a job with each pipeline. It keeps a table of current jobs, printed by the jobs command, and assigns them small integer numbers. When a job is started asynchronously with & (ampersand), the shell prints a line which looks like

```
[1] 1234
```

indicating that the job which was started asynchronously was job number 1 and had one (top-level) process, whose process id was 1234.

If you are running a job and wish to do something else you may hit the suspend key (usually ^Z), which sends a STOP signal to the current job. The shell will then normally indicate that the job has been 'Suspended' and print another prompt. If the **listjobs** shell variable is set, all jobs will be listed like the **jobs** built-in command; if it is set to 'long' the listing will be in long format, like **jobs -l**. You can then manipulate the state of the suspended job. You can put it in the background with the **bg** command or run some other commands and eventually bring the job back into the foreground with **fg**. (See also the **run-fg-editor** editor command.) A ^Z takes effect immediately and is like an interrupt in that pending output and unread input are discarded when it is typed. The **wait** built-in command causes the shell to wait for all background jobs to complete.

The ^] key sends a delayed suspend signal, which does not generate a STOP signal until a program attempts to read it, to the current job. This can usefully be typed ahead when you have prepared some commands for a job which you wish to stop after it has read them. The ^Y key performs this function in csh; in tcsh , ^Y is an editing command.

A job being run in the background stops if it tries to read from the terminal. Background jobs are normally allowed to produce output, but this can be disabled by giving the command stty tostop. If you set the stty option, then background jobs will stop when they try to produce output like they do when they try to read input.

There are several ways to refer to jobs in the shell. The character % introduces a job name. If you wish to refer to job number 1, you can name it as %1. Just naming a job brings it to the foreground; thus %' is a synonym for **fg** *%1*, bringing job 1 back into the foreground. Similarly, saying %1 & resumes job 1 in the background, just like **bg** *%1*. A job can also be named by an unambigous prefix of the string typed in to start it: %ex would normally restart a suspended 'ex' job, if there were only one suspended job whose name began with the string 'ex'. It is also possible to say %? string to specify a job whose text contains string , if there is only one such job.

The shell maintains a notion of the current and previous jobs. In output pertaining to jobs, the current job is marked with a + (plus) and the previous job with a - (hyphen). The abbreviations %+, %, and (by analogy with the syntax of the history mechanism) %% all refer to the current job, and %- refers to the previous job.

The job control mechanism requires that the stty option *new* be set on some systems. It is an artifact from a *new* implementation of the tty driver which allows generation of interrupt characters from the keyboard to tell jobs to stop. See stty and the **setty** tcsh built-in command for details on setting options in the new tty driver.

Status Reporting

The tcsh shell learns immediately whenever a process changes state. It normally informs you whenever a job becomes blocked so that no further progress is possible, but only just before it prints a prompt. This is done so that it does not otherwise disturb your work. If, however, you set the shell variable **notify**, the shell will notify you immediately of changes of status in background jobs. There is also a shell command **notify** which marks a single process so that its status changes will

be immediately reported. By default notify marks the current process; simply say 'notify' after starting a background job to mark it.

When you try to leave the shell while jobs are stopped, you will be warned that 'You have stopped jobs.' You may use the **jobs** command to see what they are. If you do this or immediately try to exit again, the shell will not warn you a second time, and the suspended jobs will be terminated.

Automatic, Periodic and Timed Events

There are various ways to run commands and take other actions automatically at various times in the **life cycle** of the shell.

- The **sched** built-in command puts commands in a scheduled-event list, to be executed by the shell at a given time.
- The **beepcmd**, **cwdcmd**, **periodic** and **precmd** special aliases can be set, respectively, to execute commands when the shell wants to ring the bell, when the working directory changes, every t-period minutes and before each prompt.
- The **autologout** shell variable can be set to log out of the shell after a given number of minutes of inactivity.
- The **mail** shell variable can be set to check for new mail periodically.
- The **printexitvalue** shell variable can be set to print the exit status of commands which exit with a status other than zero.
- The **rmstar** shell variable can be set to ask the user, when **rm *** is typed, if that is really what was meant.
- The **time** shell variable can be set to execute the **time** built-in command after the completion of any process that takes more than a given number of CPU seconds.
- The **watch** and **who** shell variables can be set to report when selected users log in or out, and the **log** built-in command reports on those users at any time.

National Language System Report

When using the system's NLS, the setlocale function is called to determine appropriate character classification and sorting. This function typically examines the LANG and LC_CTYPE environment variables; refer to the system documentation for further details.

Unknown characters (those that are neither printable nor control characters) are printed in the format *nnn*.

The version shell variable indicates what options were chosen when the shell was compiled. Note also the **newgrp** built-in and **echo_style** shell variable and the locations of the shell's input files (see “tcsh Files” on page 599).

The tcsh shell currently does not support 3 locales. They are IBM-1388 (Chinese), IBM-933 (Korean) and IBM-937 (Traditional Chinese).

Signal Handling

Login shells ignore interrupts when reading the file ~/.logout. The shell ignores quit signals unless started with **-q**. Login shells catch the terminate signal, but non-login shells inherit the terminate behavior from their parents. Other signals have the values which the shell inherited from its parent.

In shell scripts, the shell's handling of interrupt and terminate signals can be controlled with **onintr**, and its handling of hangups can be controlled with **hup** and **nohup**.

The shell exits on a hangup (see also the **logout** shell variable). By default, the shell's children do too, but the shell does not send them a hangup when it exits. **hup** arranges for the shell to send a hangup to a child when it exits, and **nohup** sets a child to ignore hangups.

Terminal Management

The shell uses three different sets of terminal (tty) modes: **edit**, used when editing, **quote**, used when quoting literal characters, and **execute**, used when executing commands. The shell holds some settings in each mode constant, so commands which leave the tty in a confused state do not interfere with the shell. The shell also matches changes in the speed and padding of the tty. The list of tty modes that are kept constant can be examined and modified with the **setty** built-in. Although the editor uses CBREAK mode (or its equivalent), it takes typed-ahead characters anyway.

The **echotc**, **settc** and **telltc** commands can be used to manipulate and debug terminal capabilities from the command line.

The tcsh shell adapts to window resizing automatically and adjusts the environment variables LINES and COLUMNS if set.

tcsh Built-in Commands

The list below contains **tcsh** built-in commands which are not **/bin/sh** built-ins. Descriptions for these commands are found at the end of this chapter.

%	**filetest**	**popd**	**uncomplete**
alloc	**glob**	**pushd**	**unhash**
bindkey	**hashstat**	**rehash**	**unlimit**
breaksw	**hup**	**repeat**	**unsetenv**
builtins	**limit**	**sched**	**watchlog**
bye	**login**	**setenv**	**where**
chdir	**logout**	**settc**	**which**
complete	**ls-F**	**setty**	
dirs	**notify**	**source**	
echotc	**onintr**	**telltc**	

Other tcsh built-in commands are also found in the OS/390 shell. In some cases, they may differ in function; see the specific command description for a discussion of the tcsh version of the command.

: (colon)	**cd**	**fg**	**nice**	**stop**	**unset**
@ (at)	**echo**	**history**	**nohup**	**suspend**	**wait**
alias	**eval**	**jobs**	**printenv**	**time**	
bg	**exec**	**kill**	**set**	**umask**	
break	**exit**	**newgrp**	**shift**	**unalias**	

As well as built-in commands, the tcsh shell has a set of special aliases:

beepcmd	**periodic**	**shell**
cwdcmd	**precmd**	

If set, each of these aliases executes automatically at the indicated time. They are initially undefined. For more information about aliases, see “Alias Substitution” on page 571.

Descriptions of these aliases are as follows:

beepcmd
Runs when the shell wants to ring the terminal bell.

cwdcmd
Runs after every change of working directory. For example, if the user is working on an X window system using xterm and a re-parenting window manager that supports title bars such as twm and does

```
> alias cwdcmd 'echo -n "^[]2;${HOST}:$cwd ^G"'
```

then the shell will change the title of the running xterm to be the name of the host, a colon, and the full current working directory. A fancier way to do that is

```
> alias cwdcmd 'echo -n "^[]2;${HOST}:$cwd^G^[]1;${HOST}^G"'
```

This will put the hostname and working directory on the title bar but only the hostname in the icon manager menu. Putting a **cd, pushd** or **popd** in **cwdcmd** may cause an infinite loop.

periodic
Runs every **tperiod** minutes. This provides a convenient means for checking on common but infrequent changes such as new mail. For example, if one does

```
> set tperiod = 30
> alias periodic checknews
```

then the **checknews** program runs every 30 minutes. If **periodic** is set but **tperiod** is unset or set to 0, **periodic** behaves like **precmd**.

precmd
Runs just before each prompt is printed. For example, if one does

```
> alias precmd date
```

then **date** runs just before the shell prompts for each command. There are no limits on what **precmd** can be set to do, but discretion should be used.

shell
Specifies the interpreter for executable scripts which do not themselves specify an interpreter. The first word should be a full pathname to the desired interpreter. For example: **/bin/tcsh** or **/usr/local/bin/tcsh** (by default, this is set to **/bin/tcsh**).

tcsh Programming Constructs

1. **breaksw**

   Causes a break from a **switch**, resuming after the **endsw**.

2. **case** *label*

   A label in a switch. See the **switch** built-in description.

3. **continue**

   Continues execution of the nearest enclosing **while** or **foreach**. The rest of the commands on the current line are executed.

4. **default**

   Labels the default **case** in a **switch** statement. It should come after all **case** labels.

5. **else**
   end
   endif
   endsw

   See the description of the **foreach, if, switch**, and **while** statements that follow.

6. **goto** *word*

   With **goto**, *word* is filename and command substituted to yield a string of the form label. The tcsh shell rewinds its input as much as possible, searches for a line of the form *label*, possible preceded by blanks or tabs, and continues exectution after that line.

7. **foreach**
   ...
   end

   Successively sets the variable name to each member of wordlist and executes the sequence of commands between this command and the matching end. (Both foreach and end must appear alone on separate lines.) The built-in command **continue** may be used to continue the loop prematurely and the built-in command **break** to terminate it prematurely. When this command is read from the terminal, the loop is read once prompting with **foreach?** (or **prompt2**) before any statements in the loop are executed. If you make a mistake typing in a loop at the terminal you can rub it out.

8. **if** (*expr*) **then**
   ...
   else if (*expr2*) **then**
   ...
   else
   ...
   endif

If the specified *expr* is true then the commands to the first **else** are executed; otherwise if *expr2* is true then the commands to the second **else** are executed. Any number of **else-if** pairs are possible; only one **endif** is needed. The **else** part is optional. (The words **else** and **endif** must appear at the beginning of input lines; the **if** must appear alone on its input line or after an **else**.)

9.

switch (*string*)
case *str1*:
...
breaksw
...
default
...
breaksw
endsw

Each case label is successively matched, against the specified *string* which is first command and filename expanded. The file metacharacters *, ? and [...] may be used in the **case** labels, which are variable expanded. If none of the labels match before a **default** label is found, then the execution begins after the **default** label. Each **case** label and the **default** label must appear at the beginning of a line. The command **breaksw** causes execution to continue after the **endsw**. Otherwise control may fall through **case** labels and default labels as in C. If no label matches and there is no default, execution continues after the **endsw**.

10.

while (*expr*)
...
end

Executes the commands between the **while** and the matching **end** while *expr* (expression) evaluates non-zero. **while** and **end** must appear alone on their input lines. **break** and **continue** may be used to terminate or continue the loop prematurely. If the input is a terminal, the user is prompted the first time through the loop as with **foreach**.

tcsh Shell and Environment Variables

The variables described in this section have special meaning to the tcsh shell. The tcsh shell sets **addsuffix**, **argv**, **autologout**, **command**, **echo_style**, **edit**, **gid**, **group**, **home**, **loginsh**, **path**, **prompt**, **prompt2**, **prompt3**, **shell**, **shlvl**, **tcsh**, **term**, **tty**, **uid**, **user**, and **version** at startup. They do not change thereafter, unless changed by the user. The tcsh shell updates **cwd**, **dirstack**, **owd**, and **status** when necessary, and sets **logout** on logout.

The shell synchronizes **group**, **home**, **path**, **shlvl**, **term**, and **user** with the environment variables of the same names: whenever the environment variable changes the shell changes the correpsonding shell variable to match (unless the shell variable is read-only) and vice versa. Although **cwd** and PWD have identical meanings, they are not synchronized in this manner.

The shell automatically interconverts the different formats of path and PATH.

Figure 15 (Page 1 of 8). tcsh Built-in Shell Variables

Variable	Purpose
addsuffix	If set, filename completion adds / to the end of directories and a space to the end of normal files.
ampm	This variable gives a user the ability to alter the time format in their tcsh prompt. Specifically, **ampm** will override the %T and %P formatting sequences in a user's prompt. If set, all times are shown in 12hour AM/PM format.
argv	The arguments to the shell. Positional parameters are taken from argv. For example, $1 is replaced by $argv. Set by default, but usually empty in interactive shells.
autocorrect	If set, the **spell-word** editor command is invoked automatically before each completion. (This variable is not implemented.)
autoexpand	If set, the **expand-history** editor command is invoked automatically before each completion attempt.
autolist	If set, possibilities are listed after an ambiguous completion. If set to *ambiguous*, possibilites are listed only when no new characters are added by completion.
autologout	Set to the number of minutes of inactivity before automatic logout. Automatic locking is an unsupported feature on the OS/390 platform. If you specify a second parameter on the **autologout** statement (intending it to be for autolock), this parameter will be assigned to **autologout**. When the shell automatically logs out, it prints 'autologout', sets the variable logout to *automatic* and exits. Set to `60` (automatic logout after 60 minutes) by default in login and superuser shells, but not if the shell thinks it is running under a window system (the DISPLAY environment variable is set), or the tty is a pseudo-tty (pty). See also the **logout** shell variable.
backslash_ quote	If set, backslashes (\) always quote \, ' (single quote) and " (double quote). This may make complex quoting tasks easier, but it can cause syntax errors in csh scripts.
cdpath	A list of directories in which **cd** should search for subdirectories if they aren't found in the current directory.
command	If set, the command which was passed to the shell with the **-c** flag.
complete	If set to **enhance**, completion first ignores case and then considers periods, hyphens and underscores ('.', '-' and '_') to be word separators and hyphens and underscores to be equivalent.
correct	If set to *cmd*, commands are automatically spelling-corrected. If set to *complete*, commands are automatically completed. If set to *all*, the entire command line is corrected.
cwd	The full pathname of the current directory. See also the **dirstack** and **owd** shell variables.
dextract	If set, **pushd** *+n* extracts the *n*th directory from the directory stack instead of rotating it to the top.
dirsfile	The default location in which **dirs -S** and **dirs -L** look for a history file. If unset, **~/.cshdirs** is used. Because only **~/.tcshrc** is normally sourced before **~/.cshdirs**, dirsfile should be set in **~/.tcshrc** instead of **~/.login**. For example: `set dirsfile = ~/.cshdirs`

Figure 15 (Page 2 of 8). tcsh Built-in Shell Variables

Variable	Purpose
dirstack	An array of all the directories on the directory stack. $dirstack[1] is the current working directory, $dirstack[2] the first directory on the stack, etc. Note that the current working directory is $dirstack[1] but =0 in directory stack substitutions, etc. One can change the stack arbitrarily by setting **dirstack**, but the first element (the current working directory) is always correct. See also the **cwd** and **owd** shell variables.
dunique	If set, **pushd** removes any instances of *name* from the stack before pushing it onto the stack.
echo	If set, each command with its arguments is echoed just before it is executed. For non-built-in commands all expansions occur before echoing. Built-in commands are echoed before command and filename substitution, since these substitutions are then done selectively. Set by the **-x** command line option.

echo_style	The style of the **echo** built-in. May be set to:
	bsd Don't echo a newline if the first argument is **-n**.
	sysv Recognize backslashed escape sequences in echo strings.
	both Recognizes both the **-n** flag and backslashed escape sequences; the default.
	none Recognize neither.
	Set to *both* by default to the local system default.
	The following is an example of this variable's use:
	`> echo $echo_style`
	`bsd`
	`> echo "\n"`
	`\n`
	`> echo -n "test"`
	`test>`
	`> set echo_style=sysv`
	`> echo $echo_style`
	`sysv`
	`> echo "\n"`
	`> echo -n "test"`
	`-n test`
	`> set echo_style=both`
	`> echo $echo_style`
	`both`
	`> echo -n "test"`
	`test> echo "\n"`
	`>set echo_style=none`
	`> echo $echo_style`
	`none`
	`> echo -n "test"`
	`-n test`
	`> echo "\n"`
	`\n`
	`>`

edit	If set, the command-line editor is used. Set by default in interactive shells.

Figure 15 (Page 3 of 8). tcsh Built-in Shell Variables

Variable	Purpose
ellipsis	If set, the %c'/'%. and %C prompt sequences (see the **prompt** shell variable) indicate skipped directories with an ellipsis (...) instead of /.
fignore	Lists file name suffixes to be ignored by completion.
filec	In the tcsh shell, completion is always used and this variable is ignored.
gid	The user's real group ID.
group	The user's group name.
histchars	A string value determining the characters used in history substitution. The first character of its value is used as the history substitution character, replacing the default character ! (exclamation point). The second character of its value replaces the character ^ (caret) in quick substitutions.
histdup	Controls handling of duplicate entries in the history list. If set to *all* only unique history events are entered in the history list. If set to *prev* and the last history event is the same as the current command, then the current command is not entered in the history. If set to *erase* and the same event is found in the history list, that old event gets erased and the current one gets inserted. The *prev* and *all* options renumber history events so there are no gaps.
histfile	The default location in which **history -S** and **history -L** look for a history file. If unset, *~/.history* is used. **histfile** is useful when sharing the same home directory between different machines, or when saving separate histories on different terminals. Because only *~/.tcshrc* is normally sourced before *~/.history*, **histfile** should be set in *~/.tcshrc* instead of *~/.login*. An example: `set histfile = ~/.history`
histlit	If set, built-in and editor commands and the savehist mechanism use the literal (unexpanded) form of lines in the history list. See also the **toggle-literal-history** editor command.
history	The first word indicates the number of history events to save. The optional second word indicates the format in which history is printed; if not given, %h\t%T\t%R\n is used. The format sequences are described below under **prompt**. (Note that %R has a variable meaning). Set to `100` by default.
home	Initialized to the home directory of the invoker. The filename expansion of ~ refers to this variable.
ignoreeof	If set to the empty string or 0 and the input device is a terminal, the **end-of-file** command (usually generated by the user by typing ^D on an empty line) causes the shell to print 'Use "logout" to leave tcsh.' instead of exiting. This prevents the shell from accidentally being killed. If set to a number *n*, the shell ignores n - 1 consecutive end-of-files and exits on the *n*th. If unset, 1 is used. That is, the shell exits on a single ^D.
implicitcd	If set, the shell treats a directory name typed as a command as though it were a request to change to that directory. If set to verbose, the change of directory is echoed to the standard output. This behavior is inhibited in non-interactive shell scripts, or for command strings with more than one word. Changing directory takes precedence over executing a like-named command, but it is done after alias substitutions. Tilde and variable expansions work as expected.

Figure 15 (Page 4 of 8). tcsh Built-in Shell Variables

Variable	Purpose
inputmode	If set to *insert* or *overwrite*, puts the editor into that input mode at the beginning of each line.
listflags	If set to *x, a* or *A*, or any combination thereof (for example, *xA*), they are used as flags to **ls-F**, making it act like **ls -xF**, **ls -Fa**, **ls -FA** or a combination (for example, **ls -FxA**): *a* shows all files (even if they start with a '.'), *A* shows all files but '.' and '..', and *x* sorts across instead of down. If the second word of listflags is set, it is used as the path to ls(1).
listjobs	If set, all jobs are listed when a job is suspended. If set to *long*, the listing is in long format.
listlinks	If set, the **ls-F** built-in command shows the type of file to which each symbolic link points. For an example of its use, see “ls-F built-in command for tcsh: List files” on page 613.
listmax	The maximum number of items which the **list-choices** editor ocmmand will list without asking first.
listmaxrows	The maximum number of rows of items which the list-choices editor command will list without asking first.
loginsh	Set by the shell if is a login shell. Setting or unsetting it within a shell has no effect. See also **shlvl**.
logout	Set by the shell to *normal* before a normal logout, *automatic* before an automatic logout, and *hangup* if the shell was killed by a hangup signal (see “Signal Handling” on page 583). See also the **autologout** shell variable.
mail	The names of the files or directories to check for incoming mail, separated by whitespace, and optionally preceeded by a numeric word. Before each prompt, if 10 minutes have passed since the last check, the shell checks each file and says 'You have new mail.' (or, if mail contains multiple files, 'You have new mail in name.') if the filesize is greater than zero in size and has a modification time greater than its access time. If you are in a login shell, then no mail file is reported unless it has been modified after the time the shell has started up, in order to prevent redundant notifications. Most login programs will tell you whether or not you have mail when you log in. If a file specified in mail is a directory, the shell will count each file within that directory as a separate message, and will report 'You have n mails.' or 'You have n mails in name.' as appropriate. This functionality is provided primarily for those systems which store mail in this manner, such as the Andrew Mail System. If the first word of mail is numeric it is taken as a different mail checking interval, in seconds. Under very rare circumstances, the shell may report 'You have mail.' instead of 'You have new mail.'
matchbeep	If set to *never*, completion never beeps. If set to *nomatch*, it beeps only when there is no match. If set to *ambiguous*, it beeps when there are multiple matches. If set to *notunique*, it beeps when there is one exact and other longer matches. If unset, *ambiguous* is used.
nobeep	If set, beeping is completely disabled.
noclobber	If set, restrictions are placed on output redirection to insure that files are not accidentally destroyed and that >> redirections refer to existing files, as described in “Input or Output” on page 577.

Figure 15 (Page 5 of 8). tcsh Built-in Shell Variables

Variable	Purpose
noglob	If set, filename substitution and directory stack substitution are inhibited. This is most useful in shell scripts which do not deal with filenames, or after a list of filenames has been obtained and further expansions are not desirable.
nokanji	If set and the shell supports Kanji (see the **version** shell variable), it is disabled so that the meta key can be used.
nonomatch	If set, a filename substitution or directory stack substitution which does not match any existing files is left untouched instead of causing an error. It is still an error for the substitution to be malformed, that is, **echo [** still gives an error.
nostat	A list of directories (or glob-patterns which match directories; see "Filename Substitution" on page 574) that should not be stat(2)ed during a completion operation. This is usually used to exclude directories which take too much time to stat(2), for example */afs*.
notify	If set, the shell announces job completions asynchronously. The default is to present job completions just before printing a prompt.
owd	The old working directory, equivalent to the - (hyphen) used by **cd** and **pushd**. See also the **cwd** and **dirstack** shell variables.
path	A list of directories in which to look for executable commands. A null word specifies the current directory. If there is no path variable then only full pathnames will execute. **path** is set by the shell at startup from the **PATH** environment variable or, if **PATH** does not exist, to a system-dependent default something like (/usr/local/bin /usr/bsd /bin /usr/bin .). The shell may put '.' first or last in path or omit it entirely depending on how it was compiled; see the **version** shell variable. A shell which is given neither the **-c** nor the **-t** option hashes the contents of the directories in path after reading *~/.tcshrc* and each time path is reset. If one adds a new command to a directory in path while the shell is active, one may need to do a rehash for the shell to find it.
printexit-value	If set and an interactive program exits with a non-zero status, the shell prints 'Exit status'.
prompt2	The string with which to prompt in while and foreach loops and after lines ending in \ (backslash). The same format sequences may be used as in prompt (note the variable meaning of %R). Set by default to *%R?* in interactive shells.
prompt3	The string with which to prompt when confirming automatic spelling correction. The same format sequences may be used as in **prompt** (note the variable meaning of *%R*). Set by default to *CORRECT>%R (y|n|e|a)?* in interactive shells.
promptchars	If set (to a two-character string), the %# formatting sequence in the prompt shell variable is replaced with the first character for normal users and the second character for the superuser.
pushdtohome	If set, **pushd** without arguments does **pushd ^**, like **cd**.
pushdsilent	If set, **pushd** and **popd** do not print the directory stack.
recexact	If set, completion completes on an exact match even if a longer match is possible.
recognize_ only_ executables	If set, command listing displays only files in the path that are executable.

Figure 15 (Page 6 of 8). tcsh Built-in Shell Variables

Variable	Purpose
rmstar	If set, the user is prompted before **rm *** is executed.
rprompt	The string to print on the right-hand side of the screen (after the command input) when the prompt is being displayed on the left. It recognises the same formatting characters as **prompt**. It will automatically disappear and reappear as necessary, to ensure that command input isn't obscured, and will only appear if the prompt, command input, and itself will fit together on the first line. If **edit** isn't set, then **rprompt** will be printed after the prompt and before the command input.
savedirs	If set, the shell does **dirs -S** before exiting.
savehist	If set, the shell does **history -S** before exiting. If the first word is set to a number, at most that many lines are saved. (The number must be less than or equal to history.) If the second word is set to **merge**, the history list is merged with the existing history file instead of replacing it (if there is one) and sorted by time stamp and the most recent events are retained. An example: `set savehist = ( 15 merge )`
sched	The format in which the sched built-in command prints scheduled events. If not given, %h\t%T\t%R\n is used. The format sequences are described above under **prompt**; note the variable meaning of *%R*.
shell	The file in which the shell resides. This is used in forking shells to interpret files which have execute bits set, but which are not executable by the system (see “Built-in and Non-Built-in Command Execution” on page 576. Initialized to the (system-dependent) home of the shell.
shlvl	The number of nested shells. Reset to `1` in login shells. See also **loginsh**.
status	The status returned by the last command. If it terminated abnormally, then `0200` is added to the status. tcsh built-in commands which fail return exit status `1`, all other built-in commands return status `0`.
tcsh	The version number of the shell in the format R.VV.PP, where R is the major release number, VV the current version and PP the patchlevel.
term	The terminal type. Usually set in *~/.login* as described under “Options and Invocation” on page 557.
tperiod	The period, in minutes, between executions of the periodic special alias.
tty	The name of the tty, or empty if not attached to one.
uid	The user's login name.
user	The user's login name.
verbose	If set, causes the words of each command to be printed, after history substitution (if any). Set by the **–v** command line option.

Figure 15 (Page 7 of 8). tcsh Built-in Shell Variables

Variable	Purpose
version	The version ID stamp. It contains the shell's version number (see tcsh), origin, release date, vendor, operating system and machine (see VENDOR, OSTYPE, and MACHTYPE environment variables) and a comma-separated list of options which were set at compile time. Options which are set by default in the distribution are noted. **8b** The shell is eight bit clean; default. **7b** The shell is not eight bit clean. **nls** The system's NLS is used; default for systems with NLS. **lf** Login shells execute **/etc/csh.login** before instead of after **/etc/csh.cshrc** and *~/.login* before instead of after **~/.tcshrc** and **~/.history**. **dl** '.' is put last in path for security; default. **nd** '.' is omitted from path for security. **vi** vi-style editing is the default instead of emacs. **dtr** Login shells drop DTR when exiting. **bye** **bye** is a synomym for logout and log is an alternate name for watchlog. **al** **autologout** is enabled; default. **kan** Kanji is used and the ISO character set is ignored, unless the nokanji shell variable is set. **sm** The system's malloc is used. **hb** The #!<program> <args> convention is emulated when executing shell scripts. **ng** The **newgrp** built-in is available. **rh** The shell attempts to set the REMOTEHOST environment variable. **afs** The shell verifies your password with kerberos server if local authentication fails. The **afsuser** shell variable or the AFSUSER environment variable override your local username if set. An administrator may enter additional strings to indicate differences in the local version.
visiblebell	If set, a screen flash is used instead of the audible bell. See **nobeep**. (`Currently not implemented`.)

Figure 15 (Page 8 of 8). tcsh Built-in Shell Variables

Variable	Purpose

watch	A list of user/terminal pairs to watch for logins and logouts. If either the user is *any* all terminals are watched for the given user and vice versa. Setting watch to (*any any*) watches all users and terminals. For example,
	`set watch = (george ttyd 1 any console $user any)`
	reports activity of the user george on ttyd1, any user on the console, and oneself (or a trespasser) on any terminal.
	Logins and logouts are checked every 10 minutes by default, but the first word of watch can be set to a number to check every so many minutes. For example,
	`set watch = (1 any any)`
	reports any login/logout once every minute. For the impatient, the **log** built-in command triggers a watch report at any time. All current logins are reported (as with the **log** built-in) when watch is first set.
	The **who** shell variable controls the format of watch reports.

who	The format string for watch messages. The following sequences are replaced by the given information: **%n** The name of the user who logged in/out. **%a** The observed action, i.e., 'logged on'. 'logged off', or 'replaced olduser on'. **%l** The terminal (tty) on which the user logged in/out. **%M** The full hostname of the remote host, or 'local' if the login/logout was from the local host. **%m** The hostname of the remote host up to the first '.' (period). The full name is printed if it is an IP address or an X Window System display. %M and %m are available only on systems which store the remote hostname in /etc/utmp. If unset, %n has %a %l from %m. is used, or %n has %a %l. on systems which don't store the remote hostname.
wordchars	A list of non-alphanumeric characters to be considered part of a word by the forward-word, back-ward word, etc. editor commands. If unset, *?_-.[]~= is used.

tcsh shell variables not described in the above table are described below:

prompt
The string which is printed before reading each command from the terminal. prompt may include any of the following formatting sequences, which are replaced by the given information:

%/ The current working directory.

%~ The current working directory, but with one's home directory represented by ~ and other users' home directories represented by ~user as per filename substitution. ~user substitution happens only if the shell has already used ~user in a pathname in the current session.

%c[[0]*n*], %.[[0]*n*]
The trailing component of the current working directory, or *n* trailing components if a digit *n* is given. If *n* begins with `0`, the number of

skipped components precede the trailing component(s) in the format **/trailing**. If the ellipsis shell variable is set, skipped components are represented by an ellipsis so the whole becomes **...trailing**. ~ substitution is done as in %~~ above, but the ~ component is ignored when counting trailing components.

%C Like **%c**, but without ^ substitution.

%h, %!, !
The current history event number.

%M The full hostname.

%m The hostname up to the first '.' (period).

%S (%s)
Start (stop) standout mode.

%B (%b)
Start (stop) boldfacing mode.

%U (%u)
Start (stop) underline mode.

%t, %@
The time of day in 12–hour AM/PM format.

%T Like %t, but in 24–hour format (but see the **ampm** shell variable).

%p The precise time of day in 12–hour AM/PM format, with seconds.

%P Like **%p**, but in 24–hour format (but see the **ampm** shell variable).

\c *c* is parsed as in **bindkey**.

^c *c* is parsed as in **bindkey**.

%% A single %.

%n The user name.

%d The weekday in 'Day' format.

%D The day in 'dd' format.

%w The month in 'Mon' format.

%W The month in 'mm' format.

%y The year in 'yy' format.

%Y The year in 'yyyy' format.

%l The tcsh shell's tty.

%L Clears from the end of the prompt to end of the display or the end of the line.

%$ Expands the shell or environment variable name immediately after the $.

%# > (or the first character of the **promptchars** shell variable) for normal users, # (or the second character of **promptchars**) for the superuser.

%{*string%***}**
Includes *string* as a literal escape sequence. It should be used only to change terminal attributes and should not move the cursor location. This cannot be the last sequence in **prompt**.

%? The return code of the command executed just before the prompt.

%R In **prompt2**, the status of the parser. In **prompt3**, the corrected string. In **history**, the history string.

The bold, standout and underline sequences are often used to distinguish a superuser shell. For example,

```
>set prompt = "%m [%h] %B[%@%b [%/] you rang?"
tut [37] [2:54] [/usr/accts/sys] you rang? _
```

Set by default to %# in interactive shells.

symlinks

Can be set to several different values to control symbolic link ('symlink') resolution:

- If set to *chase*, whenever the current directory changes to a directory containing a symbolic link, it is expanded to the real name of the directory to which the link points. This does not work for the user's home directory.
- If set to *ignore*, the shell tries to construct a current directory relative to the current directory before the link was crossed. This means that cding through a symbolic link and then **cd..'ing** returns one to the original directory. This only affects built-in commands and filename completion.
- If set to *expand*, the shell tries to fix symbolic links by actually expanding arguments which look like pathnames. This affects any command, not just built-ins. Unfortunately, this does not work for hard-to-recognize filenames, such as those embedded in command options. Expansion may be prevented by quoting. While this setting is usually the most convenient, it is sometimes misleading and sometimes confusing when it fails to recognize an argument which should be expanded. A compromise is to use *ignore* and use the editor command normalize-path (bound by default to ^X-n) when necessary.

Some examples are in order. First, let's set up some play directories:

```
> cd /tmp
> mkdir from from/src to
> ln -s from/src to/dist
```

Here's the behavior with **symlinks** unset,

```
> cd /tmp/to/dist; echo $cwd
/tmp/to/dist
> cd ..; echo $cwd
/tmp/from
```

here's the behavior with symlinks set to *chase*,

```
> cd /tmp/to/dst; echo $cwd
/tmp/from/src
> cd ..; echo $cwd
/tmp/from
```

here's the behavior with symlinks set to *ignore*,

```
> cd /tmp/to/dist; echo $cwd
/tmp/to/dst
> cd ..; echo $cwd
/tmp/to
```

and here's the behavior with symlinks set to *expand*.

```
> cd /tmp/to/dist; echo $cwd
/tmp/to/dst
> cd ..; echo $cwd
/tmp/to
> cd /tmp/to/dist; echo $cwd
/tmp/to/dst
> cd ".."; echo $cwd
/tmp/from
> /bin/echo ..
/tmp/to
> /bin/echo ".."
..
```

expand expansion:

1. works just like *ignore* for built-ins like **cd**,
2. is prevented by quoting, and
3. happens before filenames are passed to non-built-in commands.

time

If set to a number, then the **time** built-in command executes automatically after each command which takes more than that many CPU seconds. If there is a second word, it is used as a format string for the output of the time built-in. The following sequences may be used in the format string:

%U The time the process spent in user mode in cpu seconds.

%S The time the process spent in kernel mode in cpu seconds.

%E The elapsed (wall clock) time in seconds.

%P The CPU percentage computed as (%U + %S) / %E.

%W The number of times the process was swapped.

%X The average amount in (shared) text space used in Kbytes.

%D The average amount in (unshared) data/stack space used in Kbytes.

%K The total space used (%X + %D) in Kbytes.

%M The maximum memory the process had in use at any time in Kbytes.

%F The number of major page faults (page needed to be brought from disk).

%R The number of minor page faults.

%I The number of input operations.

%O The number of output operations.

%r The number of socket messages received.

%s The number of socket messages sent.

%k The number of signals received.

%w The number of voluntary context switches (waits).

%c The number of involuntary context switches.

Only the first four sequences are supported on systems without BSD resource limit functions. The default time format is

```
Uu    %Ss %E %P %X+%Dk %I+%Oio    %Fpf+%Ww
```

for systems that support resource usage reporting.

The following table contains a list of tcsh environment variables.

Figure 16. tcsh Environment Variables

Environment Variable	Purpose
COLUMNS	A list of directories in which **cd** should search for subdirectories if they aren't found in the current directory.
DISPLAY	Used by X Window System. If set, the shell does not set **autologout**.
EDITOR	The pathname to a default editor. See also the VISUAL environment variable and the **run-fg-editor** editor command.
GROUP	Equivalent to the **group** shell variable.
HOME	Equivalent to the **home** shell variable..
HOST	Initialized to the name of the machine on which the shell is running, as determined by the gethostname system call.
HOSTTYPE	Initialized to the type of the machine on which the shell is running, as determined at compile time. This variable is obsolete and will be removed in a future version.
HPATH	A colon-separated list of directories in which the run-help editor command looks for command documentation.
LANG	Gives the preferred character environment. See “National Language System Report” on page 582.
LC_CTYPE	If set, only ctype character handling is changed. See “National Language System Report” on page 582.
LINES	The number of lines in the terminal. See “Terminal Management” on page 583.
MACHTYPE	The machine type (microprocessor class or machine model), as determined at compile time.
NOREBIND	If set, printable characters are not rebound to **self-insert-command**. After a user sets NOREBIND, a new shell must be started. See “National Language System Report” on page 582.
OSTYPE	The operating system, as determined at compile time.
PATH	A colon-separated list of directories in which to look for executables. Equivalent to the **path** shell variable, but in a different format..
PWD	Equivalent to the **cwd** shell variable, but not synchronized to it; updated only after an acutal directory change.
REMOTEHOST	The host from which the user has logged in remotely, if this is the case and the shell is able to determine it. (The OS/390 tcsh shell is not currently compiled with REMOTEHOST defined; see the **version** shell variable.)
SHLVL	Equivalent to the **shlvl** shell variable.
TERM	Equivalent to the **term** shell variable.
USER	Equivalent to the **user** shell variable.
VENDOR	The vendor, as determined at compile time.
VISUAL	The pathname to a default full-screen editor. See the EDITOR environment variable and the **run-fg-editor** editor command.

tcsh Files

/etc/csh.cshrc
Read first by every shell.

/etc/csh.login
Read by login shells after /etc/csh.cshrc..

~/.tcshrc
Read by every shell after **/etc/csh.cshrc** or its equivalent.

~/.history
Read by login shells after **~/.tcshrc** if **savehist** is set. See also **histfile**.

~/.login
The shell reads ~/.login after **~/.tcshrc** and **~/.history**. See the **version** shell variable.

~/.cshdirs
Read by login shells after **~/.login** if **savedirs** is set. See also **dirsfile**.

~/.logout
Read by login shells at logout.

/bin/sh
Used to interpret shell scripts not starting with a #.

/tmp/sh*
Temporary file for < <.

tcsh shell: Problems and Limitations

When a suspended command is restarted, the tcsh shell prints the directory it started in if this is different from the current directory. This can be misleading (that is, wrong) as the job may have changed directories internally.

Shell built-in functions are not stoppable/restartable. Command sequences of the form 'a ; b ; c' are also not handled gracefully when stopping is attempted. If you suspend 'b', the tcsh shell will then immediately execute 'c'. This is especially noticeable if this expansion results from an alias. It suffices to place the sequence of commands in ()'s to force it to a subshell, for example, (a ; b ; c).

Control over tty output after processes are started is primitive. In a virtual terminal interface much more interesting things could be done with output control.

Alias substitution is most often used to clumsily simulate shell procedures; shell procedures should be provided instead of aliases.

Commands within loops are not placed in the history list. Control structures should be parsed instead of being recognized as built-in commands. This would allow control commands to be placed anywhere, to be combined with |, and to be used with & and ; (semi-colon) metasyntax.

foreach doesn't ignore here documents when looking for its end.

It should be possible to use the : (colon) modifiers on the output of command substitutions.

The screen update for lines longer than the screen width is very poor if the terminal cannot move the cursor up (terminal type 'dumb').

It is not necessary for HPATH and NOREBIND to be environment variables.

Glob-patterns which do not use '?', '*' or '[]' or which use '{}' or '~' are not negated correctly.

The single-command form of **if** does output redirection even if the expression is false and the command is not executed.

ls-F includes file identification characters when sorting filenames and does not handle control characters in filenames well. It cannot be interrupted.

visiblebell shell variable is currently not implemented.

In filename and programmed completion, the 'C' completion rule word list type does not correctly select completion from the given directory.

There are three locales (codepages) which the tcsh shell will not correctly support: IBM-1388 (Chinese), IBM-933 (Korean) and IBM-937 (Traditional Chinese).

If you want to help maintain and test tcsh, send mail to listserv@mx.gw.com with the text 'subscribe tcsh '.

Limitations

Some limitations of the tcsh shell are:

- Words can be no longer than 1024 characters.
- The system limits argument lists to 10240 characters.
- The number of arguments to a command which involves filename expansion is limited to 1/6th the number of characters allowed in an argument list.
- Command substitutions may substitute no more characters than are allowed in an argument list.
- To detect looping, the shell restricts the number of alias substitutions on a single line to 20.

Related Information

: (colon), **@ (at)**, **alias**, **bg**, **break**, **cd**, **continue**, **echo**, **eval**, **exec**, **exit**, **fg**, **history**, **jobs**, **kill**, **newgrp**, **nice**, **nohup**, **printenv**, **set**, **shift**, **stop**, **suspend**, **time**, **umask**, **unalias**, **unset**, **wait**

% (percent) built-in command for tcsh: Move jobs to the foreground or background

Format

% [*job*] [*&*]

Description

%, is a synonym for the **fg** built-in command.

- **%** (percent) without arguments will bring the current job to the foreground.
- **%** specified with a job number attempts to bring that particular job to the foreground.
- **%** *job &* will move the specified job to the background. This syntax works the same as the **bg** built-in command. If no job is specified, the current job is moved to the background.

  Note: Current jobs will have a **+** next to the status column in jobs command output. See “Jobs” on page 580.

Related Information

jobs, **tcsh**

---

alloc built-in command for tcsh: Show the amount of dynamic memory acquired

Format

alloc *argument*

Description

Shows the amount of dynamic memory acquired, broken down into used and free memory. **alloc**used with an argument, shows the number of free and used blocks in each size category. The categories start at size 8 and double at each step.

Note: **alloc** is supported, but the output is not meaningful on OS/390.

Related Information

tcsh

---

bindkey built-in command for tcsh: List all bound keys

Format

bindkey [-l|-d|-e|-v|-u]
bindkey [-a] [-b] [-k] [-r] [– –] *key*
bindkey [-a] [-b] [-k] [-c|-s] [– –] *key command*

Description

bindkey specified alone (without options, *key*, or *key command*) lists all bound keys and the editor command to which each is bound.

bindkey specified with *key* (with or without options) lists the editor command to which key is bound.

bindkey specified with *key command* (with or without options) binds the editor *command* to *key*.

Options

–l Lists all commands and a short description of each.

–d Binds all keys to the standard bindings for the default editor.

–e Binds all keys to the standard GNU Emacs-like bindings.

–v Binds all keys to the standard vi-like bindings.

–a Lists or changes key-bindings in the alternative key map. This is the key map used in vi command mode.

–b *key* is interpreted as a control character written ^*character* ('^A') or C-*character* ('C-A'), a meta character written M-character ('M-A'), or an extended prefix key written X-character ('X-A').

–k *key* is interpreted as a symbolic arrow key name, which may be one of 'down', 'up', 'left' or 'right'.

–r Removes key's binding. Be careful: **bindkey -r** does not bind key to **self-insert-command**, it unbinds *key* completely.

–c *command* is interpreted as a built-in or external command instead of an editor command.

–s *command* is taken as a literal string and treated as terminal input when *key* is typed. Bound keys in *command* are themselves reinterpreted, and this continues for ten levels of interpretation.

– – Forces a break from option processing, so the next word is taken as *key* even if it begins with '-'.

Usage Notes

1. *key* may be a single character or a string. If a command is bound to a string, the first character of the string is bound to **sequence-lead-in** and the entire string is bound to the command.
2. Control characters in *key* can be literal (they can be typed by preceding them with the editor command **quoted-insert**, normally bound to '^V') or written caret-character style, for example, '^A'. Delete is written '^?' (caret-question mark). *key* and *command* can contain backslashed escape sequences (in the style of System V echo) as follows:

\a Bell

\b Backspace

\e Escape

\f Form feed

\n Newline

\r Carriage return

\t Horizontal tab

\v Vertical tab

*****nnn* The EBCDIC character corresponding to the octal number nnn

'\' nullifies the special meaning of the following characters, notably '\' and '^'.

Related Information

tcsh

---

builtins built-in command for tcsh: Prints the names of all built-in commands

Format

builtins

Description

Prints the names of all built-in commands.

Related Information

tcsh

---

bye built-in command for tcsh: Terminate the login shell

Format

bye

Description

A synonym for the **logout** built-in command. (See the **version** shell variable.)

Related Information

logout

---

chdir built-in shell command for tcsh: Change the working directory

Format

chdir

Description

A synonym for the **cd** built-in command.

Related Information

cd, **tcsh**

---

complete built-in command for tcsh: List completions

Format

complete[*command* [*word*/*pattern*/*list*[*:select*]/[[*suffix*]/] ...]]

Description

complete, without arguments, lists all completions. With *command*, **complete** lists completions for command. With *command* and *word* etc., **complete** defines completions.

Arguments

command

command may be a full command name or a glob-pattern. See "Filename Substitution" on page 574. It can begin with – to indicate that completion should be used only when *command* is ambiguous.

word

word specifies which word relative to the current word is to be completed, and may be one of the following:

- **c** Current-word completion. *pattern* is a glob-pattern which must match the beginning of the current word on the command line. *pattern* is ignored when completing the current word.
- **C** Like **c**, but includes *pattern* when completing the current word.
- **n** Next-word completion. *pattern* is a glob-pattern which must match the beginning of the previous word on the command line.
- **N** Like **n**, but must match the beginning of the word two before the current word.
- **p** Position-dependent completion. *pattern* is a numeric range, with the same syntax used to index shell variables, which must include the current word.

list The list of possible completions, which may be one of the following:

- **a** Aliases
- **b** Bindings (editor commands)
- **d** Directories
- **D** Directories which begin with the supplied path prefix
- **e** Environment variables
- **f** Filenames
- **F** Filenames which begin with the supplied path prefix
- **g** Groupnames
- **j** Jobs
- **l** Limits
- **n** Nothing
- **s** Shell variables
- **S** Signals
- **t** Plain (text) files

T	Plain (text) files which begin with the supplied path prefix
v	Any variables
u	Usernames
x	Like **n**, but prints select when **list-choices** is used
X	Completions
$var	Words from the variable **var**
(...)	Words from the given list
...	Words from the output of command

select

select is an optional glob-pattern. If given, only words from *list* which match select are considered and the **fignor**e shell variable is ignored. The last three types of completion may not have a select pattern, and **x** uses select as an explanatory message when the **list-choices** editor command is used.

suffix

suffix is a single character to be appended to a successful completion. If null, no character is appended. If omitted (in which case the fourth delimiter can also be omitted), a slash is appended to directories and a space to other words.

Examples

1. Some commands take only directories as arguments, so there is no point in completing plain files. For example:

   ```
   > complete cd 'p/1/d/'
   ```

   completes only the first word following **cd** (*p/1*) with a directory.

2. **p**-type completion can be used to narrow down command completion. For example:

   ```
   > co[^D]
   complete compress
   > complete -co* 'p/0/(compress)/'
   > co[^D]
   > compress
   ```

   This completion completes commands (words in position *0*, *p/0*) which begin with *co* (thus matching *co**) to *compress* (the only word in the list). The leading - indicates that this completion is to be used only with ambiguous commands.

3. This is an example of **n**-type completion. Any word following *find* and immediately following *-user* is completed from the list of users.

   ```
   > complete find 'n/-user/u/'
   ```

4. This demonstrates **c**-type completion. Any word following *cc* and beginning with *-I* is completed as a directory. *-I* is not taken as part of the directory because we used lowercase **c**.

   ```
   > complete cc 'c/-I/d/'
   ```

5. Different *lists* are useful with different commands:

```
> complete alias 'p/1/a/'
> complete man 'p/*/c/'
> complete set 'p/1/s/'
> complete true 'p/1/x:Truth has no options./'
```

These complete words following **alias** with aliases, **man** with commands, and **set** with shell variables. **true** doesn't have any options, so **x** does nothing when completion is attempted and prints 'Truth has no options.' when completion choices are listed.

The **man** example, and several other examples below, could just as well have used *c/** or *n/** as *p/**.

6. Words can be completed from a variable evaluated at completion time,

```
> complete ftp 'p/1/$hostnames/'
> set hostnames = (rtfm.mit.edu tesla.ee.cornell.edu)
> ftp [^D]
rtfm.mit.edu tesla.ee.cornell.edu
> ftp [^C]
> set hostnames = (rtfm.mit.edu tesla.ee.cornell.edu uunet.uu.net)
> ftp [^D]
rtfm.mit.edu tesla.ee.cornell.edu uunet.uu.net
```

or from a command run at completion time:

```
> complete kill 'p/*/'ps | awk \{print\ \$1\}'/'
> kill -9 [^D]
23113 23377 23380 23406 23429 23529 23530 PID
```

The **complete** command does not itself quote its arguments, so the braces, space and $ in {print $1} must be quoted explicitly.

7. One command can have multiple completions:

```
> complete dbx 'p/2/(core)/' 'p/*/c/'
```

This example completes the second argument to **dbx** with the word *core* and all other arguments with commands. The positional completion is specified before the next-word completion. Since completions are evaluated from left to right, if the next-word completion were specified first it would always match and the positional completion would never be executed. This is a common mistake when defining a completion.

8. The select pattern is useful when a command takes only files with particular forms as arguments. For example,

```
> complete cc 'p/*/f:*.[cao]/'
```

completes *cc* arguments only to files ending in **.c**, **.a**, or **.o**. *select* can also exclude files, using negation of a glob-pattern as described under “Filename Substitution” on page 574.

9. One might use

```
> complete rm 'p/*/f:^*.{c,h,cc,C,tex,1,man,l,y}/'
```

to exclude precious source code from **rm** completion. Of course, one could still type excluded names manually or override the completion mechanism using the **complete-word-raw** or **list-choices-raw** editor command.

10. The **D, F** and **T** lists are like **d, f** and **t** respectively, but they use the select argument in a different way: to restrict completion to files beginning with a par-

ticular path prefix. For example, the Elm mail program uses **=** as an abbreviation for one's mail directory. One might use

```
> complete elm c@=@F:$HOME/Mail/@
```

to complete *elm -f =* as if it were *elm -f ~/Mail/*. We used @ instead of / to avoid confusion with the select argument, and we used $HOME instead of ~ because home directory substitution only works at the beginning of a word.

11. *suffix* is used to add a nonstandard suffix (not space or '/' for directories) to completed words. For example,

```
> complete finger 'c/*@/$hostnames/' 'p/1/u/@'
```

completes arguments to *finger* from the list of users, appends an @, and then completes after the @ from the **hostnames** variable. Note the order in which the completions are specified.

12. A more complex example:

```
complete find \
'n/-name/f/' 'n/-newer/f/' 'n/-{,n}cpio/f/' \
'n/-exec/c/' 'n/-ok/c/' 'n/-user/u/' \
'n/-group/g/' 'n/-fstype/(nfs 4.2)/' \
'n/-type/(b c d f l p s)/' \
'c/-/(name newer cpio ncpio exec ok user \
group fstype type atime ctime depth inum \
ls mtime nogroup nouser perm print prune \
size xdev)/' \
'p/*/d/'
```

This completes words following -name, -newer, -cpio or ncpio (note the pattern which matches both) to files, words following -exec or -ok to commands, words following user and group to users and groups respectively and words following -fstype or -type to members of the given lists. It also completes the switches themselves from the given list (note the use of c-type completion) and completes anything not otherwise completed to a directory.

Programmed completions are ignored if the word being completed is a tilde substitution (beginning with ~) or a variable (beginning with $). **complete** is an experimental feature, and the syntax may change in future versions of the shell. See also the **uncomplete** built-in command.

Related Information

tcsh, **uncomplete**

---

dirs built-in command for tcsh: Print the directory stack

Format

dirs [**-l**] [**-n**|**-v**]
dirs -S|**-L** [*filename*]
dirs -c

Description

dirs used alone prints the directory stack in the following format: The top of the stack is at the left and the first directory in the stack is the current directory. For example:

```
 > cd <========== # Change to home dir
 > pushd /bin <== # Change dir to /bin and add /bin to dir stack
 /bin ~
 > pushd /tmp <== # Change dir to /tmp and add /tmp to dir stack
 /tmp /bin ~
 > dirs <======== # Display current dir stack
 /tmp /bin ~
 > dirs -l <===== # Display in expanded (long) format
 /tmp /bin /u/erinf
 > dirs -v <===== # Display in verbose format
 0 /tmp
 1 /bin
 2 ~
 > popd <======== # Change dir back to /bin and remove /tmp from dir stack
 /bin ~
 >pwd
 /bin
Note:  dir=directory
```

Options

–l Output is expanded explicitly to home or the pathname of the home directory for the user.

-n Entries are wrapped before they reach the edge of the screen.

-v Entries are printed one per line, preceded by their stack postions.

If more than one of **-n** or **-v** is given, **-v** takes precedence.

-S Saves the directory stack to filename as a series of **cd** and **pushd** commands.

-L The tcsh shell sources filename, which is presumably a directory stack file saved by the **-S** option or the **savedirs** mechanism. In either case, **dirsfile** is used if filename is not given and **~/.cshdirs** is used if dirsfile is unset.

Login shells do the equivalent of **dirs -L** on startup and, if **savedirs** is set, you should issue **dirs -S** before exiting. Because only **~/.tcshrc** is normally sourced before **~/.cshdirs**, **dirsfile** should be set in **~/.tcshrc** instead of **~/.login**.

–c Clear the directory stack.

Related Information

tcsh

echotc built-in command for tcsh: Exercise the terminal capabilities in args

Format

echotc [**-sv**] **arg** ...

Description

echotc takes advanatage of the terminal capabilities in args. For example, **echotc** cm 3 10 sends it to column 3 and row 10.

If arg is *baud*, *cols*, *lines*, *meta* or *tabs*, **echotc** prints the value of that capability (either yes or no, which indicates that the terminal does or does not have that capability). You might use this to make the output from a shell script less verbose on slow terminals, or limit command output to the number of lines on the screen:

```
> set history=`echotc lines`
> @ history--
```

Termcap strings may contain wildcards which will not echo correctly. One should use double quotes when setting a shell variable to a terminal capability string, as in the following example that places the date in the status line:

```
> set standout=`echotc sõ
> set end_standout=`echotc se`
> echo -n "$standout"; date; echo -n "$end_standout"
Mon Oct 25 10:06:48 EDT 1999
>
```

Note: The date, as indicated above, is printed out in standard output.

The **infocmp** command can be used to print the current terminal description in termcap format (instead of terminfo format).

Options

-s Nonexistent capabilities return the empty string instead of causing an error.

-v Messages are verbose.

Related Information

tcsh

filetest built-in command for tcsh: Apply the op file inquiry operator to a file

Format

filetest *-op file –*

Description

filetest applies *op* (which is a file inquiry operator) to each file and returns the results as a space-separated list. For more information on file inquiry operators, see "File Inquiry Operators" on page 579.

Related Information

tcsh

---

glob built-in command for tcsh: Write each word TO STANDARD OUTPUT

Format

glob *wordlist*

Description

glob is like **echo**, but no \ (backslash) escapes are recognized and words are delimited by null characters in the output. **glob** is useful for programs which wish to use the shell to filename expand a list of words.

Related Information

echo, **tcsh**

---

hashstat built-in command for tcsh: Print a statistic line on hash table effectiveness

Format

hashstat

Description

hashstat prints a statistics line indicating how effective the internal hash table has been at locating commands (and avoiding exec's). An exec is attempted for each component of the path where the hash function indicates a possible hit, and in each component which does not begin with a / (forward slash).

OS/390 systems have a vfork() command, however, tcsh is not compiled to use it. Typically on machines without vfork, **hashstat** prints only the number and size of hash buckets, but on OS/390 systems, a **hashstat** print out would contain this:

```
> hashstat
> hashstat 512 hash buckets of 8 bits each
>
```

Related Information

tcsh

---

hup built-in command for tcsh: Run command so it exits on a hang-up signal

Format

hup [*command*]

Description

With *command*, **hup** runs the command such that it will exit on a hangup signal and arranges for the shell to send it a hangup signal when the shell exits. Commands may set their own response to hangups, overriding **hup**. Without an argument (allowed only in a shell script), **hup** causes the shell to exit on a hangup for the remainder of the script. See “Signal Handling” on page 583.

Related Information

nohup, **tcsh**

---

limit built-in command for tcsh: Limit consumption of processes

Format

limit [**–h**] [*resource* [*maximum-use*]]

Description

limit limits the consumption by the current process and each process it creates to not individually exceed maximum-use on the specified resource. If no *maximum-use* is given, then the current limit is printed; if no *resource* is given, then all limitations are given. If the **-h** flag is given, the hard limits are used instead of the current limits. The hard limits impose a ceiling on the values of the current limits. All hard limits can be raised only by a process which has superuser authority (except for **coredumpsize, vmemoryuse**, and **descriptors**), but a user may lower or raise the current limits within the legal range.

Controllable resources currently include:

cputime
: The maximum number of cpu-seconds to be used by each process.

filesize
: The largest single file which can be created.

datasize
: The maximum growth of the data+stack region via **sbrk** beyond the end of the program text.

stacksize
: The maximum size of the automatically-extended stack region.

coredumpsize
The size of the largest core dump that will be created.

memoryuse
The maximum amount of physical memory a process may have allocated to it at a given time.

maximum-use may be given as a (floating point or integer) number followed by a scale factor. For all limits other than *cputime* the default scale is k or kilobytes (1024 bytes); a scale factor of m or megabytes may also be used. For *cputime* the default scaling is seconds, while m for minutes or h for hours, or a time of the form mm:ss giving minutes and seconds may be used.

For both *resource* names and scale factors, unambiguous prefixes of the names suffice.

Related Information

tcsh, **ulimit**, **unlimit**

Also see **setrlimit()** in *OS/390 C/C++ Run-Time Library Reference*.

---

log built-in command for tcsh: Print the watch tcsh shell variable

Format

log

Description

Prints the **watch** shell variable and reports on each user indicated in **watch** who is logged in, regardless of when a user last logged in.

Note: The OS/390 tcsh shell is compiled to use **watchlog**. If you attempt to use **log** on an OS/390 system, you will get an error that says "Command not found".

Related Information

tcsh, **watchlog**

---

login built-in command for tcsh: Terminate a login shell

Format

login

Description

login terminates a login shell, replacing it with an instance of **/bin/login**. This is one way to log off (included for compatibility with **sh**).

Related Information

logout, **tcsh**

---

logout built-in command for tcsh: Terminate a login shell

Format

logout

Description

logout terminates a login shell. Especially useful if **ignoreeof** is set.

Related Information

login, **tcsh**

---

ls-F built-in command for tcsh: List files

Format

ls-F [*-switch ...*] [*file ...*]

Description

In the tcsh shell, **ls-F** lists files like **ls -F**, but works much faster. It identifies each type of special file in the listing with a special character:

/ Directory

***** Executable

Block device

% Character device

| Named pipe

= Socket

@ Symbolic link

If the **listlinks** shell variable is set, symbolic links are identified in more detail (only, of course, on systems which have them):

@ Symbolic link to a non-directory

> Symbolic link to a directory

& Symbolic link to nowhere

listlinks also slows down **ls-F**.

If you use files which are set-up as follows:

```
#creating a file
   touch file1
#creating a symbolic link to the file
   ln -s file1 link1
#creating a directory
   mkdir dir1
#creating a symbolic link to the directory
   ln -s dir1 linkdir1
#creating a symbolic link to a file that doesn't exist
   ln -s noexist linktonowhere
```

when you issue an **ls-F** with **listlinks** unset, you will get the following output:

```
> ls-F
dir1/ file1 link1@ linkdir1@ linktonowhere@
>
```

with **listlinks** set:

```
> set listlinks
> ls-F
dir1/ file1 link1@ linkdir1> linktonowhere&
>
```

If the **listflags** shell variable is set to x, a or A, or any combination thereof (for example, xA), they are used as flags to **ls-F**, making it act like **ls -xF, ls -Fa, ls -FA** or a combination **ls -Fx**A. On OS/390, **ls -C** is the default. However, on machines where **ls -C** is not the default, **ls-F** acts like **ls -CF**, unless **listflags** contains an x, in which case it acts like **ls -xF**.

See “tcsh — Invoke a C shell” on page 556.

Usage Note

To view an online manual description for the **ls-F** command, you must type **ls-F** without the dash. So, to see the man page you would issue:

```
man lsF
```

Related Information

ls, **tcsh**

---

notify built-in command for tcsh: Notify user of job status changes

Format

notify [*%job ...*]

Description

notify causes the shell to notify the user asynchronously when the status of any of the specified jobs (or, without *%job*, the current job) changes, instead of waiting until the next prompt. *job* may be a number, a string, ", %, + or '-' as described under “Jobs” on page 580. See also the **notify** shell variable.

Related Information

tcsh

---

onintr built-in command for tcsh: Control the action of the tcsh shell on interrupts

Format

onintr [-|*label*]

Description

onintr controls the action of the shell on interrupts. Without arguments, **onintr** restores the default action of the shell on interrupts, which is to terminate shell scripts or to return to the terminal command input level. With '-', causes all interrupts to be ignored. With *label*, causes the shell to execute a **goto** *label* when an interrupt is received or a child process terminates because it was interrupted.

onintr is ignored if the shell is running detached and in system startup files, where interrupts are disabled anyway.

Related Information

goto, **tcsh**

---

popd built-in command for tcsh: Pop the directory stack

Format

popd [**-p**] [**-l**] [**-n**|**-v**] [*+n*]

Description

popd without options, pops the directory stack and returns to the new top directory. With a number *+n*, discards the *n*'th entry in the stack. All forms of **popd** print the final directory stack, just like **dirs**. The **pushdsilent** shell variable can be set to prevent this.

Options

–l Output is expanded explicitly to home or the pathname of the home directory for the user.

-n Entries are wrapped before they reach the edge of the screen.

-p Overrides **pushdsilent**.

-v Entries are printed one per line, preceded by their stack postions.

If more than one of **-n** or **-v** is given, **-v** takes precedence.

Related Information

tcsh

---

pushd built-in command for tcsh: Make exchanges within directory stack

Format

pushd [**-p**] [**-l**] [**-nl-v**] [*name*/ *+n*]

Description

pushd with options, exchanges the top two elements of the directory stack. If **pushdtohome** is set, **pushd** without arguments does *pushd ~*, like **cd**. With *name*, **pushd** pushes the current working directory onto the directory stack and changes to name. If name is '-', it is interpreted as the previous working directory (see "Filename Substitution" on page 574). If **dunique** is set, **pushd** removes any instances of *name* from the stack before pushing it onto the stack. With a number *+n*, **pushd** rotates the *n*'th element of the directory stack around to be the top element and changes to it. If **dextract** is set, however, **pushd** *+n* extracts the *n*'th directory, pushes it onto the top of the stack and changes to it. So, instead of just rotating the entire stack around, **dextract** lets the user have the *n*'th directory extracted from its current position, and pushes it onto the top. For example:

```
> pushd /tmp
/tmp ~
> pushd /bin
/bin /tmp ~
> pushd /u
/u /bin /tmp ~
> pushd /usr
/usr /u /bin /tmp ~
> pushd +2
/bin /tmp ~ /usr /u
> set dextract
> dirs
/bin /tmp ~ /usr /u
> pushd +2
~ /bin /tmp /usr /u
>
```

Finally, all forms of **pushd** print the final directory stack, just like **dirs**. The **pushdsilent** tcsh shell variable can be set to prevent this.

Options

–l Output is expanded explicitly to home or the pathname of the home directory for the user.

-n Entries are wrapped before they reach the edge of the screen.

-p Overrides **pushdsilent**.

-v Entries are printed one per line, preceded by their stack postions.

If more than one of **-n** or **-v** is given, **-v** takes precedence.

Related Information

cd, **tcsh**

---

rehash built-in command for tcsh: Recompute internal hash table

Format

rehash

Description

rehash causes the internal hash table of the contents of the directories in the **path** variable to be recomputed. This is needed if new commands are added to directories in path while you are logged in. This should only be necessary if you add commands to one of your own directories, or if a systems programmer changes the contents of one of the system directories. Also flushes the cache of home directories built by tilde (~) expansion.

Related Information

hashstat, **tcsh**

---

repeat built-in command for tcsh: Execute command count times

Format

repeat *count command*

Description

The specified *command* is executed *count* times. **repeat** is subject to the same restrictions as the command in the one line **if** statement. I/O redirections occur exactly once, even if *count* is 0.

Related Information

tcsh

---

sched built-in command for tcsh: Print scheduled event list

Format

sched
sched *hh:mm command*
sched *n*

Description

sched used alone prints the scheduled-event list. The sched shell variable may be set to define the format in which the scheduled-event list is printed. **sched** *hh:mm command* adds command to the scheduled-event list. For example:

```
 >sched 11:00 echo It\'s eleven o\'clock.
```

causes the shell to echo 'It's eleven o'clock.' at 11 AM. The time may be in 12-hour AM/PM format

```
>sched 5pm set prompt='[%h] It\'s after 5; go home: >'
```

or may be relative to the current time:

```
>sched +2:15 /usr/lib/uucp/uucico -r1 -sother
```

A relative time specification may not use AM/PM format. The third form removes item *n* from the event list:

```
> sched
1 Wed Apr 4 15:42 /usr/lib/uucp/uucico -r1 -sother
2 Wed Apr 4 17:00 set prompt=[%h] It's after 5; go home: >
> sched -2
> sched
1 Wed Apr 4 15:42 /usr/lib/uucp/uucico -r1 -sother
```

A command in the scheduled-event list is executed just before the first prompt is printed after the time when the command is scheduled. It is possible to miss the exact time when the command is to be run, but an overdue command will execute at the next prompt. A command which comes due while the shell is waiting for user input is executed immediately. However, normal operation of an already-running command will not be interrupted so that a scheduled-event list element may be run.

This mechanism is similar to, but not the same as, the **at** command on some UNIX systems. Its major disadvantage is that it may not run a command at exactly the specified time. Its major advantage is that because **sched** runs directly from the shell, it has access to shell variables and other structures. This provides a mechanism for changing one's working environment based on the time of day.

Related Information

tcsh

---

setenv built-in command for tcsh: Set environment variable name to value

Format

setenv [*name* [*value*]]

Description

setenv without arguments, prints the names and values of all environment variables. Given *name*, sets the environment variable name to *value* or, without *value*, to the null string.

Related Information

tcsh

---

settc built-in command for tcsh: Tell tcsh shell the terminal capability cap value

Format

settc *cap value*

Description

settc tells the tcsh shell to believe that the terminal capability cap (as defined in **termcap**) has the value *value*. No sanity checking is done. Concept terminal users may have to **settc** *xn no* to get proper wrapping at the rightmost column.

Related Information

tcsh

---

setty built-in command for tcsh: Control tty mode changes

Format

setty [**-d**|**-q**|**-x**] [**-a**] [+|-]*mode*]

Description

setty controls which tty modes (see the **stty** command description which contains lists of mode operands, such as **echoe** and **echok**) the shell does not allow to change. Without arguments, **setty** lists the modes in the chosen set which are fixed on (+mode) or off (-mode). The available modes, and thus the display, vary from system to system. With +mode, -mode or mode, fixes mode on or off or removes control from mode in the chosen set. For example, **setty** *+echok echoe* fixes echok mode on and allows commands to turn echoe mode on or off, both when the shell is executing commands.

Options

–a List all tty modes in the chosen set whether or not they are fixed.

[-d|-q|-x]
Tells **setty** to act on the edit, quote or execute set of tty modes respectively; without **-d, -q** or **-x**, execute is used.

Related Information

tcsh

source built-in command for tcsh: Read and execute commands from name

Format

source [**-h**] *name* [*args ...*]

Description

Using **source**, the shell reads and executes commands from *name*. The commands are not placed on the history list. If any arguments are given, they are placed in **argv**. **source** commands may be nested; if they are nested too deeply the shell may run out of file descriptors. An error in a source at any level terminates all nested source commands.

Options

–h Commands are placed on the history list instead of being executed, much like **history -L**.

Related Information

history, **tcsh**

telltc built-in command for tcsh: List terminal capability values

Format

telltc

Description

telltc lists the values of all terminal capabilities.

Related Information

tcsh

uncomplete built-in command for tcsh: Remove completions whose names match pattern

Format

uncomplete *pattern*

Description

uncomplete removes all completions whose names match pattern. For example, **uncomplete** * removes all completions. It is not an error for nothing to be **uncomplete**d.

Related Information

complete, **tcsh**

---

unhash built-in command for tcsh: Disable use of internal hash table

Format

unhash

Description

unhash disables use of the internal hash table to speed location of executed programs.

Related Information

tcsh

---

unlimit built-in command for tcsh: Remove resource limitations

Format

unlimit [**-h**] [*resource*]

Description

unlimit removes the limitation on *resource* or, if no resource is specified, all resource limitations.

The hard limit may be lowered to any value that is greater than or equal to the soft limit. All hard limits can be raised only by a process which has superuser authority except for **coredumpsize**, **vmemoryuse**, and **descriptors**. This behavior is identical to **ulimit** in the OS/390 shell. Both the soft limit and hard limit can be changed by a single call to **setrlimit()**.

Options

–h Corresponding hard limits are removed. Only the superuser may do this.

Related Information

limit, **tcsh**, **ulimit**

Also see **setrlimit()** in *OS/390 C/C++ Run-Time Library Reference*.

---

unsetenv built-in command for tcsh: Remove environmental variables that match pattern

Format

unsetenv *pattern*

Description

unsetenv removes all environment variables whose names match pattern. For example, **unsetenv** * removes all environment variables; we **strongly** recommend against this. It is not an error for nothing to be **unsetenv**ed.

Related Information

setenv, **tcsh**

---

watchlog built-in command for tcsh: Print the watch shell variable

Format

watchlog

Description

watch is an alternate name for the **log** built-in command. It prints the **watch** shell variable and reports on each user indicated in **watch** who is logged in, regardless of when a user last logged in.

See the **version** shell variable.

Related Information

log, **tcsh**

---

where built-in command for tcsh: Report all instances of command

Format

where *command*

Description

where reports all known instances of *command*, including aliases, built-ins and executables in path.

Related Information

tcsh, **which**

---

which built-in command for tcsh: Display next executed command

Format

which *command*

Description

which displays the command that will be executed by the shell after substitutions and path searching. This command correctly reports tcsh aliases and built-ins. See also the **which-command** editor command.

Related Information

tcsh, **where**

---

tee — Duplicate the output stream

Format

tee [**–ai**] [*file ...*]

Description

tee clones an output stream. It copies the standard input to each output file as well as to the standard output.

Options

–a Appends to (rather than overwrites) each output file.

–i Ignores interrupt signals, making it suitable for use as a background process.

Examples

The following command runs the program **prog** and pipes the program's standard output into **tee**:

```
prog | tee file
```

As a result, **tee** writes the output to both the standard output and the specified file.

Localization

tee uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion

1 Failure due to any of the following:

- Out of memory when allocating I/O buffers
- I/O error reading or writing to a file
- Error creating an output file
- Error opening an output file for appending

2 Failure due to incorrect command-line option

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Information

cat

---

test — Test for a condition

Format

test *expression*
[*expression*]

Description

test checks for various properties of files, strings, and integers. It does not produce any output other than error messages, but returns the result of the test as the exit status.

The command line is a Boolean expression. The simplest expression is a string that is true if the string is nonempty (that is, has nonzero length). More complex expressions are composed of operators and operands, each of which is a separate argument (that is, surrounded by white space). The operators imply the number and type of their operands. The operators taking a *file* operand evaluate as false (without error) if the file does not exist.

The following is a list of recognized operands:

–b *file* True if *file* is a block special file (block special files are not supported)

–c *file* True if *file* is a character special file

–d *file* True if *file* is a directory

–e *file* True if *file* exists

–f *file* True if *file* is an ordinary file

–g *file* True if the set-group-ID attribute of *file* is on

–h *file* True if *file* is a symbolic link

–k *file* True if the “sticky” bit is on *file* is on

–L *file* True if *file* is a symbolic link

–n *string* True if the length of *string* is greater than zero

–p *file* True if *file* is a FIFO (named pipe)

–r *file* True if *file* is readable (checks permission bits and access control)

–s *file* True if size of the *file* is nonzero

–t *fd* True if the numeric file descriptor *fd* is open and associated with a terminal

–u *file* True if the set-user-ID attribute of *file* is on

–w *file* True if *file* is writable (checks permission bits and access control)

–x *file* True if *file* is executable (checks permission bits and access control)

–z *string* True if the length of the *string* is zero

string True if *string* is not a null string

string1 **=** *string2*
True if *string1* and *string2* are identical

string **!=** *string*
True if *string1* and *string2* are not identical

number1 **–eq** *number2*
True if *number1* and *number2* are equal

Within the shell, either number can be an arbitrary *shell* arithmetic expression; the same applies for the other five numerical comparisons that follow. Both *number1* and *number2* must be integers.

number1 **–ge** *number2*
True if *number1* is greater than or equal to *number2*

number1 **–gt** *number2*
True if *number1* is greater than *number2*

number1 **–le** *number2*
True if *number1* is less than or equal to *number2*

number1 **–lt** *number2*
True if *number1* is less than *number2*

number1 **–ne** *number2*
True if *number1* is not equal to *number2*

file1 **–nt** *file2*
True if *file1* is newer than *file2*

file1 **–ot** *file2*
True if *file1* is older than *file2*

file1 **–ef** *file2*
True if *file1* has the same device and inode number as *file2*

expr1 **–a** *expr2*
Logical AND; true if both *expr1* and *expr2* are true

expr1 **–o** *expr2*
Logical OR; true if either *expr1* and *expr2* is true

! *expr* Logical negation; true if *expr* is false

(*expr* **)** Binding; true if *expr* is true

The precedence of the operators in descending order is: unary operators, comparison operators, logical AND, logical OR.

The second form of the test command:

`[ expression ]`

is synonymous with the first.

Usage Notes

1. **test** is a built-in shell command.
2. **test** can compare variables; however, if the variable is null, the expression may be incorrect for **test**. For example:

   ```
   NULL=
   test $NULL = "so"
   ```

   does not work, because the OS/390 shell expands this to:

   ```
   test = "so"
   ```

   which is not a valid expression for **test**. A way to get around this is to prepend some value to both strings, as in:

   ```
   test x$NULL = x"so"
   ```

   Failure to quote variable expansions is a common mistake. For example:

   ```
   test $NULL != string
   ```

   If `NULL` is undefined or empty, this results in:

   ```
   test != string
   ```

   which is not a valid test expression. This problem can be fixed by enclosing $NULL in quotes.

   Note: These two examples perform basically the same function; that is, they protect the command against a variable having a possible null value.

Examples

The following command reports on whether the first positional parameter contains a directory or a file:

```
if [ -f $1 ]
then
        echo $1 is a file
elif [ -d $1 ]
then
        echo $1 is a directory
else
        echo $1 neither file nor directory
fi
```

This example illustrates the use of **test**, and is not intended to be an efficient method.

Localization

test uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 The *expression* was true
1 The *expression* was false
2 The *expression* was badly formed

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The **–k**, **–L**, **–nt**, **–ot**, **–ef**, **–a**, and **–o** operators plus the use of parentheses to group operators together are all extensions of the POSIX standard.

Related Information

expr, **find**, **let**, **ls**, **sh**

---

tic — Put terminal entries in the terminfo database

Format

tic [**–v**]*number* [**–c**] *file*

Description

tic creates the terminfo database. It puts the compiled terminal entries in the directory **/usr/share/lib/terminfo**. If the **TERMINFO** environment variable is set, the results are placed in the directory specified by the **TERMINFO** environment variable rather than in the directory **/usr/share/lib/terminfo.**

The Curses application uses the terminfo database, which contains a list of terminal descriptions. This enables you to manipulate a terminal's display regardless of the terminal type. For information on defining the terminfo database, see *OS/390 UNIX System Services Planning*.

For more information about curses, see *OS/390 C Curses*.

Options

–v*Number*
Writes trace entries on the progress of **tic**. *Number* is an integer that indicates the level of verbosity. Levels 1, 2, 5, 7, 8, and 9 or greater are supported.

–c Specifies that the input terminal specifications are to be checked for correctness, but the terminfo database is not to be updated. If an incorrect terminal specification is encountered, a message identifying the error is written to **stdout**. The checking continues until all of the input terminal specifications have been processed.

file_name
Specifies the name of a file containing the terminal specifications. Only a single filename can be specified. The files supported by OS/390 curses are identical to the specifications with the exception that the source code must be EBCDIC rather than ASCII.

If the files are copied from an MVS data set into the HFS, the MVS data set must be in record format VB. If a filename is not specified, terminal specifications are read from the **terminfo.src** file. (The **terminfo.src** file is in the directory **/samples**.)

Note: All of the **.ti** files have been moved to the **/samples** directory.

Example

A sample command is:

```
tic /samples/ibm.ti
```

There is no output to the shell.

Environment Variables

tic uses the following environment variable:

TERMINFO
Contains the pathname of the terminfo database.

Related Information

infocmp

---

time — Display processor and elapsed times for a command

Format

time [**–p**] *command-line*

tcsh shell: **time** [*command*]

Description

time runs the command given as its argument and produces a breakdown of total time to run (`real`), total time spent in the user program (`user`), and total time spent in system processor overhead (`sys`).

Times given are statistical, based on where execution is at a clock tick. Output is written to standard error (**stderr**).

time in the tcsh shell

time executes *command* (which must be a simple command, not an alias, a pipeline, a command list, or a parenthesized command list) and prints a time summary as described under the tcsh **time** variable (see "tcsh — Invoke a C shell" on page 556). If necessary, an extra shell is created to print the time statistic when the *command* completes. Without *command*, **time** prints a time summary for the current shell and its children.

Option

–p Guarantees that the historical format of the **time** command is output.

Usage Note

time is a built-in shell command.

Localization

time uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_NUMERIC**
- **NLSPATH**

See Appendix F for more information.

Exit Values

If **time** successfully invokes *command-line*, it returns the exit status of *command-line*. Otherwise, possible exit status values are:

`0` Successful completion
`1` An error occurred in the **time** utility
`2` Failure due to an invalid command-line option
`2` Invalid command-line argument
`126` **time** found *command* but could not invoke it
`127` **time** could not find *command*

Portability

POSIX.2 User Portability Extension, X/Open Portability Guide, UNIX systems.

Related Information

sh, **tcsh**

---

times — Get process and child process times

Format

times [**–p**]

Description

times displays user and system times accumulated by the shell and commands run as children of the shell. Times are displayed in minutes and seconds. User time is CPU time spent in user programs. System time is CPU time spent in the operating system on behalf of the user process.

Option

–p Formats the output in seconds without units. For example, 1 minute and 3.7 seconds is displayed as:

```
63.47
```

Times are displayed in minutes and seconds. User time is processor time spent in user programs. System time is processor time spent in the operating system on behalf of the user process. The output layout is:

```
shell user time         shell system time
child user time         child system time
```

Usage Note

times is a built-in shell command.

Localization

times uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion

2 Failure that resulted in a usage message, usually due to an incorrect command-line option

Portability

X/Open Portability Guide.

The **–p** option is an extension to the XPG standard.

Related Information

sh, **time**

---

touch — Change the file access and modification times

Format

touch [**–acm**] [**–f** *agefile*] [**–r** *agefile*] [**–t** *time*] *file ...*
touch [**–acm**] *time file ...*

Description

The **touch** command changes certain dates for each *file* argument. By default, **touch** sets both the date of last file modification and the date of last file access to the current time. This is useful for maintaining correct release times for software and is particularly useful in conjunction with the **make** command.

Options

–a Sets only the access time.

–c Does not create any *file* that does not already exist. Normally, **touch** creates such files.

–m Sets only the modification time.

If you do not specify **–a** or **–m**, **touch** behaves as though you specified both.

To tell **touch** to use a time other than the current, use one of the following options:

–f *agefile*
Is an obsolete version of the **–r** option.

–r *agefile*
Sets the access and modification times (as indicated by the other options) to those kept for *agefile*.

–t *time*
Specifies a particular time using this format:

`[[[[cc]yy]mm]dd]hhmm [.ss]`

where:

- *cc* is the first two digits of the year (optional)
- *yy* is the last two digits of the year (optional)
- *mm* is the number of the month (01—12) (optional)
- *dd* is the day of the month (optional)
- *hh* is the hour in 24-hour format (required)
- *mm* is the minutes (required)
- *ss* is the seconds (optional)

An obsolete (but still supported) version of this command lets you omit the **–t**, but the format is:

[[*mm*]*dd*]*hhmm*[.*ss*]

or:

mmddhhmmyy[.*ss*]

Examples

1. To set the modification time of **newfile** to the present, enter:

   ```
   touch newfile
   ```

2. To set the modification time of **oldfile** to 13:05 on July 3, 1994, enter:

   ```
   touch –t 9407031305 oldfile
   ```

3. To set the modification time of **newfile** to that of **oldfile**, enter:

   ```
   touch –r oldfile newfile
   ```

Environment Variable

touch uses the following environment variable:

TZ Contains the time zone that **touch** is to use when interpreting times.

See Appendix I for more information.

Localization

touch uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

Exit Values

0 Successful completion

1 Failure due to any of the following:
- Inability to access the desired file
- Too early a date was specified
- Inability to create a file
- Inability to change a file's times

2 Failure that resulted in a usage message, including:
- Unknown command-line option
- Only one of **–t**, **–f**, or **–r** is allowed
- **–r** was missing the *agefile*
- **–t** was missing its argument
- Incorrect date string

Messages

Possible error messages include:

Age file inaccessible
Indicates that time could not be found for the file given with the **–f** or **–r** option either because that file does not exist or because the requesting user is not granted the appropriate permission for the file.

Missing age file argument
You specified **–f** or **–r**, but did not give a filename after it.

Years earlier than year incorrect
Your system recognizes dates only back to the given *year*. **touch** does not accept dates before that time.

Bad date conversion

Only one –r, –f, or –t flag allowed

Missing the date or time argument

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Information

cp, **date**

Appendix I also explains how to set the local time zone with the **TZ** environment variable.

---

tput — Change characteristics of terminals

Format

tput [**–T** *type*] *action ...*
tput [**–T** *type*] **–S** *action ...*

Description

tput lets you change your terminal's characteristics. The *action* arguments indicate how you want to change the characteristics. Possible actions are:

`clear`
Clears the screen
`init` Initializes your terminal
`reset`
Resets your terminal

tput does its work by outputting appropriate character sequences to the standard output. These character sequences are terminal-specific.

Usually, **tput** looks for an environment variable named **TERM**. If **TERM** exists, **tput** uses its value as the terminal type. If it doesn't exist, **tput** assumes a default terminal type.

Options

–T *type*
Identifies the type of your terminal. This overrides the **TERM** environment variable.

The second format of this command provides extensions for XPG/System V. This format of **tput** accepts an additional option, **–S**.

–S Takes input from standard input, one capability/action per line. A blank line terminates input.

An additional action is supported for System V:

longname
Returns the long descriptive name of the terminal.

An extension to provide System V capabilities allows *action* to be a capability from the terminfo database. If the capability requires arguments, they appear after the *action* option.

Localization

tput uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Environment Variables

tput uses the following environment variables:

TERM Contains the current terminal type.

TERMINFO Can be used to override the default database.

Exit Values

0 **tput** successfully wrote an appropriate character sequence to change the terminal's characteristics, or a Boolean terminfo variable is true.

1 A Boolean terminfo variable is false.

2 Failure that generated a usage message such as:

- *action* was not a recognized keyword
- You specified an incorrect command-line option

3 **tput** has no information about the terminal type given by **–T** or **TERM**.

4 The requested action cannot be performed on your type of terminal.

>4 An error occurred.

Portability

POSIX.2 User Portability Extension, UNIX systems.

Related Information

stty, **tabs**

---

tr — Translate characters

Format

tr [**–cs**] *string1* [*string2*]
tr –s [**–c**] *string1* [*string2*]
tr –d [**–c**] *string1*
tr –ds [**–c**] *string1 string2*

Description

tr copies data read from the standard input to **stdout**, substituting or deleting characters as specified by the options and *string1* and *string2*. *string1* and *string2* are considered to be sets of characters. In its simplest form, **tr** translates each character in *string1* into the character at the corresponding position in *string2*.

Note: **tr** works on a character basis, not on a collation element basis. Thus, for example, a range that includes the multicharacter collation element `ch` in regular expressions, does not include it here.

Options

–c Complements the set of characters specified by *string1*. This means that **tr** constructs a new set of characters, consisting of all the characters not found in *string1* and uses this new set in place of *string1*.

–d Deletes input characters found in *string1* from the output. This string is in ascending order.

–s **tr** checks for sequences of a *string1* character repeated several consecutive times. When this happens, **tr** replaces the sequence of repeated characters with one occurrence of the corresponding character from *string2*; if *string2* is not specified, the sequence is replaced with one occurrence of the repeated character itself. For example:,

```
tr –s abc xyz
```

translates the input string `aaaabccccb` into the output string of `xyzy`.

If you specify both the **–d** and **–s** options, you must specify both *string1* and *string2*. In this case, *string1* contains the characters to be deleted, whereas *string2* contains characters that are to have multiple consecutive appearances replaced with one appearance of the character itself. For example:

```
tr –ds a b
```

translates the input string `abbbaaacbb` into the output string `bcb`.

The actions of the **–s** option take place after all other deletions and translations.

String Options

You can use the following conventions to represent elements of *string1* and *string2*:

character
Any character not described by the conventions that follow represents itself.

ooo
An octal representation of a character with a specific coded value. It can consist of one, two, or three octal digits. Doublebyte characters require multiple, concatenated escape sequences of this type, including the leading \ for each byte.

character
The \ (backslash) character is used as an escape to remove the special meaning of characters. It also introduces escape sequences for nonprinting characters, in the manner of C character constants: `\b`, `\f`, `\n`, `\r`, `\t`, and `\v`.

c1–c2
This represents all characters between characters *c1* and *c2* (in the current locale's collating sequence) including the end values. For example, '`a–z`' represents all the lowercase letters in the POSIX locale, whereas '`A–Z`' represents all that locale's uppercase letters. One way to convert lowercase and uppercase is with the following filter:

```
tr 'a-z' 'A-Z'
```

This is not, however, the recommended method; use the `[:`*class*`:]` construct instead.

*c***n* This represents *n* repeated occurrences of character *c*. (If *n* has a leading zero, **tr** assumes it is octal; otherwise, it is assumed to be decimal.) You can omit the number for the last character in a subset. This representation is valid only in *string2*.

`[:`*class*`:]`
This represents all characters that belong to the character class *class* in the locale indicated by **LC_CTYPE**. When the class `[:upper]` or `[:lower:]` appears in *string1* and the opposite class, `[:lower:]` or `[:upper:]` appears in *string2*, **tr** uses the **LC_CTYPE** `tolower` or `toupper` mappings in the same relative positions.

`[=`*c*`=]`
This represents all characters that belong to the same equivalence class as the character *c* in the locale indicated by **LC_COLLATE**. Only international versions of the code support this format.

Examples

This example creates a list of all words (strings of letters) found in file1 and puts it in file2:

```
tr –cs "[:alpha:]" "\n*" <file1 >file2
```

Localization

tr uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion
1 Failure because of unknown command line option, or too few arguments

Portability

POSIX.2, X/Open Portability Guide.

tr is downward–compatible with both the UNIX Version 7 and System V variants of this command, but with extensions (C escapes, handles ASCII NUL, internationalization).

---

trap — Intercept abnormal conditions and interrupts

Format

trap ['*handler*'] [*event ...*]

Description

trap intercepts certain kinds of exception conditions. Any signal may be intercepted by specifying an event corresponding to the signal number.

With an event of `ERR`, **trap** invokes the handler after receiving any having a nonzero exit status. The exception to this is conditions in **if**, **while**, and **until** statements. This trap is not inherited within a function.

With a trap number of `0` or `EXIT`, **trap** invokes the handler during exit from the shell. Within a function, it is invoked during exit from the function.

Any other event corresponds to a signal number or signal name. (See **kill** for a table of valid signal numbers and their names.) If a signal is being ignored when you enter the shell, the shell continues to ignore it without regard to any traps.

Because system initialization sets the value of the SIGIOERR signal to `ignore`, this signal cannot be set by **trap**.

The *handler* argument is a command list. It is usually more than one word, and so you must quote it to appear as a single argument. It is scanned when the trap function is initially invoked. When the trap condition is raised, the shell scans the command list again and runs the commands. A missing argument or an argument

of – (dash) resets the default trap condition. A null argument (`''`) causes the trap condition to be ignored.

If there are no arguments at all, **trap** prints a list of all the traps and their commands.

Usage Note

trap is a special built-in shell command.

Examples

1. On error or exit, this example deletes a temporary file created during command execution.

   ```
   trap 'rm –f /tmp/xyz$$; exit' ERR EXIT
   ```

   When an interrupt signal is received, the example prompts whether to abort, and exits if the answer is `y`.

   ```
   trap 'read REPLY?"ABORT??"
           case $REPLY in
           y)      exit 1;;
           esac'   2
   ```

2. This example saves your shell history file (specified by the value you give the **HISTFILE** environment variable) before timing you out, so you can restore it when you log on again.

   ```
   trap 'cp $HISTFILE $HOME/old_hist.bak; exit' ALRM
   ```

Localization

trap uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

`0` Successful completion

`1` Failure due to any of the following:

- Incorrect signal name
- Incorrect signal number

`2` Incorrect command-line argument

Messages

Possible error messages include:

name **not a valid trap name**
You specified an unrecognized trap name. The usual cause of this error is a typing mistake on the command line.

Portability

POSIX.2, X/Open Portability Guide.

Related Information

sh

---

true — Return a value of 0

Format

true [*argument ...*]

Description

true simply yields an exit status of zero (success). It ignores any arguments given on the command line. This can be surprisingly useful—for example, when you are evaluating shell expressions for their side effects.

Usage Note

true is a built-in shell command.

Localization

true uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Value

Since **true** always succeeds, the only possible exit status is:

`0` Successful completion

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Information

sh

---

tso — Run a TSO/E command from the shell

Format

tso [**–o**] [**–t**] *TSO_command*

Description

tso runs a TSO/E command from the shell using the TSO/E service routine or the OMVS interface.

Options

–o Specifies that the command be issued through the OMVS interface.

–t Specifies that the command be issued through the TSO/E service routine. If a mini-TSO/E environment is to be established, use environment variables to specify the allocations that you need.

If you do not specify an option, the following rules determine how to run the TSO/E command:

- If **stdout** is not a tty, the TSO/E service routine is used because it is possible that the command output will be redirected to a file or piped to another command.
- If the controlling tty supports 3270 passthrough mode, the OMVS interface is used.
- If neither of the above is applicable, then the TSO/E service routine is used.

Examples

1. To use OPUT to copy a file to a file in your current directory, issue:

   ```
   tso –t "oput source.c(hello) 'hello.c' "
   ```

   If you do not specify **–t**, the command is run in your TSO/E session through OMVS, if possible. This copies the file to a file relative to the working directory of your TSO/E session, which is usually your home directory.

   Quotes are used around the command to avoid shell parsing.

2. To use OPUTX to copy all members of a PDS to your current directory, issue:

   ```
   tso –t "oputx source.c . lc suffix(c)"
   ```

   If you do not specify the **–t** option, the command is run in your TSO/E session through OMVS, if possible. This copies the file to a file relative to the working directory of your TSO/E session which is usually your home directory.

   Quotes are used around the command to avoid shell parsing.

   Since OPUTX uses ISPF, allocations for the ISPF DD names must be performed to run this command. The following is an example of the environment variables that are set to perform these allocations. This can be included in your .profile for convenience. Make sure the export statements start in column one. The data set names may differ on your system.

```
# Assign the DD names to allocate
#
export TSOALLOC=ispprof:ispplib:ispmlib:isptlib:isplllib:ispslib:\
isptabl:isplog:sysexec
#
# Allocate an empty, temporary ISPF profile data set
#
export ispprof="alloc new unit(sysvio) space(1,1) cyl dir(5) \
recfm(f,b) lrecl(80) blksize(3120)"
#
# Allocate an empty, temporary ISPF table data set
#
export isptabl="alloc new unit(sysvio) space(1,1) cyl dir(5) \
recfm(f,b) lrecl(80) blksize(3120)"
#
# Allocate the ISPF log to SYSOUT
#
export isplog="alloc sysout(a) recfm(v,a) lrecl(125) blksize(129)"
#
# Allocate the OpenMVS and ISPF panel data sets to ISPPLIB
#
export ispplib=SYS1.SBPXPENU:SYS1.ISP.SISPPENU
#
# Allocate the OpenMVS and ISPF message data sets to ISPMLIB
#
export ispmlib=SYS1.SBPXMENU:SYS1.ISP.SISPMENU
#
# Allocate the ISPF table data set to ISPTLIB
#
export isptlib=SYS1.ISP.SISPTENU
#
# Allocate the ISPF skeleton data set to ISPSLIB
#
export ispslib=SYS1.ISP.SISPSENU
#
# Allocate any load library to ISPLLIB if ISPF is in LINKLIST/LPA
#
export isplllib=SYS1.LINKLIB
#
# Allocate the OpenMVS EXEC data set to SYSEXEC
#
export sysexec=SYS1.SBPXEXEC
```

Environment Variables

If the **tso** command is to be run through the TSO/E service routine, you may need to perform allocations or other customization for the TSO/E environment. These can be specified using environment variables. You can use the following environment variables:

SYSEXEC Specifies the allocation specification for the SYSEXEC DD name. If the **TSOALLOC** variable is set, this variable is not automatically used.

SYSPROC Specifies the allocation specification for the SYSPROC DD name. If the **TSOALLOC** variable is set, this variable is not automatically used.

TSOALLOC
Specifies the names of the environment variables that contain allocation specifications. The names are separated by colons. Case is

respected; lowercase letters are treated as lowercase. The names of the environment variables also correspond to the name of the DD name to be allocated. The DD name is always treated as uppercase but the variable name can be specified in mixed case to avoid possible conflict with similar environment variable names.

TSOPREFIX
Specifies a prefix for temporary data sets that need to be cataloged. Lowercase letters are treated as uppercase letters. If you do not specify this variable, the user's login name (user ID) is used.

TSOPROFILE
Specifies that the profile be reset with the arguments you specify when running the TSO/E command. (The specified arguments replace the default values.) For example, to set the TSO prefix and to turn off message IDs, issue:

```
export TSOPROFILE="prefix(wjs) nomsgid"
```

The value of this variable is passed to the TSO/E PROFILE command as is. If the PROFILE command fails, the requested command is not run. The output from the PROFILE command is sent to **stdout** along with the PROFILE command that was issued.

An allocation specification can be either a list of cataloged data set names separated by colons or a data set allocation request. If a list of data set names is used, lowercase letters are treated as uppercase and the data set names must be fully qualified.

You can specify a data set allocation request by beginning the specification with the keyword *alloc* followed by keywords or keyword-value pairs in a format similar to the TSO/E ALLOCATE command. Keys are supported by blanks.

DA (*data set name* [(*member name*)]) | **DSN** (*data set name* [(*member name*)])
Data set name to allocate. The name must be fully qualified and can include a member name. Quotes can be used but are ignored.

MOD | **NEW** | **OLD** | **SHR**
Specifies the status of the data set.

CATALOG | **DELETE** | **KEEP** | **UNCATALOG**
Specifies the data set disposition.

TRACKS
Specifies that space be allocated in units of tracks.

CYL Specifies that space be allocated in units of cylinders

DIR(*directory blocks*)
Specifies the number of directory blocks.

SPACE(*primary*[,*secondary*])
Specifies that primary and (optionally) secondary space be allocated.

VOL(*volume serial*)
Specifies the VOLSER.

UNIT(*unit name*)
Specifies the unit name, device type, or unit address.

SYSOUT[(*class*)]
Specifies that a sysout data set is to be allocated and optionally defines the output class.

WRITER(*external writer name*)
Specifies the external writer.

FORMS(*forms name name*)
Specifies the print form.

DEST(*destination*)
Specifies the output destination.

COPIES(*number of copies*)
Specifies the number of copies to be printed.

DUMMY
Specifies that a dummy data set be allocated.

BLKSIZE(*block size*)
Specifies the block size.

LRECL(*record length*)
Specifies the logical record length.

DSORG(**PS**|**PO**|**DA**)
Specifies the data set organization.

RECFM(*format*[,*format...*])
Specifies the record format. The values are A, B, D, F, M, S, T, U, and V. You can combine several of these values.

STORCLAS(*storage class*)
Specifies the storage class.

MGMTCLAS(*management class*)
Specifies the management class

DATACLAS(*data class*)
Specifies the data class.

RECORG(**LS**)
Specifies that a VSAM linear data set be created.

DSNTYPE(**LIBRARY**|**PDS**|**HFS**)
Specifies the data set type.

SPIN(**UNALLOC**)
Specifies that a sysout data set be spun off at allocation.

NORECALL
Specifies that the allocation request be failed if the data set is migrated.

PATH('*pathname*')
Specifies that the allocation is for a file in the HFS.

PATHOPTS(*pathopt*[,*pathopt*]...)
Specifies a list of path options: **ORDWR OEXCL OSYNC OTRUNC OCREAT OWRONLY ORDONLY OAPPEND ONOCTTY ONONBLOCK**.

PATHMODE(*pathmode*[,*pathmode*]...)
Specifies a list of pathmodes: **SIRUSR SIWUSR SIXUSR SIRWXU SIRGRP SIWGRP SIXGRP SIRWXG SIROTH SIWOTH SIXOTH SIRWXO SISUID SISGID SISVTX**

PATHDISP(**KEEP**|**DELETE**[,**KEEP**|**DELETE**])
Specifies the normal and abnormal file disposition.

FILEDATA(**TEXT**|**BINARY**)
Specifies whether the data is to be treated as text or binary.

Messages

`0–254` Successful completion
`255` The return code is outside the range `0–254` or the **tso** command ended in error

---

tsort — Sort files topologically

Format

tsort [*file*]

Description

tsort reads input from files (or from the standard input if you do not specify a file) and produces an ordered list of items consistent with a partial ordering of items provided by the input.

Input to **tsort** takes the form of pairs of items (nonempty strings) separated by blanks. A pair of two different items indicates ordering. A pair of identical items indicates presence, but not ordering.

Example

The command:

```
tsort <<EOF
a b c c d e
g g
f g e f
h h
EOF
```

produces the output:

```
a
b
c
d
e
f
g
h
```

Localization

tsort uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion
>0 An error occurred

Portability

POSIX.2, X/Open Portability Guide.

---

tty — Return the user's terminal name

Format

tty [**–s**]

Description

tty displays the filename of the terminal device associated with the standard input.

Options

–s Does not display the name; the exit status of **tty** indicates whether the standard input is a terminal.

Localization

tty uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Standard input is a terminal
1 Standard input is not a terminal
2 Failure because of an unknown command-line option, or too many arguments

Messages

Possible error messages include:

Not a tty The standard input is not associated with a terminal.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The POSIX standard considers the **–s** option to be obsolete.

type — Tell how the shell interprets a name

Format

type *name ...*

Description

type identifies the nature of one or more names. Names can be shell reserved words, aliases, shell functions, built-in commands, or executable files. For executable files, the full pathname is given.

Usage Note

type is a built-in shell command.

Localization

type uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion
2 Failure because of an incorrect command-line argument

Messages

Possible error messages include:

name **is not found**
type could not locate the specified name. Check that the *name* was specified properly and that you have the appropriate permissions.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Information

alias, **command**, **sh**, **whence**

typeset — Assign attributes and values to variables

Format

typeset ±**f** [**tux**] *name ...*
typeset [±**lprtuxH**] [±**iLRZ**[*number*]] [*variable*[**=***value*] ....]

Description

Invoking **typeset** with no options displays a list of all variables and their attributes. This list is sorted by variable name and includes quoting so that it can be reinput to the shell with the built-in command **eval**. When only arguments of the form **+***option* are specified, **typeset** displays a list of the variables that have all specified attributes set. When only arguments of the form **–***option* are present, **typeset** displays a list of all the variables having all the specified attributes set, and also displays their values.

When the **f** option is used, **typeset** applies to functions; otherwise, it applies to variables. For functions, the only other applicable options are **–t**, **–u** and **–x**.

If the command line contains at least one variable, the attributes of each variable are changed. In this case, parameters of the form **–***option* turn on the associated attributes. Parameters of the form **+***option* turn off the associated attributes. (Notice that, contrary to what you might expect, – means *on,* and + means *off*.) Parameters of the form *variable***=***value* turn on the associated attributes and also assign *value* to *variable*.

When **typeset** is invoked inside a function, a new instance of each variable is created. After the function ends, each variable is restored to the value and attributes it had before the function was called.

Options

–H Performs file mapping from POSIX to the host name.

–i[*number***]**
Marks each variable as having an integer value, thus making arithmetic faster. If *number* is given and is nonzero, the output base of each *variable* is *number*. The default is decimal.

–l Converts uppercase characters to lowercase in any value assigned to a variable. If the **–u** option is currently turned on, this option turns it off.

–p Writes output to the coprocess. This option is not currently implemented.

–r Makes each variable read-only. See **readonly**.

–t Tags each variable. Tags are user-defined, and have no meaning to the shell. For functions with the **–f** option, this turns on the `xtrace` option. See **set** for a discussion of the `xtrace` option.

–u Converts lowercase characters to uppercase in any value assigned to a variable. If the **–l** option is currently turned on, this option turns it off.

When used with **–f**, the **–u** option indicates that the functions named in the command line are not yet defined. The attributes specified by the **typeset** command are applied to the functions once they are defined.

–x Sets each variable for automatic export. See **export**.

The last three options that follow justify, within a field, the values assigned to each variable. The width of the field is *number* if it is defined and is nonzero; otherwise, the width is that of the first assignment made to variable.

–L[*number***]**
Left-justifies the values assigned to each variable by first removing any leading blanks. Leading zeros are also removed if the **–Z** option has been turned on. Then blanks are added on the end or the end of the value is truncated as necessary. If the **–R** flag is currently turned on, this option turns it off.

–R[*number***]**
Right-justifies the values assigned to each variable by adding leading blanks or by truncating the start of the value as necessary. If the **–L** flag is currently turned on, this option turns it off.

–Z[*number***]**
Right-justifies values assigned to each variable. If the first nonblank character of value is a digit, leading zeros are used. See also the **–L** option.

Usage Note

typeset is a built-in shell command as well as a separate utility.

Localization

typeset uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion
2 Failure due to an incorrect command-line argument

If the command is used to display the values of variables, the exit status value is the number of names that are incorrect.

Messages

Possible error messages include:

Base number not in [2,36]
You used the **–i** option to specify a base for an integer, but the base was not in the range 2 through 36. All bases must be in this range.

name **not a function**
You tried to declare the given name as a function, but the name already referred to something that was not a function (for example, a variable).

Portability

POSIX.2. It is an extension to the POSIX.2 and XPG standards.

Related Information

export, **readonly**, **sh**

---

uconvdef — Create binary conversion tables

Format

uconvdef [–f SrcFile] [–v] uconvTable

Description

uconvdef reads **SrcFile** and creates **uconvTable**, a binary conversion table. **SrcFile** is the input source file that defines a mapping between UCS-2 and multibyte code sets.

Note: UCS-2 is the Universal Multiple-Octat Coded Character Set defined by ISO/IEC 10646-1:1993(EE), while multibyte code sets consists of one or more bytes per character.

uconvTable is in a format that can be opened and read by **iconv** conversion functions.

Options

–f SrcFile
SrcFile is the input source file that defines a mapping between UCS-2 and another single or multibyte code set. If this option is not used, standard input is read. For information on the format of the input source table, refer to the **ucmap** description in *OS/390 C/C++ Programming Guide*.

–v Specifies that the **SrcFile** file statements be displayed.

uconvTable
Specifies the pathname of the compiled table created by the **uconvdef** command. This file defines conversions into and out of UCS-2.

Example

To create the compiled **uconvTable** that defines the conversion table between IBM-1047 and UCS-2, issue:

```
uconvdef –f IBM-1047.ucmap /usr/lib/nls/locale/uconvTable/IBM-1047
```

The \ (backslash) is a line continuation character that is needed if the command is broken into multiple lines.

Exit Values

0 Successful completion.
>**0** An error occurred.

Related Information

iconv

The **iconv** subroutine, **iconv_close** subroutine, **iconv_open** subroutine (refer to *OS/390 C/C++ Programming Guide*).

---

ulimit — Set process limits

Format

ulimit [**–aSHctdfsn**] [*num*]

Description

ulimit sets or displays the resource limits on processes created by the user.

Options

–a Displays all resource limits that are available.

–S Set or display the soft limit(s). The soft limit may be modified to any value that is less than or equal to the hard limit. For certain *resource* values, the soft limit cannot be set lower than the existing usage.

–H Set or display the hard limit(s). The hard limit may be lowered to any value that is greater than or equal to the soft limit. The hard limit can be raised only by a process which has superuser authority.

–c Set or display the core file limit. The core file limit is the maximum size of a dump of memory (in 512–byte blocks) allowed for the process. A value of 0 (zero) prevents file creation. Dump file creation will stop at this limit.

–t Set or display the cpu time limit. The cpu time limit is the maximum amount of CPU time (in seconds) allowed for the process. If the limit is exceeded, a SIGXCPU signal is sent to the process and the process is granted a small CPU time extension to allow for signal generation and delivery. If the extension is used up, the process is terminated with a SIGKILL signal. An attempt to set the CPU limit lower than that already used will fail.

–d Set or display the data size limit. The data size limit is the maximum size of the break value for the process, in units of 1024 bytes. This resource always has unlimited hard and soft limits.

–f Set or display the file size limit. The file size limit is the maximum file size (in 512–byte blocks) allowed for the process. A value of 0 (zero) prevents file creation. If the size is exceeded, a SIGXFSZ signal is sent to the process. If the process is blocking, catching, or ignoring SIGXFSZ, continued attempts to increase the size of a file beyond the limit will fail.

–s Set or display the stack size limit. The stack size limit is the maximum size of the stack for a process, in units of 1024 bytes. The stack is a per-thread resource that has unlimited hard and soft limits.

–n Set or display the file descriptors limit. The file descriptors limit is the maximum number of open file descriptors allowed for the process. This number is one greater than the maximum value that may be assigned to a newly created descriptor. Any function that attempts to create a new file

descriptor beyond the limit will fail. An attempt to set the open file descriptors limit lower than that already used will fail.

num The new limit. *num* can be specified as “unlimited.”

Usage Notes

ulimit is a built-in shell command.

Localization

ulimit uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Related Information

setrlimit in *OS/390 C/C++ Run-Time Library Reference*.

---

umask — Set or return the file mode creation mask

Format

umask [**–S**] [*mode*]

tcsh shell: **umask** [*value*]

Description

umask sets the file-creation permission-code mask of the invoking process to the given *mode*. You can specify the *mode* in any of the formats recognized by **chmod**; see **chmod** for more information.

The *mode* may be specified in symbolic (rwx) or octal format. The symbolic form specifies what permissions are allowed. The octal form specifies what permissions are disallowed.

The file-creation permission-code mask (often called the *umask*) modifies the default (initial) permissions for any file created by the process. The umask specifies the permissions which are **not** to be allowed.

If the bit is turned off in the umask, a process can set it on when it creates a file. If you specify:

```
umask a=rx
```

you have allowed files to be created with read and execute access for all users. If you were to look at the mask, it would be 0222. The write bit is set, because write is not allowed. If you want to permit created files to have read, write, and execute access, then set umask to 0000. If you call **umask** without a *mode* argument, **umask** displays the current umask.

umask in the tcsh shell

In the tcsh shell, **umask** sets the file creation mask to *value*, which is given in octal. Common values for the mask are `002`, giving all access to the group and read and execute access to others, and `022`, giving read and execute access to the group and others. Without *value*, **umask** prints the current file creation mask. See "tcsh — Invoke a C shell" on page 556.

Options

–S Displays the umask in a symbolic form:

`u=perms,g=perms,o=perms`

giving owner, group and other permissions. Permissions are specified as combinations of the letters **r** (read), **w** (write), and **x** (execute).

Localization

umask uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

`0` Successful completion
`1` Failure due to an incorrect command-line argument, or incorrect *mode*

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Information

chmod, **tcsh**

---

unalias — Remove alias definitions

Format

unalias *name ...*
unalias –a

tcsh shell: **unalias** *pattern*

Description

unalias removes each alias *name* from the current shell execution environment.

unalias in the tcsh shell

In the tcsh shell, **unalias** removes all aliases whose names match *pattern*. For example,

```
unalias *
```

removes all aliases. It is not an error for nothing to be **unalias**ed. See "tcsh — Invoke a C shell" on page 556.

Options

–a Removes all aliases in the current shell execution environment.

Localization

unalias uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Usage Notes

unalias is a built-in shell command.

Exit Values

0 Successful completion

1 There was an alias that could not be removed

2 Failure due to an incorrect command-line option or there were two aliases that could not be removed

>2
Tells the number of aliases that could not be removed

Portability

POSIX.2 User Portability Extension, X/Open Portability Guide.

Related Information

alias, **sh**, **tcsh**

---

uname — Display the name of the current operating system

Format

uname [**–amnrsv**]

Description

The **uname** command lets shell scripts and other programs determine configuration information about the machine upon which the shell is running.

Options

The following options select the information to be displayed:

–a All fields (equivalent to **–mnrsv**).

–m The processor or machine type.

–n The node name of this particular machine. The node name is set by the SYSNAME sysparm (specified at IPL), and it usually differentiates machines running at a single location.

–r The release (minor version) number of the operating system.

–s The name of the operating system. This is the default output, when no options are given.

–v The version (major version) number of the operating system.

uname displays the selected information in the following order:

<system name> <nodename> <release> <version> <machine>

Example

The following shell command changes the prompt to identify the node name of the system:

```
export PS1=" uname –n   $ "
```

Localization

uname uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Usage Note

uname is a built-in shell command.

Exit Values

0 Successful completion
1 Failure due to inability to find the desired information
2 Failure due to a incorrect command-line option

Portability

POSIX.2, X/Open Portability Guide, UNIX System V.

Related Information

sh

---

uncompress — Undo Lempel-Ziv compression of a file

Format

uncompress [**cDfVv** [*file*]]

Description

uncompress expands compressed data written by the Lempel-Ziv compression program **compress**. Data is read from *file* or the standard input. On UNIX systems, the name of the file to be uncompressed must end with **.Z**. If it doesn't, **uncompress** adds one before looking for the file. It places the uncompressed output in a file with the same name but without the **.Z** extension. If this file already exists, **uncompress** asks if you want to overwrite it, unless you specify the **–f** option.

Since the number of bits of compression is encoded in the compressed data, **uncompress** automatically uses the correct number of bits. This includes the 9–14 bit compression range specified by POSIX.

Options

–c Writes uncompressed output to the standard output (like **zcat**).

–D Must be used to uncompress a sorted dictionary file compressed using the **–D** option of **compress**.

–f Forces file to be uncompressed, regardless of whether a file with the same base name already exists.

–V Prints version number information for **uncompress**.

–v Displays name of each file when it is uncompressed.

Localization

uncompress uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion

1 Failure due to any of the following:

- Unknown command-line option
- Inability to obtain information about an argument file
- File has more than one link
- File is not a regular file
- File is not in compressed format
- File was compressed using more than 16 bits
- There is no space for decompress tables
- A compressed file is corrupt

Portability

uncompress is found on many Unix systems.

The **–D** option is an extension to traditional implementations of **uncompress**; the **–D** and **–V** options are extensions to the POSIX standard.

Related Information

compress, **cpio**, **pack**, **unpack**, **zcat**

---

unexpand — Compress spaces into tabs

Format

unexpand [**–a**] [**–t** *tablist*] [*file ...*]

Description

unexpand replaces blank characters in the data from each *file* argument with the most efficient use of tabs and spaces. If you do not specify any files, **unexpand** reads the standard input. The result is sent to standard output.

Backspace characters are preserved. By default, **unexpand** compresses only leading spaces into tabs; tab stops are set every eight spaces.

Options

unexpand supports the following options:

–a Compresses spaces into tabs wherever the resulting output is shorter, regardless of where the spaces occur in the line.

–t *tablist*
Specifies tab stops. The numbers in *tablist* are delimited by blanks or commas. If *tablist* is a single number, then **unexpand** places tab stops every *tablist* positions. If *tablist* contains multiple numbers, **unexpand** places tab stops at those specific positions. Multiple numbers in *tablist* must be in ascending order. This option, like the **–a** option, compresses spaces to tabs at any appropriate point in the line. If you specify **–t**, **unexpand** ignores the presence or absence of **–a**.

Localization

unexpand uses the following localization variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion

1 Failure due to an incorrect command-line argument, or an inability to open the input files

Portability

POSIX.2 User Portability Extension, X/Open Portability Guide, 4.2BSD.

Related Information

expand, **pr**

---

uniq — Report or filter out repeated lines in a file

Format

uniq [**–c**|**–d**|**–u**] [**–f** *number1*] [**–s** *number2*] [*input_file* [*output_file*]]
uniq [**–cdu**] [**–***number*] [**+***number*] [*input_file* [*output_file*]]

Description

uniq manipulates lines that occur more than once in a file. The file must be sorted, since **uniq** only compares adjacent lines. When you invoke this command with no options, it writes only one copy of each line in *input_file* to *output_file.* If you do not specify *input_file* or you specify –, **uniq** reads the standard input.

If you do not specify *output_file*, **uniq** uses the standard output. The specified *output_file* cannot be a FIFO.

Options

–c Precedes each output line with the number of times that line occurred in the input.

–d Displays only lines that are repeated (one copy of each line).

–f *number1*
Ignores the first *number1* fields when comparing lines. Blanks separate fields in the input.

–s *number2*
Ignores the first *number2* characters when comparing lines. If you specify both **–s** and **–f**, **uniq** ignores the first *number2* characters after the first *number1* fields.

–u Displays only those lines that are not repeated.

You can choose only one of the **–c**, **–d**, or **–u** options.

–*number* Equivalent to **–f** *number* (obsolescent).
+*number* Equivalent to **–s** *number* (obsolescent).

Examples

1. The command:

   ```
   uniq
   ```

   is a filter which prints one copy of each different line in its sorted input.

2. The command:

   ```
   uniq -f 2 -s 1
   ```

   compares lines starting with the second character of the third field.

3. The command:

   ```
   uniq -d
   ```

   prints one instance of each repeated line in the input (and omits all unique lines).

Localization

uniq uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

`0` Successful completion

`1` Failure due to any of the following:

- Incorrect command-line option
- Missing *number* after **–f**
- Missing or incorrect *number* after **–s**
- Inability to open the input or output file

Messages

Possible error messages include:

Missing character skip count
You specified **–s** but did not supply a number after the **–s**.

Missing number of fields to skip
You specified **–f** but did not supply a number after the **–f**.

Field skip not a number in *string*
In a **-***number* or **+***number* construct, *number* was not a valid number. This could arise because of a typographical error in entering a **–** option.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Information

comm, **sort**

---

unlink — Removes a directory entry

Format

unlink *file*

Description

unlink removes a directory entry.

Following the format, *file* specifies the entry to be removed, which can refer to a pathname, a hard link, or a symbolic link. If *file* refers to a symbolic link, **unlink** removes the symbolic link but not any file or directory named by the contents of the symbolic link. If the entry that is unlinked is the last one associated with a file, then the file itself will be deleted.

unlink is implemented as a shell built-in.

Localization

unlink uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Values

`0` Successful completion

`1` Failure due to any of the following:

- No write permission for directory containing link to be removed
- Attempting to unlink a file that does not exist
- Pathname specified is a directory

`2` Failure due to incorrect number of arguments specified

Related Information

mv, **rm**, **rmdir**

---

unmount — Remove a file system from the file hierarchy

Format

unmount [**–R**] [**–v**] [**–o normal|drain|immediate|force|reset**] *pathname...*

Description

The **unmount** shell command, located in **/usr/sbin**, unmounts file systems.

Note: An **unmount** user must have UID(0) or at least have READ access to the SUPERUSER.FILESYS.MOUNT resource found in the UNIXPRIV class.

Options

–R Unmounts the specified file system and all the file systems below it in the file system hierarchy.

–o normal|drain|immediate|force|reset

normal Specifies that if no user is accessing any of the files in the specified file system, the system processes the **unmount** request. Otherwise, the system rejects the **unmount** request. This is the default

drain Specifies that an unmount drain request is to be made. The system will wait for all use of the file system to be ended normally before the unmount request is processed or until another UNMOUNT command is issued.

Note: Currently **unmount –o drain** is not supported in a sysplex. If an **unmount –o drain** is issued in a sysplex, the following behavior is exhibited:

- If there is no activity in the file system, **unmount -o drain** will perform the unmount, but it will behave like an **unmount normal**.
- If there is activity in the file system, **unmount -o drain** will return a Return_value of -1 with Return_code EINVAL and Reason_code JrNotSupInSysplex.

immediate
The system will unmount the file system immediately. Any users accessing files in the specified file system will receive failing return codes. All data changes to files in the specified file system are saved. If the data changes cannot be saved, the **unmount** request fails.

force Also specifies that the system will unmount the file system immediately. Any users accessing files in the specified file system will receive failing return codes. If possible, all data changes to files in the specified file system are saved. If the

data changes to the files cannot be saved, the **unmount** request continues and data is lost.

Note: An **unmount –o immediate** request must be issued before you can request an **unmount –o force** of a file system. Otherwise, **unmount –o force** fails.

reset A **reset** request stops a previous **unmount –o drain** request.

Note: **unmount –o reset** is not supported in a sysplex.

–v Lists all file systems that are unmounted.

pathname... specifies the pathnames to use for locating the file systems you want unmounted. This may be the pathname for any file or directory within that file system.

Examples

1. The output of **mount –q** can be used for the input of **unmount**. For example:

   ```
   mount -q /ict/hfsfir
   ```

   can be used as input:

   ```
   unmount $(mount -q /ict/hfsdir)
   ```

2. To unmount a file system that contains the file or directory **/u/wjs**:

   ```
   unmount /u/wjs
   ```

3. To unmount a file system that contains the file or directory **/u** along with all other file systems mounted over or below that file system:

   ```
   unmount -R /u
   ```

Exit Values

0 Successful completion

Related Information

chmount, **mount**

---

unpack — Decode Huffman packed files

Format

unpack *file...*

Description

unpack uncompresses files compressed by **pack**, using a Huffman minimal redundancy code. By default, **unpack** looks for *file* with a **.z** extension. It places the decompressed output in a file with the same name, but without the extension. The owner, permissions, and times of last access and last modification are also preserved. Packed files can be identified by *file*. You can use **pcat** to view packed text files without unpacking them in place.

unpack doesn't unpack a file if:

- The file name is too long after the **.z** is removed

- The input file cannot be opened
- An existing file has the same name as the output file
- The output file can't be created
- The input file doesn't appear to have been created by **pack**

Localization

unpack uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLATE**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

File

unpack uses the following file:

pk$* Temporary copy of input file. (You may see this in the current directory if **unpack** is interrupted.) The file is located in the same directory as the file being unpacked.

Exit Values

`0` Successful completion

`n` Indicates that files could not be unpacked properly. For example, if three out of six files could not be unpacked properly, the exit status is `3`.

Possible reasons for failure include:

- Unknown command-line option
- Error creating a name for a temporary file
- Error opening an input file or a temporary file
- Error writing to a temporary file
- Inability to rename a temporary file
- Inability to restore the modification time on a packed file
- Input file was not packed
- A packed file is corrupt

Messages

Possible error messages include:

file: **Not a packed file**
pack did not process the file. In this case, the file is not changed.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Information

file, **pack**, **pcat**

---

unset — Unset values and attributes of variables and functions

Format

unset *name ...*
unset –fv *name ...*

tcsh shell: **unset** *pattern*

Description

Calling **unset** with no options removes the value and attributes of each variable or function name.

unset in the tcsh shell

unset removes all variables whose names match pattern, unless they are read-only. For example:

```
unset *
```

which we **strongly** recommend you do not do, will remove all variables unless they are read-only. It is not an error for nothing to be **unset**.

See “tcsh — Invoke a C shell” on page 556.

Options

–f Removes the value and attributes of each function *name*.
–v Removes the attribute and value of the variable *name*. This is the default if no options are specified.

unset cannot remove names that have been set read-only.

Usage Notes

unset is a special built-in shell command.

Localization

unset uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion
1 Failure due to an incorrect command-line option
2 Failure due to an incorrect command-line argument

Otherwise, **unset** returns the number of specified *names* that are incorrect, not currently set, or read-only.

Messages

Possible error messages include:

name **readonly variable**
The given *name* cannot be deleted because it has been marked read-only.

Portability

POSIX.2, X/Open Portability Guide.

Related Information

readonly, **sh**, **tcsh**

---

uucc — Compile UUCP configuration files

Format

uucc

Description

uucc reads the contents of the **uucp** configuration files and compiles them into a single configuration file called **/usr/lib/uucp/config**. The configuration files are:

- Systems
- Devices
- Dialers
- Dialcodes
- Permissions

Because **uucc** expects these text files to be in the current working directory, you need to change the directory (with the **cd** command) to **/usr/lib/uucp** before issuing **uucc**. For more information on creating and maintaining UUCP configuration files, refer to *OS/390 UNIX System Services Planning*.

Files

uucc uses the following files:

/usr/lib/uucp/Systems
Contains a list of remote systems and the methods for connecting with them.
/usr/lib/uucp/Devices
Describes the physical and logical connections listed in the Systems file.
/usr/lib/uucp/Dialers
Contains dialing information for the modems and dialers listed in the Devices file.

/usr/lib/uucp/Dialcodes
Contains abbreviations that can be used in the phone numbers specified in Systems.

/usr/lib/uucp/Permissions
Defines the commands and areas of the file system that remote sites can access on your system.

/usr/lib/uucp/config
Contains the above information compiled into one file for use by the uucp utilities.

Exit Values

0 Successful completion

1 Failure due to any of the following:

- Inability to open file
- Insufficient memory
- Ctrl-C interrupt

Related Information

uucp

---

uucico daemon — Process UUCP file transfer requests

Format

uucico [**–f**] [**–g** *grade*] [**–r0**|**–r1**] [**–s** *system*] [**–x** *type*]

Description

The **uucico** daemon processes file transfer requests that were queued by **uucp** and **uux**. It establishes the connection with remote sites and manages the transfer of data between the local and remote sites as specified by the queued **uucp** or **uux** job.

uucico is automatically invoked after the **uucp** or **uux** command completes (unless the **–r** option was specified on the **uucp** or **uux** command). To process requests that cannot be successfully completed at the time the **uucp** or **uux** command was executed and to initiate transfers from remote sites, the traditional approach is to use **cron** to start **uucico** at regular intervals. (See *OS/390 UNIX System Services Planning* for more information about using **cron** to start uucico. It contains information on creating crontabs.)

uucico has two modes: slave mode and master mode.

- In slave mode, **uucico** receives requests from the remote site. The **–r0** option (the default option) starts **uucico** in slave mode. **uucico** is typically started in slave mode by either the **uucpd** daemon (for remote connections via TCP/IP) or as the login shell for special UUCP user IDs that can be logged onto via serial connections. See **uucpd** and **uucp** for more information.
- In master mode, **uucico** processes requests from the local site; the **–r1** and **–s** options start **uucico** in master mode. **uucico** is typically started in master mode via **cron**. **uucp** and **uux** also invoke **uucico** in master mode by default.

If **uucico** cannot contact a remote system, it does not allow itself to run again until a specified amount of time has passed. You can specify how long the daemon should wait before trying to call each system again by setting a parameter in the Permissions file. For information on how to do this, refer to *OS/390 UNIX System Services Planning*.

If **uucico** receives a SIGQUIT, SIGTERM or SIGPIPE signal, it ends any current conversation with a remote site and exits.

Options

–f Ignores the required wait period for all remote systems and makes calls as requested.

–g *grade*
Processes outgoing work only if it is designated priority *grade* or better. *grade* is a number (0–9) or letter (A–Z, a–z), where 0 is the highest priority and z is the lowest.

r0 | –r1
Specifies the mode for **uucico** to use. **r0** (the default) specifies slave mode; **r1** specifies master mode. If you want **uucico** to call a remote system (master mode), specify **–r1**.

–s *system*
Calls the remote system. By default, **uucico** calls all defined systems.

–x *type*
Turns on debugging. *type* is a number indicating the level of detail. 0 is the least detail and 9 is the most detail. The debugging output is written to **stderr** if **uucico** is run in the foreground, or to **/usr/spool/uucp/LOGFILE** if **uucico** is run in the background by **uucpd** or by a remote **uucico** logging into a UUCP user ID.

The LOGFILE must be monitored so that it does not fill up your file system.

Examples

To call the remote site west, with debugging output sent to **stdout**:

```
uucico –r1 –x 9 –s west
```

Files

uucico uses the following files:

/usr/lib/uucp/config
UUCP configuration file. See **uucc**.

/usr/spool/uucp/LOGFILE
UUCP debug file

/usr/spool/locks
The directory containing the lock files created by **uucico**.

/usr/spool/uucp/.Status
UUCP status file

Exit Values

0 Successful completion
1 Failure due to any of the following:
- Unknown command-line option
- Not running setuid **uucp**
- Argument list too long
- Unable to open log file
- CTRL-C interrupt

Portability

X/Open Portability Guide.

Related Information

uucc, **uucp**, **uulog**, **uux**, **uuxqt**

---

uucp — Copy files between remote UUCP systems

Format

uucp [**–Ccdfjmr**] [**–g** *grade*] [**–n** *user*] [**–x** *debug_level*] [*site1*!] *file1* [*site2*!] *file2*
uucp [**–Ccdfjmr**] [**–g** *grade*] [**–n** *user*] [**–x** *debug_level*] *file...* [*site2*!] *directory*

Description

uucp copies a source file or files from one site to a target on another site. The source can be a file or group of files specified by metacharacters. The source cannot be a directory. The target can be a corresponding filename or directory.

Filenames given to **uucp** have the form:

[*site*!] *pathname*

or

[site1![site2!]... *pathname*

where *site* names the remote site. If a site is not specified, *pathname* is a pathname on your machine. *site* must be on the list of site names that **uucp** knows about. Use **uuname** to list sites that are known to **uucp**.

You can also specify multiple site names as a way of sending files to a site that your system does not have a direct connection to. Filenames that contain multiple site names are called *multinode* or *multihop* names. When processing a **uucp** request involving multihop names, only the destination name can be a multihop name. The source filename cannot be a multihop name.

Pathnames can have one of these forms:

- A full pathname.
- A pathname preceded by ~*name*/, where ~*name* is replaced on the specified site by the login directory of user *name*.
- A pathname preceded by ~/, where ~/is replaced on the specified site with the name of the public UUCP directory.

- A filename or prefix name containing the current directory on your machine as a prefix.

Destination pathnames cannot begin with exactly two slashes, which indicate an MVS filename.

If the target is a directory, you must append / to the end of the pathname to ensure that it is not treated as a file. If the / is not appended to a directory name, then the name is treated as a filename and multiple copies to that command will behave like the **cp** command. That is, each subsequent copy will overlay the previous one.

Pathnames can contain the shell metacharacters ?, *, and [ ]. The character ~ also has a special meaning, as previously described. The appropriate site expands these characters after encountering them.If the destination file is a multihop name, then the source file cannot contain shell metacharacters because **uucp** uses **uux** to handle multihop requests, and **uux** does not allow shell metacharacters in names. Be careful when using metacharacters, because expansions on other sites may occur in unforeseen ways. For more information on metacharacters and their expansion by the shell, see **sh**.

Options

–C Copies named files to the spool directory for transfer. If both this option and the **–c** option are given, this option takes precedence. This option is useful if you will be making changes to the file after running the **uucp** command and want to send the version of the file before you changed it.

–c Does not copy files to the spool directory for transfer. This is the default.

–d Makes all necessary intermediate directories to complete file transfer. This is the default.

–f Does not make intermediate directories. If **–f** is specified with the **–d** option, **–f** takes precedence.

–g *grade*
Sets the priority of this job to *grade*. It is a number (0–9) or letter (A–Z, a–z), where 0 is the highest priority and z is the lowest.

–j Passes the UUCP job ID number to standard output; this job ID can be used with **uustat** to determine the job's status or to terminate it. If **uucp** generates several job requests and several job IDs, only the last one appears.

–m Sends mail notifying you when the copy has completed. The default is to send mail only if an error occurs that prevents the copy from being made.

–n *user*
Notifies the user at the destination site when a file you have sent to the destination site has arrived. This option has no effect when you use **uucp** to get files from the remote system.

–r Queues the job to be processed later. Do not start **uucico** to begin transferring the file.

–x *debug_level*
Sets the verbosity of the debugging information to the specified debug level, which is a number, 0 or greater. Level 0 provides tersed messages while level 9 provides verbose messages. Values greater than 9 give no additional information. The default level is 0.

Options are not passed on to remote sites when the destination of your **uucp** command is a multihop name. For this **uucp** command:

```
uucp –mf file1 site1!site2!/file1
```

the **–m** and **–f** options are ignored. For multihop, **uucp** creates a **uux** request to run a **uucp** command at the next site (*site1* in our example). But because *site1* can be any system that supports **uucp**, it is possible that this particular system may not support the same options that are supported by **uucp**. For that reason, options are not passed to the **uucp** command to be run at *site1*.

To summarize the restrictions when using multihop destination names:

- Options are not passed.
- Shell metacharacters cannot be used in source file names.

Examples

1. To copy the file **/notes/memo** from your site to a file named **minutes** in the public UUCP directory on a site named `south`:

   ```
   uucp /notes/memo south!~/minutes
   ```

2. You can also copy files locally. To copy the file **resume.txt** on your site to the file **/business/resumes/november** on your site:

   ```
   uucp resume.txt /business/resumes/november
   ```

   You must have read permission on the current directory. If the directories **business/resumes** don't already exist, they are created, if you have write permission in **/**.

3. To copy the file **index** from the public UUCP directory on `north` to the current directory on the local site:

   ```
   uucp north!~/index
   ```

   You must have write permission on the current directory.

4. To copy the file **index** from the public UUCP directory on `south` to the subdirectory **south/records** in the public UUCP directory on the current site:

   ```
   uucp -f -m south!~/index '~/south/records/'
   ```

   You need to protect the tilde so the shell does not expand them to the user's home directory. If the subdirectory **south/records** does not already exist, the file copy fails. Mail is sent to you when the transfer is completed successfully.

5. You want to copy a file from your system to the site named `east`. Your system does not have a connection to `east`, but you do have a connection to `north`, and `north` has a connection to `east`:

   ```
   uucp memo north!east!~/memo
   ```

6. You want to use shell metacharacters to specify the files to be transferred to a remote site.

   In this command, the source pathname is expanded by the shell. The **uucp** command succeeds as long as there is at least one file that matches the name specification:

   ```
   uucp /mystuff/file?.[ab&]* remote!/tmp/
   ```

In this command, the source pathname is not expanded by the shell, because it cannot find any matching file. The '!' is not allowed, because **uucp** interpretes all '!' characters as delimiting system names.

```
uucp remote!/tmp/file?.[!b]* /mystuff/
```

Environment Variable

uucp uses the following environment variable:

TZ Sets the time zone used with date and time messages

Localization

uucp uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_COLLECT**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_TIME**

See Appendix F for more information.

Files

uucp uses the following files:

/usr/lib/uucp/config
UUCP configuration file generated by **uucc**.

/usr/spool/uucp/LOGFILE
Log file for **uucp** and other UUCP utilities.

/usr/spool/uucppublic
Public UUCP directory.

/usr/spool/uucp/.Sequence/*sitename*
Sequence files, one for each remote site.

Usage Note

uucp does not convert files to or from EBCDIC. If a text file is sent from an ASCII system to an MVS system, it must be converted to EBCDIC after its arrival. Similarly, if an EBCDIC text file is sent to an ASCII system, the file is not automatically converted to ASCII. The receiving user must convert the file to ASCII.

Exit Values

0 Successful completion
1 Failure due to any of the following:
- Inability to open log file
- Insufficient memory
- Ctrl-C interrupt

2 Unknown command-line option

uucp can also have partial failures, where a file is inaccessible or a host could not be determined. **uucp** returns the 1 exit value and logs the partial failure in the log

file **/usr/spool/uucp/LOGFILE**. Files that were accessible or had a known host are still queued for transfer.

Portability

X/Open Portability Guide, UNIX systems.

The **–g** option is an extension to the POSIX standard.

Related Information

uucc, **uucico**, **uulog**, **uux**

---

uucpd daemon — Invoke uucico for TCP/IP connections from remote UUCP systems

Format

uucpd [–l seconds] [–x debug_level]

Description

The **uucpd** program allows remote **uucico** programs to communicate with local **uucico** in order to perform file transfers via TCP/IP connections. **inetd** starts **uucpd** when the remote **uucico** connects to the UUCP port. **uucpd** manages the login sequence with the remote **uucico**. After successful login, it then starts **uucico** to complete the transfer.

In order for **inetd** to start **uucpd**, the **inetd** configuration file (for example, **inetd.conf**) must contain a **uucp** entry such as the following:

```
uucp  stream  tcp  nowait  OMVSKERN  /usr/sbin/uucpd  uucpd -l0
```

Options

–l seconds
Sets the login timeout value in seconds.

–x *debug_level*
Invoke **uucpd** and **uucico** with the **–x** option. *debug_level* indicates the level of detail (**0** is least detail and **9** is most detail). The **uucpd** login sequence debug output is written to a file in **TMPDIR** with a filename beginning with *uucpd* and followed by randomly generated characters. The **uucico** debug output is written to the **uucp** logfile.

Note: When using the **–x** option, the UUCP logfile should be monitored so that it does not become too large and fill up the file system.

The Permissions file provides an alternative method for setting debug for connections on a system by system basis. See the section on Permissions files in *OS/390 UNIX System Services Planning* for more information.

Usage Note

uucpd is not affected by the locale information specified in locale-related environment variables.

Exit Values

0 Successful completion
1 Failure to establish a connection with the remote system
>1
uucico failure

Portability

X/Open Portability Guide.

Related Information

inetd, **uucico**, **uucp**, **uux**

---

uudecode — Decode a transmitted binary file

Format

uudecode [*file*]

Description

uudecode decodes data that was encoded by **uuencode**. If a file is specified on the command line, **uudecode** decodes that file; if no file is specified on the command line, input is read from the standard input. Output is written to the filename that was specified when the file was encoded. See **uuencode** for more information.

uudecode automatically strips off any leading and trailing lines added by mailers.

Localization

uudecode uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion
1 Failure due to any of the following:
- Inability to open the input file
- Missing `begin` line in the input file
- Inability to create the output file
- Missing `end` line in the input file
- A file that is too short

2 Failure because of an incorrect command-line option

Portability

POSIX.2 User Portability Extension, X/Open Portability Guide. Generally found on most UNIX systems.

Related Information

uuencode

---

uuencode — Encode a file for safe transmission

Format

uuencode [*file*] *remote*

Description

When files are transmitted over a network or over phone lines, nonprintable characters (for example, control characters) may be interpreted as commands, telling the network to do something. In general, therefore, it is not safe to transmit a file if it contains nonprintable characters.

uuencode translates a binary file into a special code that consists entirely of printable characters from the POSIX portable character set. A file encoded in this way is generally safe for transmission over networks and phone lines. **uuencode** is often used to send binary files through electronic mail.

If a *file* is specified on the **uuencode** command line, **uuencode** reads that file as input. Otherwise, it reads the standard input. **uuencode** always writes the encoded result to the standard output. The encoded version of the data is about 35% larger than the original. If the size is a problem, you can shrink the file with **compress** before encoding it. The recipient must decode it and then uncompress it.

The *remote* command-line argument is the name that the file should be given after it has been transmitted to its destination. When the file reaches its destination, **uudecode** can be used to translate the encoded data into its original form. The first line of the encoded file records the file's permissions and the *remote* argument.

Because the encoded file consists entirely of printable characters, you may use a text editor to edit the file. Of course, the only things you are likely to edit are the name of the original file or the name of the remote file.

Examples

This command encodes the file **long_name.tar.Z** so it decodes with the name **arc.trz** and redirects the output to **arc.uue**:

```
uuencode long_name.tar.Z arc.trz > arc.uue
```

Localization

uuencode uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion

1 Failure because of an incorrect command-line option, or a missing command-line argument

Portability

POSIX.2 User Portability Extension, X/Open Portability Guide. Generally found on most UNIX systems.

Related Information

uudecode

---

uulog — Display log information about UUCP events

Format

uulog [**–s** *site*]

Description

uulog displays information about UUCP events, such as file transfers and remote command execution. It also displays the most recent debug output to the log. In order to use **uulog**, you must have permission to read the file `/usr/spool/uucp/LOGFILE`.

The format of the display is:

```
user ID    local_site   date/time   messagetext
```

where:

user ID Login ID of the user who requested the file transfer or requested the command be run. Entries created by **uuxqt** or by programs spawned by **uuxqt** have the ID `uucp`.

local_site Name of the local site.

date/time Date and time of the event in the form (*mm*/*dd*-*hh*:*mm*).

messagetext
Text of the log entry. The message text depends on the event being recorded; most entries are self-explanatory.

Options

If you do not specify an option, **uulog** displays the debug information for the last conversation that failed.

–s *site*
Displays information about UUCP events for this site.

Environment Variables

uulog uses the following environment variable:

TZ Sets the time zone used with date and time messages.

Localization

uulog uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_LCTIME**
- **NLSPATH**

See Appendix F, “Localization” on page 817 for more information.

Files

uulog uses the following files:

/usr/lib/uucp/config
UUCP configuration file. (See **uucc**.)

/usr/spool/uucp
UUCP spool directory.

/usr/spool/uucp/LOGFILE
Log file for **uulog** and other UUCP utilities.

Exit Values

0 Successful completion
1 Failure due to any of the following:
- **LOGFILE** could not be opened
- Could not lock **LOGFILE**

2 Unknown command-line option

Portability

X/Open Portability Guide, UNIX systems.

Related Information

uucc, **uucp**, **uux**

uuname — Display list of remote UUCP systems

Options

uuname [**–l**]

Description

uuname displays a list of all remote systems known to UUCP. Systems are listed in the order they are entered in **/usr/lib/uucp/Systems**. To display only the local system name, use the **–l** option.

Options

–l Displays the local system name.

Localization

uuname uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_LCTIME**
- **NLSPATH**

See Appendix F for more information.

File

uuname uses the following file:

/usr/lib/uucp/config
UUCP configuration file. See **uucc**.

Exit Values

0 Successful completion
1 Failure due to any of the following:
- Argument list too long
- Inability to open log file
- Insufficient memory
- Ctrl-C interrupt

2 Unknown command-line option

Portability

X/Open Portability Guide, UNIX systems.

Related Information

uucc, **uucp**, **uux**

uupick — Manage files sent by uuto and uucp

Format

uupick [**–s** *system*]

Description

uupick is an interactive shell script used to manage files in the UUCP public receive directory that were sent to you using the **uuto** command. Only those files in the receive directory are managed. (See “Files” on page 678 for a description of this directory.)

For each file or directory entry found, **uupick** prompts you with one of the following messages, depending on the type of the entry:

```
from system:  file name ?
from system:  dir name ?
```

where *system* is the name of the system that sent the file or directory, and *name* is the name of the file or directory.

To tell **uupick** how to handle an entry, issue one of the following commands:

ENTER Skips this entry and go to the next one.

***** Display the **uupick** command summary.

d Deletes the specified entry.

m [*target***]** Moves the entry to the named target directory or file. If the target does not specify an absolute pathname or no directory, the pathname is assumed to be relative to the current directory. If no directory is given, **uupick** assumes the current directory.

a [*dir***]** Moves all files from *system* to the target directory *dir*.

p Prints the contents of the entry to standard output. If the entry is a directory, **p** lists the files in the directory.

q Quits **uupick**.

CTRL-D Quits **uupick**.

!*command*
Escapes to the shell in order to perform *command*.

The tilde (~) does not stand for the public UUCP directory in pathnames specified inside **uupick**. It is interpreted by the command shell being used.

Options

–s *system* Displays only files from the system *system*.

Localization

uupick uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

- **LC_LCTIME**
- **NLSPATH**

See Appendix F for more information.

Files

uupick uses the following files:

/usr/lib/uucp/config
The UUCP configuration file, which contains the list of known systems as well as the location of the public UUCP library. See **uucc**.

/usr/spool/uucppublic
The default value for the public UUCP directory. The public UUCP directory is always the home directory of the user *uucp* as defined in the user database.

/usr/spool/uucppublic/receive/*user***/system**
When files are sent to your system using **uuto**, they are placed into **/usr/spool/uucppublic/receive/user/system**, where *user* is your login name and *system* is the name of the system that sent the files.

Usage Notes

1. **uupick** does not convert files from EBCDIC. If you receive a file from an ASCII system, you will need to convert it to EBCDIC.
2. When moving files, **uupick** does not check for files of the same name in the destination directory. These files are overwritten.
3. **uupick** is a shell script.

Portability

X/Open Portability Guide.

Related Information

uuto

---

uustat — Display status of pending UUCP transfers

Format

uustat [**–j** *jobid* | **–k** *jobid* | **–r** *jobid*]
uustat [**–m**]
uustat [**–q**]
uustat [**–s** *site*] [**–u** *user*]
uustat [**–a** [**–o** *number*] [**–y** *number*]]

Description

uustat displays reports on the progress of pending UUCP transfers. You can display the status of transfers for a particular job ID or user ID. **uustat** can also stop or restart jobs in the queue.

If you do not specify any options, it displays the status of all UUCP requests for all sites made by the current user.

Options

–a Displays the jobs queued for all users instead of only the jobs for the user issuing the command.

–j *jobid*
Displays the status of the specified job.

–k *jobid*
Stops the UUCP job identified by *jobid*. **uustat** can display the job ID of a job in the queue, when used with one of the other options. You cannot use this option with the **–q** or **–r** options.

–o *number*
Displays the jobs that are older than *number* hours.

–q Displays the latest conversation status and retry times for all sites that recently had errors, as well as a count of the jobs queued. You cannot use this option with the **–k** or **–r** options.

–m Displays the latest conversation status and retry times for all sites, as well as a count of the jobs queued. You cannot use this option with the **–k** or **–r** options.

–r *jobid*
Restarts the UUCP request specified by *jobid*. This option updates the timestamp on the file, making the request appear recent. It cannot restart jobs that have been stopped with the **–k** option. You cannot use this option with the **–k** or **–q** options.

–s *site*
Displays the status of all UUCP transfers requested for *site*.

–u *user*
Displays the status of all UUCP transfers requested by *user*.

–y *number*
Displays the jobs that are younger than *number* hours.

Output

uustat uses a variety of output formats, depending on the options specified.

If you do not specify an option, or if you specify the **–s** and **–u** options, the output is in this format, one line to every request within a work file:

```
job ID mo/dy–hh:mm  rtype  site  user  information
```

The following list explains the fields:

`job ID` Identifies the job. If a job contains more than one request, subsequent requests are displayed below the first, without a job ID.

`mo/dy–hh:mm` Time of the request.

`rtype` The request type, either `S` (for send) or `R` (for receive).

`site` The name of the remote site.

`user` The name of the user who requested the job.

`information` Describes the request. The format depends on the type of request.

For a send request, `information` has the format:

size filename

where *size* is the size in bytes of the file to be sent and *filename* is either the absolute pathname on your site, or the UNIX-style filename relative to your spool directory for the remote site.

For a receive request, *information* has the format:

`filename`

For a remote execution request (such as a request produced by **mailx**, the command to be run is displayed after any data files associated with it.

For the **–q** and **–m** options, the output is in this format:

```
site  transfersC (age)   commandsX(age)   status retry
```

where:

`site` Remote site name.

`transfersC(age)`
Number of file transfer jobs pending; if any are over one day old, the age in days of the oldest job is given in parentheses.

`commandsX(age)`
Number of pending command requests that have been received; if any are over one day old, the age in days of the oldest job is given in parentheses.

`status` Time and result of the last attempt to call this site. The `status` field shows the status of attempts made by this system to connect to other systems. When other systems call this system, this field is not updated.

`retry` Time to the next connection attempt in *hours:minutes* and the current retry count. The `retry` field is displayed only between retry attempts.

For the **–k** and **–r** options, **uustat** displays a message telling you if the attempt to stop or restart a job was successful.

Examples

1. To display all waiting UUCP requests:

   ```
   uustat
   ```

2. To display all jobs waiting for remote site `east`:

   ```
   uustat –s east
   ```

3. To stop the UUCP job associated with job ID westn0003:

   ```
   uustat –k westn0003
   ```

Environment Variables

uustat uses the following environment variable:

TZ Sets the time zone used with date and time information.

See Appendix I for more information.

Localization

uustat uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Files

uustat uses the following files:

/usr/lib/uucp/config
UUCP configuration file.

/usr/spool/uucp
UUCP spool directory, containing site-specific subdirectories and information files.

/usr/spool/uucp/*site*
Subdirectory containing queued job requests, work files, data files, and execution files for the UUCP host site.

/usr/spool/uucp/.Status/*site*
Status file for the remote UUCP host site. **uustat** queries the status file with the **–q** option.

Exit Values

0 Successful completion

1 Failure due to any of the following:
- Argument list too long
- Unable to open log file
- Log files
- Insufficient memory
- CTRL-C interrupt

2 Unknown command-line option.

Portability

X/Open Portability Guide, UNIX systems.

Related Information

uucp, **uulog**, **uuxqt**

uuto — Copy files to users on remote UUCP systems

Format

uuto [**–mp**] *file* ... *destination*

Description

uuto is a simplified method of using **uucp** to copy a file, or files, to a user on another system. *file* is a file, or files, on your system. The **destination** has the following form:

```
system!user
```

where *system* is a system known to **uucp** and **user** is the login name of a user on the remote system. You can use **uuname** to list the names of the remote system known to **uucp**. Make sure to enter the user name in the proper case. Otherwise, the recipient will not be able to use **uuto** to receive the files you have sent.

uuto sends files to the UUCP public directory on the remote system. In particular, the files are sent to the directory:

pubdir/receive/*user*/*sendsystem*

where *pubdir* is the UUCP public directory, *user* is the user's name specified in the destination, and *sendsystem* is the name of the sending system.

The recipient is notified by mail when the files arrive. If several files are sent, the recipient is notified when the last file arrives. Depending on the nature of the remote system, the recipient may move files from this directory using the **uupick** utility or by using the usual system copy commands.

Options

–m Sends the user a note when the copy is completed.

–p Places files in spool directory before transfer to remote system.

Localization

uuto uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_LCTIME**
- **NLSPATH**

See Appendix F for more information.

Files

uuto uses the following files:

/usr/lib/uucp/config
The configuration file for UUCP contains the list of systems that **uucp** knows about. This configuration file is compiled from a number of text configurations using the **uucc** utility. (See **uucc** for more information.)

/usr/spool/uucppublic
The public UUCP directory.

Usage Notes

1. **uuto** is a shell script.
2. **uuto** does not convert files to ASCII. If you use **uuto** to send a file to an ASCII system, it will have to be converted to ASCII after it has been sent.

Portability

X/Open Portability Guide, UNIX systems.

Related Information

uucc, **uulog**, **uupick**, **uustat**, **uux**

---

uux — Request command execution on remote UUCP systems

Format

uux [**–bCcjnprz**] [**–g** *grade*] [**–x** *debug_*level] [*site!*] *commandstring*

Description

uux specifies that *commandstring* be run on another site. If files required to run the command are on different sites, **uux** generates the UUCP requests to gather the files together on one site, runs the command, and sends the standard output of the command to a file on a specified site.

commandstring is any valid command for the remote site, with arguments, except that the command and any filenames can specify a site in the UUCP manner:

site1!*command site2*!*file1*

where *site1* is the name of the site where the command is to be run, and *site2* is the name of the site where *file1* is.

- If you do not specify any site names, then the command and any files are assumed to reside at your site.
- If you specify a site for the command, but not for the files, then the files are assumed to reside on the same site named for the command.
- If you specify a site for some of the files, then those files without a site name are assumed to reside on the site named.

site must be a valid site name, as listed by the **uuname** command. Specifying multiple site names, such as *site1!site2!command* or *site1!site2!file* is not allowed for **uux**

Pipes of commands are valid, but only the first command in a pipeline can have a site name. All other commands in the pipeline take place on the site specified for the first command.

Filenames can have one of these forms:

- A full pathname.

- A pathname preceded by ~*name*/, where ~*name* is replaced on the specified site by the login directory of user *name*.
- A pathname preceded by ~/, where ~/ is replaced on the specified site with the name of the public UUCP directory.
- A filename or prefix name containing the current directory on your machine as a prefix.

Unlike arguments to **uucp**, pathnames cannot contain the shell metacharacters ?, *, and [ ].

Nonlocal filenames must be unique within the command, or the command fails. This is because nonlocal files are copied to a working directory on the remote site; if the filenames are not unique, one overwrites another.

If the command fails, you are notified by electronic mail.

Options

–b Mails input back to the user. The contents of **stdin** are sent back to the user if the command fails.

–C Copies named files to the spool directory for transfer. If both this option and the **–c** option are given, this option takes precedence. This option is useful if you will be making changes to the file after running the **uux** command and want to send the version of the file before you changed it.

–c Does not copy files to the spool directory for transfer. This is the default.

–g *grade*
Sets the priority of the job to **grade**. It is a number (0–9) or a letter (A–Z, a–z), where 0 is the highest priority and z is the lowest.

–j Passes the UUCP job ID number to standard output. This job ID can be used with **uustat** to determine the job's status or to terminate it. If the **uux** request generates several job IDs, only the last is shown.

–n Does not send mail if the command fails.

–p Uses standard input of **uux** as the standard input for the specified command. The input is stored in a temporary file that is passed to the command when it runs.

–r Queues the job to be processed later. Do not start **uucico** to begin transferring the file.

–x *debug_level*
Sets the verbosity of the debugging information to *debug_level*, which is a number that is 0 or greater. Level 0 provides terse messages while level 9 provides verbose messages. Values greater than 9 give no additional information. The default level is 0.

–z Returns notification of success to the user who issued the **uux** command.

Commands on remote sites are actually run by **uuxqt** in its own directory, **/usr/spool/uucp/.Xqtdir**.

Special Characters

The command string passed to **uux** can use the shell metacharacters <, >, **;**, and |. If any of these characters are not valid for the command interpreter on the destination system, the command fails.

More complex redirection, such as 2>, is not handled by **uux** because the 2 is interpreted as a parameter to the preceeding command). Only the simple metacharacters listed above are allowed.

To escape a filename or quoted string, use parentheses. Parentheses pass the filename to the command on the remote site without special interpretation by **uux**. For example, the following command will not do what you expect because “hello” will be treated as a file unless enclosed in parentheses.

```
uux "Remote!echo hello >test.out"
```

The correct way to enter that command is:

```
uux "Remote!echo (hello) >test.out"
```

Examples

1. Suppose that a neighboring site, south, has a program called **laser** for printing and formatting documents. You have execute permission for **laser**. To print the file **inventor.y** in south's public UUCP directory using south's **laser** program:

   ```
   uux south!laser '~/inventor.y'
   ```

   The tilde needs protection from shell expansion.

   To print the file **report.001** in your public UUCP directory:

   ```
   uux south!laser !~/report.001
   ```

2. Suppose you have execute permission for **uucp** on south. To request that south use **uucp** to copy the file **index** from its public UUCP directory to west, a neighbor of south:

   ```
   uux south!uucp \(~/index\) \(west!~/\)
   ```

   The arguments ~/index and west!~/ are not interpreted by **uux** because of the parentheses. The backslashes are necessary to escape the parentheses on the OS/390 shell.

Security

uux is potentially a security risk to your system. UUCP minimizes the risk by allowing you to specify the commands that can be run by each remote site. (See the section on Permissions files in *OS/390 UNIX System Services Planning* for more information.)

For electronic mail, each remote site must be able to execute a mail routing agent on your site. Further permissions can be granted at your discretion.

Localization

uux uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**

See Appendix F for more information.

Files

uux uses the following files:

/usr/lib/uucp/config
UUCP configuration file

/usr/spool/uucp/*site*
Subdirectory containing queued job requests, work files, data files, and execution files for the UUCP host site.

/usr/spool/uucp/LOGFILE
Log file for **uux** and other UUCP utilities.

/usr/spool/uucp/.Sequence/*sitename*
Sequence file containing the 4-digit sequence number of the last job queued. If **uux** requires a sequence number, it is based on the value in this file. If this file does not exist, **uux** creates it with the sequence number `0000`. *sitename* is the name of a remote site; each remote site has its own sequence number.

Exit Values

`0` Successful completion

`1` Failure due to any of the following:

- Argument list too long
- Inability to open log file
- Insufficient memory

`2` Unknown command-line option

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The **–g**, **–p**, **–r**, and **–x** options are extensions to the POSIX standard. They are retained for compatibility with other UNIX UUCP implementations.

Related Information

uucc, **uucico**

---

uuxqt daemon — Carry out command requests from remote UUCP systems

Format

uuxqt [**–c** *command_name*] [**–s** *site*] [**–x** *debug_level*]

Description

The **uuxqt** daemon carries out the command requests made on other sites by mail programs, news programs, or by the **uux** command.

uuxqt is automatically started after **uucico** completes. Additionally, **cron** can be used to start **uuxqt** at predetermined times.

Options

–c *command_name*
Processes only requests to run *command_name.*

–s *site*
Runs only commands requested by *site.*

–x *debug_level*
Sets the verbosity of the debugging information to *debug_level*, which is a number, 0 or greater. Level 0 provides terse messages while level 9 provides verbose messages. Values greater than 9 give no additional information. The default level is 0.

Examples

To run all of the commands requested by the remote site `north`:

```
uuxqt –s north
```

Usage Notes

1. The **uuxqt** command is a security risk on all sites, because it allows outside access to your computer. UUCP limits the danger by setting execute permissions for every site in the configuration file. See the section on Permissions files in *OS/390 UNIX System Services Planning* for more information.
2. **uuxqt** checks the command requests from each site against the list of allowed commands and either runs them or sends a mail message that says:

   ```
   Permission denied
   ```

Localization

uuxqt does not use localization environment variables.

Files

uuxqt uses the following files:

/usr/lib/uucp/config
UUCP configuration file

/usr/spool/uucp/.Xqtdir
This file contains permissions for UUCP sites.

/usr/spool/uucp/.Sequence/*sitename*
Sequence file containing the four-digit sequence number of the last job queued. If **uuxqt** requires a sequence number (for example, to mail a message), it is based on the value in this file. If this file does not exist,

uuxqt creates it with the sequence number 0000. *sitename* is the name of a remote site; each remote site has its own sequence file.

/usr/spool/uucp/*site*
Subdirectory containing commands from the UUCP host site (as well as all work files and data files associated with *site*). The format of the execute files is described in **uucp**.

/usr/spool/uucp/.Xqtdir
Working directory for **uuxqt**. All required files are copied here before **uuxqt** runs a command.

Exit Values

0 Successful completion
1 Failure because of any of the following:
- Argument list too long
- Unable to open log file
- Insufficient memory
- Ctrl-C interrupt

2 Unknown command-line option

Portability

X/Open Portability Guide

Some UUCP systems produce execute files with command lines that are not supported by **uuxqt**.

Related Information

uucc, **uucp**, **uux**

---

vi — Use the display-oriented interactive text edit or

Format

vi [**–elRrsv**] [**–c** *command*] [**–t** *tag*] [**–w** *size*] [*file ...*]
vi [**–elRrsv**] [**+***command*] [**–t** *tag*] [**–w** *size*] [*file ...*]

These symbols are used throughout this command description:

Ctrl-L followed by a single letter indicates the control character transmitted by holding down the Ctrl key and the letter key at the same time.

BACKSPACE indicates the real backspace key. This may differ from the Ctrl-H key.

ENTER indicates the ENTER key, which is labeled RETURN on some keyboards.

ESCAPE indicates the Escape key.

INTERRUPT indicates the break key; often Ctrl-C.

→ indicates the right arrow key.

← indicates the left arrow key.

↓ indicates the down arrow key.

↑ indicates the up arrow key.

Description

vi has two components: a screen editor (**vi**), and a line editor (**ex**). Each has a different set of commands. You can invoke the line editor from within the screen editor. Conversely, you can invoke the screen editor from within the line editor.

In the screen editor, you are in either *command mode* or *insert mode*. In command mode, every character you type is immediately interpreted as a command. In insert mode, every character you type is added to the text that you are editing.

There are two ways to start your session in **ex** mode:

- Invoke the command under the name **ex**.
- Invoke it under the name **vi** but specify the **–e** option.

Similarly, there are two ways to start your session in **vi** mode:

- Invoke it under the name **ex** but specify the **–v** option.
- Invoke the command under the name **vi** (without specifying **–e**).

vi and **ex** work on text files. If a file contains the NUL character (value `.0` or `\0`), it is turned into the value `0x7F`. The newline character is interpreted as a line delimiter. Each line is limited to a maximum length of {LINE MAX}–1 bytes, including the newline. Any lines exceeding that length are truncated at that length. If the last line in the file does not end in a newline, a newline is added. In all those cases, **vi** marks the file as modified and displays a message.

Note: **vi** is available only if you login to the shell with the **rlogin** command. It is not available if you login with the OMVS command.

Options

–c *command*
: Runs *command* before displaying any text on the screen. *command* is any **ex** command. You can specify multiple **ex** commands by separating them with an or-bar (|) and enclosing them in quotes. The quotes ensure that the shell does not interpret the | as a pipe character. For example:

```
–c 'set all | ver'
```

–e Invokes **ex**.

–l Sets lisp mode. The (and) commands use blocks of lisp code as their context rather than sentences.

–R Sets the **READONLY** variable, preventing the accidental overwriting of files. Any command that writes to a file requires the ! suffix.

–r Tries to recover all files specified on the command line after a system or editor crash. If you do not specify any files, **vi** displays a list of all recoverable files.

–s Turns on *quiet mode.* This tells the editor not to print file information messages, thus allowing **ex** to be used as a filter. Because the file isn't displayed, the editor doesn't read the value of the **TERM** environment variable. This

option also keeps **ex** from reading any startup files (**.exrc** or the file specified by **EXINIT**).

–t *tag*
Searches for a tag in the same way that you use with the **ex tag** command (described later).

–v Puts the editor into **vi** mode.

–w *size*
Sets the option variable `window` to *size*. See “Set Option Variables” on page 710 for more information.

+command
Is an obsolete version of the **–c** option.

Current Position Pointer

The *current position marker* indicates a position in the text that is currently being edited (or has just been edited). In **ex** mode, the current position pointer is just the line number of the line being edited. In **vi** mode, the pointer gives this line number plus the position of the cursor within the line. The line indicated by the current position pointer is always on the screen.

Display Conventions

vi uses three display conventions:

- **vi** displays the input for search commands (/ and ?), **ex** commands (:), and system commands (!) on the bottom line of the screen. Error and informational messages also appear on this line.
- If the last line in the file is above the bottom of the screen, screen lines beyond the end of the file are displayed with a single ~ character in column one.
- In certain infrequent circumstances (usually involving lines longer than the width of the screen), **vi** is unable to fill the display with complete lines. In this case, one or more screen lines are shown with a single `@` character in column one.

These lines are not part of the file content and should be ignored.

vi Command Summary

vi commands can be divided into several categories:

- *Scrolling commands* adjust the position of text on the screen. The current position pointer only changes if the current line is scrolled off the screen. For example, **Ctrl-E** scrolls the text on the screen up one line. The cursor remains pointing to the same text that it was pointing to, unless that text is moved off the screen.
- *Movement commands* move the cursor in the file. For example, the character **j** moves the cursor down one line and the screen is scrolled only if necessary. There are two types of movement commands:
  - Absolute movement commands move the cursor, regardless of the nature of the surrounding text. For example, **j** always moves the cursor down one line.

    Absolute movement commands are listed in “Absolute Movement Commands” on page 692.

  - Context-dependent movement commands move move the cursor based on the nature of the text; for example, **w** moves the cursor to the beginning of the next word, so it must look at the text to determine where the next word begins.

    Context-dependent movement commands are listed in “Context-Dependent Movement Commands” on page 693.

- *Text insertion commands* let you add new text to the existing text. They are listed in “Text Insertion Commands” on page 697.
- *Manipulation commands* let you change the text that is already in the file. They are listed in “Object Manipulator Commands” on page 695.

Scrolling Commands

vi scrolling and movement commands can be preceded by a decimal integer that serves as a *count*, as in:

```
[count] command
```

count means different things with different commands. If you type *count*, it is not displayed anywhere on the screen.

Ctrl-B Scrolls text back by a page, (that is, a screen), less two lines. The cursor is placed on the bottom line of the screen. *count* specifies a number of pages to scroll. The default value for *count* is 1.

Ctrl-D Scrolls text onto the bottom of the screen. The current position pointer moves forward the same amount in the text, which means that the cursor stays in the same relative position on the screen. If *count* is given, the screen scrolls forward by the given number of lines; this number is used for all future **Ctrl-D** and **Ctrl-U** commands (until a new *count* is given). The default scrolling amount is half the screen.

Ctrl-E Scrolls a new line onto the bottom of the screen. The current position pointer is not changed unless the current line scrolls off the top of the screen; then the pointer is set to the top line. If *count* is given, the screen scrolls forward the given number of lines. The default value for *count* is 1.

Ctrl-F Scrolls text forward a page (that is, a screen), less two lines. The cursor is placed on the top line of the screen. *count* specifies the number of pages to scroll. The default value for *count* is 1.

Ctrl-U Scrolls text onto the top of the screen. The current position pointer moves backward the same amount in the text, which means that the cursor stays in the same relative position on the screen. *count* operates as for **Ctrl-D**. The default scrolling amount is half the screen.

Ctrl-Y Scrolls a new line onto the top of the screen. The current position pointer is not changed unless the current line scrolls off the bottom of the screen; then the pointer is set to the bottom line. If *count* is given, the screen scrolls backward the given number of lines. The default value for *count* is 1.

[*n***] z [***m***]** *type* Redraws the screen in a window of *m* lines. *type* determines the position of the current line. If *type* is the newline character, the current line is placed at the top of the window. If *type* is a period

(**.**), the current line is placed in the middle of the window. If *type* is a minus sign (–), the current line is placed at the bottom of the window. If *n* is given, the current position pointer is first set to that absolute line number; then the screen is positioned according to *type*. If you omit *n*, it defaults to the current line. If you omit *m*, it defaults to `window`. (See "Set Option Variables" on page 710.)

Absolute Movement Commands

All the following movement commands except **m**, **0**, ^, `, and ´ can be preceded by *count* to repeat the movement that many times.

G Moves to the absolute line number specified as *count*. As a special case, if *count* is zero or is not specified, the cursor is moved to the last line of the file.

h Moves the cursor one position to the left.

BACKSPACE Moves the cursor one position to the left.

← Moves the cursor one position to the left.

Ctrl-H Moves the cursor one position to the left.

↓ Moves the cursor to the next line at the same column on the screen. Scroll the screen one line if needed.

j Moves the cursor to the next line at the same column on the screen. Scroll the screen one line if needed.

Ctrl-J Moves the cursor to the next line at the same column on the screen. Scroll the screen one line if needed.

Ctrl-N Moves the cursor to the next line at the same column on the screen. Scroll the screen one line if needed.

k Moves the cursor to the previous line at the same column on the screen. Scrolls the screen up one line if needed.

↑ Moves the cursor to the previous line at the same column on the screen. Scrolls the screen up one line if needed.

Ctrl-P Moves the cursor to the previous line at the same column on the screen. Scrolls the screen up one line if needed.

l Moves the cursor one position to the right.

→ Moves the cursor one position to the right.

SPACE Moves the cursor one position to the right.

m Records the current position pointer under a mark name. A *mark name* is a single lowercase letter, given immediately after the **m**. For example, the command **ma** records the current location of the current position pointer under the name **a**.

0 (Zero) Moves the cursor to the first character of the current line.

+ Moves the cursor to the first nonblank character on the next line. Scroll the screen one line if needed.

Ctrl-M Moves the cursor to the first nonblank character on the next line. Scroll the screen one line if needed.

– Moves the cursor to the first nonblank character on the previous line. Scrolls the screen up one line if needed.

| Moves the cursor to the column number specified as *count*. This is a screen column number, not a character offset. If a doublebyte character occupies column positions 5 and 6, the command **6|** moves the cursor to the character that includes column 6.

If *count* is greater than the length of the current line, **vi** moves the cursor to the last character on the line. If the column indicated is spanned by a tab, **vi** moves the cursor to the first character after the tab.

Moves the cursor to the first nonblank character of the current line.

$ Moves cursor forward to the end of a line. *count* specifies the number of lines, including the current line, to move forward.

When followed by a mark name, moves the cursor to the position that has been associated with that name. The position is set by the **m** command. A grave character followed by another grave character moves the cursor to the previous context. The previous context is typically the last place where you made a change. More precisely, the previous context is set whenever you move the cursor in a nonrelative manner.

' Similar to the grave ( ) character, except that the cursor is set to the first nonblank character on the marked line.

Context-Dependent Movement Commands

vi defines a *word* as:

- A sequence of letters, digits, and underscores delimited at both ends by characters that are not letters, digits, or underscores; the beginning or end of a line; or the end of the editing buffer.
- A sequence of characters other than letters, digits, underscores, or white space delimited at both ends by a letter, digit, underscore, white space, the beginning or end of a line, or the end of the editing buffer.

vi defines a *fullword* as a a sequence of nonblank characters delimited at both ends by blank characters (space, tab, newline) or by the beginning or end of a line or file.

B Moves the cursor back to the first character of the current fullword. If the cursor is already at the beginning of a fullword, **vi** moves it to the first character of the preceding fullword.

b Moves the cursor back to the first character of the current word. If the cursor is already at the beginning of a word, **vi** moves it to the first character of the preceding word.

E Moves the cursor forward to the end of a fullword. If the cursor is already at the end of a word, **vi** moves it to the last character of the next fullword.

e Moves the cursor forward to the end of a word. If the cursor is already at the end of a word, **vi** moves it to the last character of the next word.

F*c* Searches backward in the line for the single character *c* and positions the cursor on top of it. When *count* is given, the editor searches back for the *count* the such character.

f*c* Searches forward in the line for the single character *c* and positions the cursor on top of it. When *count* is given, the editor searches for the *count* the such character.

H Places the cursor on the first nonblank character of the top line of the screen. *count* specifies the number of lines from the top of the screen.

L Places the cursor on the first nonblank character of the bottom line of the screen. *count* specifies the number of lines from the bottom of the screen.

M Places the cursor on the first nonblank character of the middle line of the screen.

N Repeats previous **/** or **?**, but in the opposite direction.

n Repeats previous **/** or **?**.

T*c* Searches backward in the line for the character *c* and position the cursor after the character being sought. *count* searches backward for the *count* the matching character and then positions the cursor after the character being sought.

t*c* Searches forward in the line for the character *c* and position the cursor on the preceding character. *count* searches forward for the *count* the matching character and then positions the cursor on the preceding character.

W Moves forward to the start of the next fullword.

w Moves forward to the start of the next word.

(Moves back to the beginning of the previous sentence. A sentence is bounded by a period (`.`), exclamation mark (`!`), or question mark (`?`); followed by any number of closing double quotes, (`"`), closing single quotes (`'`), closing parentheses (`)`), or closing square brackets (`]`); followed by two spaces or the end of the line. Paragraph and section boundaries are also sentence boundaries; see **[[** and **{**.

If you specified the *lisp* option, a *lisp* s-expression is considered a sentence for this command.

) Moves forward to the beginning of the next sentence. See **(** for the definition of a sentence.

If you specified the *lisp* option, a *lisp* s-expression is considered a sentence for this command.

{ Moves back to the beginning of a paragraph. A paragraph begins on a blank line, a section boundary, or a text formatter macro in the `paragraphs` variable.

} Moves forward to the beginning of the next paragraph. See **{** for the definition of a paragraph.

[[Moves back to the beginning of a section. A section begins on lines starting with a form feed (**Ctrl-L**), starting with an open brace **{**, a text formatter macro in the `sections` variable, or begin or end of file.

If you specified the *lisp* option, a section boundary is also identified by a line with a leading (.

]] Moves forward to the beginning of the next section. See **[[** for the definition of a section.

If you specified the *lisp* option, a section boundary is also identified by a line with a leading (.

% Finds the balancing character to that under the cursor. The character should be one of the following characters:

[{(< >**)}].**

; Repeats the previous **F**, **f**, **T**, or **t** command.

, Repeats the previous **F**, **f**, **T**, or **t** command in the opposite direction.

/*regexp* **,** Search forward in the file for a line matching the regular expression *regexp* and position the cursor at the first character of the matching string. When used with an operator to define a text range, the range begins with the character at the current cursor position and ends with the first character of the matching string. You can specify whole lines by following *regexp* with **/+***n* or **/–***n*, where *n* is the offset from the matched line.

?*regexp* Is similar to **/**, but searches backwards in the file.

Ctrl-] Uses the word after the cursor as a tag. (For information about tag, see **ex**.)

Object Manipulator Commands

An *object manipulator command* works on a block of text. The command character is followed immediately by any kind of movement command. The object that is manipulated by the object manipulator command is the text from the current position pointer to wherever the movement command would leave the cursor.

For example, in **dL**, **d** is the object manipulator command to delete an object. It is followed by the movement command **L** which means move to the bottom line of the screen. The object manipulated by the command thus extends from the current line to the bottom line on the screen; these lines are deleted.

Normally an object extends up to, but not including, the position of the cursor after the move command. However, some movements work in a *line* mode. For example, **L** puts the cursor on the first nonblank character of the last line on the screen. If it is used in an object manipulation command, it includes the entire starting line and the entire ending line. Some other objects include the cursor position. For example, **d$** deletes up to and including the last character on a line; by itself the **$** would have placed the cursor on the final character. Repeating the command letter implies working on a line basis; thus **5dd** deletes five lines.

Objects that are deleted or otherwise manipulated have their original values placed in a *buffer*, an area of computer memory that can hold text. There are several ways this can be done:

1. You can use a named buffer. Buffers are named with single lowercase letters. To place an object in a buffer, type a double quote **"** followed by the buffer name, followed by the object manipulator command, as in:

   ```
   "adL
   ```

   This deletes text from the current line to the bottom line on the screen and puts the deleted text in buffer **a**. Usually, this sort of operation overwrites the current contents of the buffer. However, if you use the same form but specify the buffer

name in uppercase, the object is appended to the current contents of the buffer. For example:

```
"AdL
```

deletes from the current line to the bottom line on the screen, and adds the deleted text to buffer **a**.

2. If you are deleting material and delete at least one full line, **vi** uses buffers numbered **1** through **9**. The first time a full line or more is deleted, the text is placed in buffer **1**. The next time, the old contents of **1** are copied to **2**, and the newly deleted text is put into **1**. In the same way, deleted text continues to be rippled through the nine numbered buffers. When text is rippled out of buffer **9**, it is gone for good.
3. In all other cases, the object manipulated goes to the unnamed buffer. For example, the unnamed buffer is used if you delete less than a line of text. The unnamed buffer is like the other buffers, but doesn't have a name.

Following are some examples of the use of buffers:

1. To delete text from the current cursor position through to the bottom of the screen and place it into buffer 1 (this will also ripple numbered buffers), enter:

   ```
   dL
   ```

2. To delete from the current cursor position through to the next position containing (but not including) the string *fred*, and place the deleted text into buffer *a*, enter:

   ```
   "ad/fred/+0
   ```

3. To delete the current word and place it into an unnamed buffer, enter:

   dw

The following section lists the object manipulator commands.

c Deletes the object and enters insert mode for text insertion after the current cursor position. If less than one line is changed, a dollar sign ($) is placed on the final character of the object and typing goes directly over top of the current object until the dollar sign ($) is reached. Additional text is inserted, with the existing text shifting to make room for the new text.

d Deletes the object.

y Moves the object to the appropriate buffer; the source is not changed. This can be used to duplicate or copy objects.

< Shifts the object left by the value of the variable `shiftwidth`. This operator always works on a line basis. This command replaces all leading blanks and tabs required for the new indent amount. *count* shifts *count* lines.

> Shifts the object right by the value of the variable `shiftwidth`. This operator always works on a line basis. This command replaces all leading blanks and tabs required for the new indent amount. *count* shifts *count* lines.

! Filters the object through an external command. After typing the object, the command line opens up for a system command which is parsed in the same manner as the **ex** system command (**:!**). This operator then invokes the given command and sends the entire object on a line basis to that command. The object is then deleted and the output from the command replaces it. For example, **1G!Gsort** moves to the first line of the file; then takes all the text

from the first line to the last line and runs it through the **sort** command. The output of **sort** then replaces the original text.

Object Manipulator Abbreviations

To make things easier, the following shorthand commands are equivalent to the shown object manipulations. Each can be preceded by *count* or by a buffer name to save the manipulated text.

C Changes to the end of the current line. This is equivalent to the **c$** command.

D Deletes to the end of the current line. This is equivalent to the **d$** command.

s Substitute the character. This is equivalent to the **cl** command.

S Substitute the line. This is equivalent to the **cc** command.

x Deletes the current character. This is equivalent to the **dl** command.

X Deletes the previous character. This is equivalent to the **dh** command.

Y Yanks the current line. This is equivalent to the **yy** command.

Text Insertion Commands

Text insertion commands let you add new text to existing text.

A Enters insert mode at end of line. This is equivalent to the **$a** command.

a Enters insert mode after the current cursor position.

I Enters insert mode before first nonblank character on line. This is equivalent to the **i** command.

i Enters insert mode before the current cursor position.

O Opens up a new line before the current line and enters insert mode on it.

o Opens up a new line after the current line and enters insert mode on it.

R Replaces characters on the screen with characters typed up to the next ESC. Each character typed overlays a character on the screen. The newline character is an exception; it is simply inserted and no other character is replaced. While you are doing this, the screen may not correspond exactly to the contents of the file, because of such things as tabs. The screen is updated when you leave insert mode.

r Replaces the character under the cursor with the next character typed. When *count* is given, *count* characters following the cursor to the new character are changed. If *count* is given and the newline character is the replacement character, *count* characters are deleted (as usual) and replaced with a single newline character, not *count* newlines.

Miscellaneous Commands

J Joins *count* lines together. If you do not specify *count*, or *count* is less than 2, **vi** uses a *count* of 2, joining the current line and the next line. This command supplies appropriate spacing: one space between words, two spaces after a period, and no spaces at all when the first character of the line is a). When a line ends with white space, **vi** retains the white space, does not add any further spaces, and then appends the next line.

p Same as **p** except that text is pasted before the cursor instead of after it.

P Put buffer contents before the cursor. Also called a *paste* operation. If preceded by quote *buffername* (for example, **"b**), the contents of that buffer are used; otherwise the contents of the unnamed buffer are used. If the buffer was created in **ex** mode, the contents of the buffer are inserted before the current line. If the buffer was created in **vi** mode, the contents are inserted before the cursor. As a special case, if a paste operation is repeated with the period (**.**) command and it used a numbered buffer, the number of the buffer is incremented. Thus, **"1p ...**, pastes in the contents of buffer 1 through buffer 6; in other words the last six things that were deleted are put back.

Q Switches to **ex** mode. You leave **vi** mode and the **ex** prompt is shown on the bottom line of the screen.

U Undoes all changes to current line. As soon as you move off a line or invoke an **ex** command on the line, the original contents of the line are forgotten and **U** is not successful.

u Undoes last change. If repeated, you undo the undo (that is, go back to what the text was before the undo). Some operations are treated as single changes; for example, everything done by a global **G** is undone with undo.

ZZ Writes the file out, if changed, and then exits.

. Repeats the last command. Any command that changes the contents of the file can be repeated by this command. If you do not specify *count* with the **.** command, **vi** uses the *count* that was specified for the command being repeated.

~ Toggles the case of the character under the cursor and moves the cursor right by one. This command can be preceded by *count* to change the case of *count* characters.

& Repeats the previous **ex** substitute command, using the current line as the target. Flags set by the previous command are ignored. Equivalent to the **ex** command **&**.

: Invokes a single **ex** command. The editor places the cursor on the bottom line of the screen and displays a colon (**:**) to prompt for input. You can then type one or more **ex** commands; when you press ESC or a RETURN, the line you have entered is passed to **ex** and executed there.

@ Invokes a macro. When the next character is a letter from **a** through **z**, **vi** treats it as the name of a buffer. The contents of that buffer are treated as input typed to **vi**. The text of a macro may contain an **@** calling another macro. A macro may call itself, provided it is invoked at the end of the macro (tail recursion). Such a macro executes forever or until an error occurs or the INTERRUPT key is pressed. A macro that invokes itself at the beginning (head recursion) loops until it runs out of memory. A **vi** error terminates all currently executing macros. All changes made during a macro call are treated as a unit and may be undone with a single **u** command.

= Reindents the specified line as though they were set via lisp and autoindent-set, if the *lisp* option was specified.

Ctrl-G Displays the current pathname, current line number, total number of lines in the file, and the percentage of the way through the file. This is equivalent to the **ex** command **file**.

Ctrl-L Redraws the screen assuming another process has written on it. This should never happen unless a filter **!** command writes to the screen rather than the standard output.

Ctrl-R Redraws the screen, removing any deleted lines flagged with the `@` convention.

Ctrl-Z Stops the editor and returns you to system level. You can return to the editor with the **fg** command; however, when you resume a **vi** session in this way, all of the session's buffers are empty. The **jobs** command lists all the stopped **vi** jobs. The amount of available memory limits the number of **vi** sessions that may be stopped at one time (see **fg** and **jobs**).

Ctrl- Switches to editing the alternate file (see **ex** for an explanation of **write**). If you attempt this and you have not written out the file since you made the most recent change, **vi** does not switch to the alternate file.

Insert Mode Commands

The object manipulation command **c**, and the text insertion commands [**AaIiOoRr**] put **vi** into INSERT mode. In this mode, most characters typed are inserted in the file. The following characters have special meaning.

Ctrl-D Decrements the *autoindent* for the current line by one level. This is only relevant if the variable **autoindent** is on.

Ctrl-H Deletes the last typed character. The character is not removed from the screen; however it is no longer in your file. When you backspace over characters, new text overwrites the old ones. You are permitted to backspace to the start of the current line regardless of where you started to insert text. (This is not true of some other versions of **vi**.)

BACKSPACE
Deletes the last typed character. The character is not removed from the screen; however it is no longer in your file. When you backspace over characters, new text overwrites the old ones. You are permitted to backspace to the start of the current line regardless of where you started to insert text. (This is not true of some other versions of **vi**.)

Ctrl-J Ends the current line and starts a new one.

Ctrl-M Ends the current line and starts a new one.

RETURN
Ends the current line and starts a new one.

Ctrl-Q Inserts the following character literally, instead of using its special meaning. You could use this to escape, say, the ESC character itself. It is impossible to insert a Ctrl-J or the null character in your line.

Ctrl-V Inserts the following character literally, instead of using its special meaning. You could use this to escape, say, the ESC character itself. It is impossible to insert a Ctrl-J or the null character in your line.

Ctrl-T Increments the autoindent for the current line by one level. This is only relevant if the variable `autoindent` is on.

Ctrl-W Deletes the word preceding the cursor and blanks. Although the characters are not removed from the screen, they are no longer in your file.

Ctrl-@ If this is the first character typed after entering insert mode, the previously typed insert mode contents are repeated. After this, you exit insert mode. Only up to 256 characters from the previous insertion are inserted.

ESC Leaves insert mode.

INTERRUPT
Leaves insert mode.

ex Command Mode

vi enters **ex** command mode if the program is invoked with the **–e** option or if the **Q** command is issued from **vi**. You can issue a single **ex** command from **vi** using the **:** command.

An **ex** command takes the general form:

[*address-list*] [[**command**] [**!**] [*parameters*]]

Each part is optional and may be invalid for some commands. You can specify multiple commands on a line by separating them with an or-bar |.

address-list
Commands can take zero, one, or two addresses. The address % is a short form to indicate the entire file. You can omit any or all of the addresses. In the command descriptions to follow, the addresses shown are the addresses that the commands use by default. Possible default addresses are:

[.,.] Indicates a two-address line range defaulting to the current line.

[1,$] Indicates a two-address line range defaulting to the entire file.

[.+1] Indicates a single address defaulting to the next line.

address An address refers to a line in the text being edited. An address can be an expression involving the following forms:

. The value of dot; that is, the current line.

n A line number indicating an absolute line in the file; the first line has absolute line number 1.

$ The last line in the file.

+[*n***]** *n* lines forward in the file. If you omit *n*, it defaults to 1.

–[*n***]** *n* lines backward in the file. If you omit *n*, it defaults to 1.

'*x* The value of the mark *x*.

/*pat***/** Search for regular expression *pat* forward from the current line.

?*pat***?** Search for regular expression *pat* backwards from the current line.

Thus:

```
/pattern/+3
++
100
```

are three addresses: the first searches for a pattern and then goes three lines further; the second indicates two lines after dot; and the third indicates the 100th line in the file.

command

The **command** is a word, which can be abbreviated. Characters shown in square brackets are optional. For example:

a[**ppend**]

indicates that the **append** command can be abbreviated to simply **a**.

! Some commands have a variant; this is usually toggled with an exclamation mark (!) immediately after the command.

parameters

Many **ex** commands use parameters to allow you to specify more information about commands. Common parameters include:

buffer

Specifies one of the named areas for saving text. For more information, see the description of buffers on page 695.

count

Is a positive integer, specifying the number of lines to be affected by the command. If you do not specify *count*, it defaults to 1.

file Is the pathname for a file. If *file* includes the **%** character, **vi** replaces that character with the pathname of the current file. If *file* includes the **#** character, **vi** replaces that character with the pathname of the alternate file. If you do not specify a file, the default is the current file.

flags

Indicate actions to be taken after the command is run. It can consist of leading plus (+) and minus (–) signs to adjust the value of the current line indicator, followed by **p**, **l**, or **#** to print, list, or number a line. Thus:

```
.+5 delete 6 ++#
```

deletes starting five lines down from dot; six lines are deleted; the current line indicator is set to the following line, then incremented by two; and that line is printed with its line number.

Regular Expressions and Replacements

Many **ex** commands use regular expressions when searching and replacing text. A *regular expression* (indicated by *pat* in the command descriptions) is used to match a set of characters.

A regular expression consists of a string of normal characters that exactly match characters in a line. These can be intermixed with special characters (known as *metacharacters*), which allow matching in some special manner. Metacharacters can themselves be matched directly by preceding them with the backslash (\) character. If the variable **magic** is turned off, all but two of the metacharacters are disabled; in this case, the backslash character must precede them to allow their use as metacharacters. See Appendix C, “Regular Expressions (regexp)” on page 797 for examples.

Summary of Regular Expressions

Matches the start of a line. This is only a metacharacter if it is the first character in the expression.

$ Matches the end of a line. This is only a metacharacter if it is the last character in the expression.

. Matches any single character.

***** Matches zero or more occurrences of the previous expression.

\< Matches the empty string preceding the start of a word. A word is a series of alphanumeric or underscore characters preceded by and followed by characters that are not alphanumeric or underscore.

\> Matches the empty string following the end of a word. A word is a series of alphanumeric or underscore characters preceded by and followed by characters that are not alphanumeric or underscore.

[*string***]** Matches any of the characters in the class defined by *string*. For example, **[aeiouy]** matches any of the vowels. You can put a range of characters in a class by specifying the first and last characters of the range, with a hyphen (-) between them. For example, in ASCII **[A–Za–z]** matches any upper or lowercase letter. If the first character of a class is the caret (^), the class matches any character not specified inside the square brackets. Thus, in ASCII **[a–z_][0–9]** matches a single alphabetic character or the underscore, followed by any nonnumeric character.

\(...\) A set of characters in the pattern can be surrounded by escaped parentheses. See "Summary of Replacement Patterns" for a discussion of the *n* replacement pattern. This is not affected by the setting of **magic**.

~ Matches the replacement part of the last substitute command.

A *replacement* (indicated by *repl* in the command descriptions) describes what to put back in a line for the set of characters matched by the regular expression.

Summary of Replacement Patterns

& Is replaced by the entire string of matched characters.

~ Is replaced by the entire replacement pattern from the last substitute.

*****n* Is replaced by the string that matched the *n*th occurrence of a **\(...\)** in the regular expression. For example, consider:

```
s/\([a-zA-Z]*\)our/\1or/
```

The **\1** represents the string that matched the regular expression **\([a-zA-Z]*\)**. Thus the previous command might change the word `colour` to `color`.

\u Turns the next character in the replacement to uppercase.

\l Turns the next character in the replacement to lowercase.

\U Turns the following characters in the replacement to uppercase.

\L Turns the following characters in the replacement to lowercase.

\E, \e Turns off the effects of **\U** or **\L**.

ex Commands

You can enter these commands as shown in **ex** mode. In **vi** mode, they must be preceded by the colon (**:**) character.

ab[breviate] *lhs rhs*
Indicates that the word *lhs* should be interpreted as abbreviation for *rhs*. (See "Context-Dependent Movement Commands" on page 693 for the definition of *word*.) If you enter *lhs* surrounded by white space in **vi** INSERT mode, it is automatically changed into *rhs*. If you do not specify any arguments for the **ab** command, it displays the abbreviations that are already defined. Abbreviated names cannot contain **#** or any other form of punctuation.

[.] a[ppend][!]
Enters **ex** INSERT mode. Text is read and placed after the specified line. An input line consisting of one period (**.**) leaves INSERT mode. If you specify an address of zero, text is inserted before the first line of the file. The current line indicator points to the last line typed.

If an exclamation mark (!) is specified, the `autoindent` option is toggled during input. This command cannot be invoked from **vi** mode.

ar[gs] Displays the current list of files being edited. The current file is shown enclosed by square brackets.

cd[!] *path*
Changes the current directory to *path*. If you omit *path*, **cd** sets the current working directory to the directory identified by the **HOME** variable. If *path* is a relative pathname, **cd** searches for it using the directories specified in the **CDPATH** variable. If path is –, then **cd** changes to the previous working directory. If you modified the buffer since the last write, **vi** displays a warning message. You can override this behavior by including the exclamation mark (!).

[.,.] c[hange][!] [*count***]**
Deletes the line range given and then enters INSERT mode. If an exclamation mark (!) is specified, `autoindent` is toggled during input. You cannot invoke this command from **vi** mode.

chd[ir][!] [*path***]**
Same as **cd**.

[.,.] co[py] *addr* **[***flags***]**
Copies the line range given after *addr*. If *addr* is zero, the lines are inserted before the first line of the file. The current line indicator points to the last line of the inserted copied text.

[.,.] d[elete] [*buffer***] [***count***] [***flags***]**
Deletes the specified line range. After the line range is deleted, the current line indicator points to the line after the deleted range. A *buffer* can be specified as a letter **a**–**z**. If so, deleted lines are saved in the buffer with that name. If an uppercase letter is specified for *buffer*, the lines are appended to the buffer of the corresponding lowercase name. If no buffer name is given, deleted lines go to the unnamed buffer.

e[dit] [!] [+*line***] [***file***]**
Begins a new editing session on a new file; the new file replaces the old file on the screen. This command is usually invalid if you have modified the contents of the current file without writing it back to the file. Specifying

an exclamation mark (!) goes on to start a new session even you have not saved the changes of the current session.

You can specify *line* as either a line number or as a string of the form */regexp* or *?regexp* where *regexp* is a regular expression. When *line* is a line number, the current line indicator is set to the specified position. When it has the form */regexp*, **vi** searches forward through the file for the first occurrence of *regexp* and sets the current line indicator to that line. *?regexp* is similar to */regexp* except that **vi** searches through the file backwards. If you omit *line* and do not specify a file, the value of the current line indicator does not change. Otherwise, if a file is specified, the current line indicator is set to either the first or last line of the buffer, depending on whether the command was issued in **vi** or **ex** mode.

ex[!] [*+line***] [***file***].**
Begins a new editing session on a new file; the new file replaces the old file on the screen. This command is usually invalid if you have modified the contents of the current file without writing it back to the file. Specifying an exclamation mark (!) goes on to start a new session even you have not saved the changes of the current session.

You can specify *line* as either a line number or as a string of the form */regexp* or *?regexp* where *regexp* is a regular expression. When *line* is a line number, the current line indicator is set to the specified position. When it has the form */regexp*, **vi** searches forward through the file for the first occurrence of *regexp* and sets the current line indicator to that line. *?regexp* is similar to */regexp* except that **vi** searches through the file backwards. If you omit *line* and do not specify a file, the value of the current line indicator does not change. Otherwise, if a file is specified, the current line indicator is set to either the first or last line of the buffer, depending on whether the command was issued in **vi** or **ex** mode.

f[ile] [*file***]**
Changes the current file name to *file* and marks it **[Not edited]**. If this file exists, it cannot be overwritten without using the exclamation mark (!) variant of the **write** command.

[1,$] g[lobal] [!] */pat/* **[***commands***]**
Matches *pat* against every line in the given range. On lines that match, the *commands* are run. If the exclamation mark (!) variant is set, the *commands* are run on lines that do not match. This is the same as using the *vi* command.

The **global** command and the **undo** command cannot occur in the list of *commands*. A subsequent **undo** command undoes the effect of the entire **global** command. In **ex** mode, multiple command lines can be entered by ending all but the last with a backslash (\). Commands that will take input are permitted; the input is included in the command list, and the trailing period (.) can be omitted at the end of the list. For example:

```
g/rhino/a\
hippo
```

appends the single line `hippo` to each line containing `rhino`. *delim* is an arbitrary, nonalphabetic character. The total length of a global command list is limited (see “Limits” on page 719).

[.] i[nsert][!]
Enter **ex** INSERT mode, reads text and places it before the specified line. Otherwise, this is identical to the **append** command. This command cannot be entered from **vi** mode.

[.,.+1] j[oin][!] [*count***] [***flags***]**
Joins together the lines of text within the range. Unless an exclamation mark (`!`) is specified, all white space between adjacent joined lines is deleted. Two spaces are provided if the previous line ended in a period, no spaces if the joined line begins with a opening parenthesis, and one space otherwise.

[.] k *x* Synonymous with the **mark** command.

[.,.] l[ist] [*count***] [***flags***]**
Displays the line range in a visually unambiguous manner. This command displays tabs as **I**, and the end of lines as **$**. The only useful flag is **#**, for line numbering. The current line indicator points to the last line displayed.

map[!] *lhs rhs*
This defines macros for use in **vi**. The *lhs* is a string of characters; whenever that string is typed exactly, **vi** behaves as if the string *rhs* had been typed. If *lhs* is more than one character long, none of the characters are echoed or acted on until either a character is typed that isn't in the *lhs* (in which case all the characters up to that point in the *lhs* are run) or the last character of *lhs* is typed. If the variable `remap` is set, *rhs* itself can contain macros. If the flag `!` is specified, the map applies within **vi** INSERT mode; otherwise it applies to command mode. A **map** command with no arguments lists all macros currently defined.

[.] ma[rk] *x*
Records the specified line as being marked with the single lowercase letter *x*. The line can then be addressed at any point as '*x*.

[.,.] m[ove] [*addr***] [***flags***]**
Moves the specified line range after the *addr* given. If *addr* is zero, the text is moved to the start of the file. The current line indicator is set to the last line moved.

n[ext][!] [+*command***] [***file ...***] [ext][!] [+***command***] [***file ...***]**
Begins editing the next file in the file list (where the file list was either specified on the command line or in a previous **next** command). If the current file has been modified since the last write, **ex** usually prevents you from leaving the current file. You can get around this by specifying an exclamation mark (`!`). If the `autowrite` is set, the current file is written automatically and you go to the next file. If a list of files is specified, they become the new file list. If necessary, expressions in this list are expanded. Thus:

```
next *.c
```

sets the file list to all the files in the current directory with names ending in `.c` (typically C source files).

[.,.] nu[mber] [*count***] [***flags***] [.,.] # [***count***] [***flags***]**
Displays the specified line range with leading line numbers. The current line indicator points to the last line displayed.

[.] o[pen] [*pat***] [***flags***]**
Enters open mode, which is simply **vi** mode with a oneline window. If a match is found for the regular expression *pat* in the specified line, then the cursor is placed at the start of the matching pattern.

pre[serve]
Saves the current buffer in a form that can later be recovered using the **–r** option on the **recover** command. **vi** sends you mail telling you that you can recover this file and explains how to do so.

[.,.] p[rint] [*count***] [***flags***]**
Displays the specified line range. The current line indicator points to the last line displayed.

[.] pu[t] [*buffer***]**
Pastes deleted or yanked lines back into the file after the given line. If no buffer name is given, the most recently changed buffer is used.

Because the **edit** command does not destroy buffers, you can send that command in conjunction with **put** and **yank** to move text between files.

q[uit][!] Exit from **vi** or **ex**. If the current file has been modified, an exclamation mark (`!`) must be used or you cannot exit until you write the file.

[.] r[ead] [!][*file***]**
Reads the contents of *file* and inserts them into the current file after the given line number. If the line number is 0, the contents of the given file are inserted at the beginning of the file being edited. If the current filename is not set, a *file* must be given, and it becomes the current file name; otherwise, if a *file* is given, it becomes the alternate file name. If the *file* begins with an exclamation mark (`!`), then it is taken as a system command. Pipes are used to read in the output from the command after the given line number.

rec[over] *file*
Attempts to recover *file* if it was saved as the result of a **preserve** command or a system or editor crash. If you do not specify *file*, this command displays a list of all recoverable files.

rew[ind][!]
Rewinds the file argument list back to the beginning and starts editing the first file in the list. If the current file has been modified, an exclamation mark (`!`) must be specified; otherwise, you cannot leave the current file until you have written it out. If `autowrite` is set, the current file is written out automatically if it needs to be.

se[t] [*parameter-list***] [t] [***parameter-list***]**
Assigns or displays the values of option variables, If you do not specify a parameter list, **set** displays all the variables with values that have changed since the editing session started. If the parameter **all** is specified, **ex** displays all variables and their values. You can use the parameter list to set or display each of many variable values. Each argument in the list is a variable name; if it is a Boolean variable, the value is set on or off depending on whether the name is prefixed by `no`. Non-Boolean variables alone in an argument are a request to display their values. A Boolean variable's value can be displayed by following the name by a question mark (`?`). You can set numeric or string variables with:

name=value

In a string variable, spaces must be preceded by a backslash. For example:

```
set readonly? noautowrite shell=/bin/sh
```

shows the value of the **readonly** flag, sets **noautowrite**, and sets the **shell** to `/bin/sh`.

```
set report report=5
```

shows the value of the `report` variable, and then set the value to 5. See "Set Option Variables" on page 710 for more details.

sh[ell] Invokes a child shell. The environment variable **SHELL** is used to find the name of the shell to run.

so[urce] *file*
Runs editor commands from *file*. A file being executed with **source** can contain **source** commands of its own.

st[op] Suspends the editor session and returns to system level. For more information, see the description of the **vi** command **Ctrl-Z** on page 699.

[.,.] s[ubstitute] [/*pat*/*repl*/**] [***options***] [***count***] [***flags***]**
Searches each line in the line range for the regular expression *pat* and replaces matching strings with *repl*.

Normally, **ex** only replaces the first matching string in each line. If *options* contains **g** [global], all matching strings are changed.

If *options* contains **c** [confirm], **ex** first prints the line with caret (^) characters marking the *pat* matching location; you can then type `y` if you want **ex** to go ahead with the substitution. *pat* cannot match over a line boundary; however in **ex** mode, *repl* can contain a newline, escaped by a preceding backslash (\). See Appendix C, "Regular Expressions (regexp)" on page 797 for full information on both *pat* and *repl*. If there is no *pat* or *repl*, **ex** uses the most recently specified regular expression or replacement string. You can use any nonalphabetic character in place of the slash (/) to delimit *pat* and *repl*.

su[spend]
This is synonymous with the **stop** command.

[.,.] t *addr* **[***flags***]**
This is synonymous with the **copy** command.

ta[g][!] *tagname*
Looks up *tagname* in the files listed in the variable `tags`. If the tag name is found in a tags file, that file also contains the name of the file that contains the tag and a regular expression required within that file to locate that tag. If the given file is different from the one you are currently editing, **ex** normally begins editing the new file. However, if you have modified the current file since the last time it was written out, **ex** does not start editing a new file unless the **tag** command contains an exclamation mark (!). If `autowrite` is on, the current file is automatically written out and the new file read in. When the new file is read in, the regular expression from the tags file is invoked with the `magic` variable off.

Tag names are typically used to locate C function definitions in C source files. The first step is to create a `tags` file using the **ctags** command. After you do this, you can use the **ex tag** command to look up a particular function definition and go directly to that definition in the file that contains it.

All characters in tag names are significant unless the variable `taglength` is nonzero; in this case, only the given number of characters are used in the comparison.

una[bbreviate] *lhs*
The abbreviation *lhs* previously created by **abbreviate** is deleted.

u[ndo] Undoes the last change or set of changes that modified the buffer. Globals and **vi** macros are both considered as single changes that can be undone. A second **undo** undoes the **undo** restoring the previous state. The **edit** command cannot be undone, because it cleans up the temporary file which is used to maintain undo information. You cannot undo operating system commands and commands that write output to the file system.

unm[ap][!] *lhs*
Deletes the *lhs* map. If the flag `!` is used, this applies to the insert mode maps; otherwise it applies to the command mode maps.

[1,$] v /*pat***/** *commands*
This is a synonym for the **global** command with the `!` flag; that is, a global for all nonmatching lines. You can use any nonalphabetic character to delimit *pat* instead of the slash (/).

ve[rsion]
Displays the current version information for **vi** or **ex**.

[.] vi[sual] [*type***] [***count***] [***flags***]**
Enters **vi** mode. If no *type* is specified, the current line is at the top of the screen. If *type* is caret (^), the bottom line of the screen is one window before the current line. If *type* is a minus sign, (–), the current line is at the bottom of the screen. If *type* is a period (`.`), the current line is in the middle of the screen.

You can use the **undo** command to undo all the changes that occurred during the **vi** command.

[1,$] w[rite][!] [>>**] [***file***]**
Writes the given range of lines to *file*. If two right angle brackets (>>) are included, the lines are appended to the current contents of the file. If the current file name is not set, a *file* must be given. This becomes the current file name. Otherwise, *file* becomes the alternate filename if it is specified. If the *file* begins with an exclamation mark (`!`), then it is taken as a system command. **vi** writes the given range to the command through a pipe.

If a *file* is given, it must not already exist. The variable **READONLY** must not be set. If a *file* is not given, the file must be edited; that is, it must be the same file as that read in. All these conditions can be overridden by using the flag `!`.

[1,$] wn[!] [>>**] [***file***]**
Similar to **write**, except that it begins editing the next file in the file list immediately afterwards (if the write is successful).

[1,$] wq[!] [>>**] [***file***]**
Similar to **write**, except that it exits the editor immediately afterwards (if the write is successful).

x[it] If you have modified the current file since the last write, performs a **write** command using the specified range and file name and then terminates.

[.,.] y[ank] [*buffer***] [***count***]**

Copies the given line range to the specified *buffer* (a letter from **a** through **z**). If a buffer is not specified, the unnamed buffer is used. Buffers are not destroyed by an **edit** command, so **yank** and **put** can be used to move text between files.

Because the **edit** command does not destroy buffers, you can use that command in conjunction with **put** and **yank** to move text between files.

[.+1]z [*type***] [***count***] [***flags***]**

Displays *count* lines. If no count is specified, **ex** uses the current value of the `scroll` variable. The lines are displayed with the given line located according to the *type*. If *type* is a plus sign (+), the editor displays the given line and a screenful after that. If *type* is a period (.), the editor displays screenful with the given line in the middle. If *type* is a minus sign (-), the editor displays a screenful with the given line at the end. If *type* is a caret (^), the editor displays the screenful before that. If *type* is an equal sign (=), the current line is centered on the screen with a line of hyphens printed immediately before and after it. The current line indicator points to the last line displayed.

[.,.] <[<...] [*count***] [***flags***]**

Shifts the line range by the value of the `shiftwidth` variable. If there are multiple left angle brackets (<), each one causes another shift. The current line indicator points to the last line displayed. If a *count* is specified, that many lines are shifted.

[.,.] >[>...] [*count***] [***flags***]**

Shifts the line range right by the value of the `shiftwidth` variable. If there are multiple right angle brackets (>), each one causes another shift. The current line indicator points to the last line displayed. If a *count* is specified, that many lines are shifted.

[*range***] !** *command*

Submits *command* to be run by the command interpreter named by the **SHELL** variable. If *range* is given, the *command* is invoked with the contents of that line range as input. The output from the *command* then replaces that line range. Thus:

```
1,$!sort
```

sorts the entire contents of the file.

Substitutions are made in *command* before it is run. Any occurrences of an exclamation mark (!) are replaced by the previous *command* line, while occurrences of percentage (%) and hash mark (#) characters are replaced with the pathnames of the current and alternate files, respectively. If any such substitutions actually take place, the new command line is displayed before it is executed. (See the **read** and **write** sections in “ex Command Mode” on page 700 for more information about the current and alternative files.)

If the file has been modified and the variable `autowrite` is on, the file is written before calling the command. If **autowrite** is off, a warning message is given.

[$] = Displays the given line number. The default line number is the last line of the file. The current line indicator is not changed.

" *a line of text*
This is a comment.

[.,.] & [*options***] [***count***] [***flags***]**
Repeats the last **substitute** command. If any *options*, *count*, or *flags* are specified, they replace the corresponding items in the previous **substitute** command.

[.,.] ~ [*options***] [***count***] [***flags***]**
Repeats the last **substitute** command. However, the regular expression that is used is the last regular expression; that is, if there has been a search, the search's regular expression is used. The simple **substitute** with no arguments, or the **&** command, uses the regular expression from the previous substitute. **substitute** with an empty regular expression uses the last regular expression, like ~. If any *options*, *count*, or *flags* are specified, they replace the corresponding items in the previous **substitute** command.

@ *buffer*
Executes each line in *buffer* as an **ex** command. If you do not specify *buffer* or if you specify a buffer named @, the last buffer executed is used.

Ctrl-D Displays the number of lines of text given by the `scroll` variable. The current line indicator points to the last line displayed.

Special Characters in ex Commands

When an **ex** command contains the percentage character (%), the character is replaced by the name of the current file. For example, if you are about to try out a macro and you are worried that the macro may damage the file, you could issue:

```
!cp % /tmp
```

to copy the current file to a safe holding place. As another example, a macro could use the percentage character (%) to refer to the current file.

When an **ex** command contains the hash mark (#), the character is replaced by the name of the alternate file. The name of the alternate file can be set with the **read** command as described previously. Thus a command like:

```
e #
```

tells **ex** to edit the alternate file. Using an alternate file can be particularly convenient when you have two files that you want to edit simultaneously. The command just given lets you flip back and forth between the two files.

Set Option Variables

Options are set with the **set** command. For example:

```
set autowrite
```

sets the **autowrite** option. For options which are flags, i.e., are not numeric, the variables can be turned off by putting **no** in front of the name in the **set** command, as in:

```
set noautowrite
```

In the following list, variables that are off by default are preceded by `no`. The minimal abbreviation of each option is shown after the comma. Default values are shown after the equal sign (=).

`autoflush, af`
When this option is set, it holds the maximum number of seconds of data a user would lose if a system crash occurs. **vi** will flush memory out to its temporary files approximately this many seconds, unless no changes have been made to the current edit buffer, or the user is sitting idle. It allows you to eventually recover a more current representation of your edit buffer (after the **exrecover** daemon and **vi** —r is run) because it intermittently updates **vi**'s temporary files which are used by the **exrecover** daemon.

Note the following:

- The default is set to 120 seconds (2 minutes).
- To turn off this option, set `autoflash` to 0.
- This option has no effect on read-only files.
- This option is different than the previous `preserve` option because it works with **vi**'s temporary files (whose location is specified by the environment variables: **TMP_VI**, **TMPDIR** or **TMP**) as opposed to recovered files found in **/etc/recover/$LOGNAME**.

`autoindent, ai`
When **autoindent** is on and you are entering text, the indentation of the current line is used for the new line. In **vi** mode, you can change this default indentation by using the control keys **Ctrl-D** (to shift left) or **Ctrl-T** to shift right. In **ex** mode, a tab or spaces can be typed at the start of a line to increase the indent, or a **Ctrl-D** can be typed at the start of the line to remove a level. **Ctrl-D** temporarily removes the ident for the current line. **0Ctrl-D** places the current line at a zero indent level, and the next line has this indent level as well.

The size of indent levels is defined by the variable `shiftwidth`.. Based on this value and the value of `tabstop`, the editor generates the number of tabs and spaces needed to produce the required indent level.

The default is `noautoident`.

`autoprint, ap`
When this option is set in **ex** mode, the current line is printed after the following commands: **copy**, **delete**, **join**, **move**, **substitute**, **undo**, **&**, **~**, <, and >. Automatic displaying of lines does not take place inside global commands.

The nofault is `autoprint`.

`autowrite, aw`
When this option is on, the current file is automatically written out if it has been changed since it was last written and you have run any of the following commands: **next**, **rewind**, **tag**, **Ctrl-** (**vi**), and **Ctrl-]** (**vi**). Using an exclamation mark (!) with any of these commands stops the automatic write.

The default is `noautowrite`.

`beautify, bf`
When this option is on, the editor discards all nonprinting characters from text read in from files.

The default is `nobeautify`.

`cdpath` Used by **cd** to find relative pathnames when changing the directory. You must delimit entries with a colon (:). If the current directory is to be included in the search, it must be indicated by a dot (`.`). `cdpath` defaults to the contents of the **CDPATH** environment variable if it exists, or to dot (`.`) if it doesn't.

`directory, dir`
The editor uses temporary files with unique names under the given directory. Any error on the temporary files is fatal.

The default is `directory=tmp`.

`edcompatible`
When this option is on, the editor attempts to make substitution commands behave in a way that is compatible with the **ed** editor. The **g** and **c** options on the substitute commands are remembered and toggled by their occurrence. The **r** option uses the last regular expression rather than the last substitute regular expression. Percentage mark (`%`) as the entire pattern is equivalent to the previous pattern.

The default is `nocompatible`.

`errorbells, eb`
When this option is on, **vi** precedes error messages with the alert character. When it is off, the editor warns you of an error by displaying a message using a standout mode of your terminal (such as reverse video).

The default is `noerrorbells`.

`exrc` When this option is on, **ex** and **vi** access any **.exrc** files in the current directory during initialization. If it is off, **ex** and **vi** ignore such files unless the current directory is the **HOME** directory.

`home` Used as the destination directory by **cd**. If no path is specified, `home` defaults to the contents of the **HOME** environment variable if it exists, or to the **vi** startup directory if it does not.

`ignorecase, ic`
When this option is on, the case of letters is ignored when matching strings and regular expressions.

The default is `noignorecase`.

`linedelete`
vi sets the line delete character automatically to the current terminal line delete character, as specified by the user. Within **vi**, you can set the line delete character with the `linedelete` variable. The value you specify is the numeric value of the line delete character. The default is `0x15`, the ASCII value for **Ctrl-U**. Another popular value is `0x18` for **Ctrl-X**.

`list` When this option is on, tabs are displayed as a caret mark (^) rather than expanded with blanks, and the ends of lines are indicated with a dollar sign ($).

The default is `nolist`.

`magic` When this option is off (`nomagic`), regular expression characters ^ \ and $ become the only ones with special meanings. All other regular expression metacharacters must be preceded by a backslash (\) to have their special meaning.

The default is `magic`.

`maxbuffers`
The number of K units (1024 bytes) of memory to be used for the editor buffers. These are allocated in units of 16K.

The default is `maxbuffers=512`, but if that is not available upon entry, this is set to the number actually obtained. At least 32K is needed. This is in addition to the code and data space required by **vi**; this may be as much as 128K. Changing **maxbuffers** has no effect.

`mesg` When this option is on, **ex** allows others to use the **write** or **talk** commands to write to your terminal while you are in visual mode. The command

```
mesg n
```

overrides this variable (see **mesg**). This option has no effect on systems not supporting **mesg**.

`number, nu`
When this option is on, line numbers are displayed to the left of the text being edited.

The default is `nonumber`.

`paragraphs`
This list of character pairs controls the movement between paragraphs in **vi** mode. Lines beginning with a period (**.**) followed by any pair of characters in the list are paragraph boundaries (for example, **.**`IP`). Such lines are typically commands to text formatters like **nroff** or **troff**.

The default is `paragraphs="IPLPPPQPP LIpplpipbp""`

`prompt` When this option is on, **ex** command mode prompts with a colon (**:**). No prompts are given if input is not being read from a terminal.

The default is `prompt`.

`pwd` This is a read-only variable. The value always refers to the current working directory, and can only be changed by the **cd** command.

`quiet` When this option is on, **vi** does not display file information messages.

The default is set by the **–s** option.

READONLY
When this option is on, **vi** does not let you write to the current file.

The default is based on the permissions of the current file. If you do not have write permission on this file, the default is **READONLY**. Otherwise, the default is set by the **–R** option.

`remap` If this option is on and a `map` macro is expanded, the expansion is reexamined to see if it also contains `map` macros.

The default is `remap`.

`report` The editor displays a message whenever you issue a command that affects more than this number of lines.

The default is `report=5`.

`restrict` All filenames are restricted to the current directory. Subcommands cannot be called. This variable is automatically set if you invoke the editor with a command that starts with the letter **r**, as in **rvi**. When the option is turned on, it cannot be turned off.

The default is `norestrict`.

`scroll` This sets the number of lines to scroll for the **z ex** and **Ctrl-D** (**ex**) commands.

The default is the value of the variable `window`, divided by two.

`sections` This list of character pairs controls the movement between sections in **vi** mode. Lines beginning with a period (`.`) followed by any pair of characters in the list are section boundaries (for example, `.SH`). Such lines are typically commands to text formatters like **nroff** or **troff**.

The default is `sections="SHNHH HU"`

`shell=,sh` This is the name of the command interpreter to be used for **!** commands and the shell command. The default value is taken from the **SHELL** environment variable.

`shiftwidth=, sw`
This sets the width of indent used by shift commands and `autoindent`.

The default is `shiftwidth=8`.

`showmatch, sm`
If this option is on and you type a closing parenthesis or closing brace in input mode, the cursor moves to the matching open parenthesis or brace. It stays there for about one second and then moves back to where you were. This lets you note the relationship between opening and closing parentheses/braces.

The default is `noshowmatch`.

`showmode` When this option is on, **vi** displays an indicator in the bottom right-hand corner of the screen if you are in Insert/Open/Change/Replace mode. If no indicator is displayed, you are in Command mode.

The default is `noshowmode`.

`tabstop` Tab stops for screen display in **vi** mode are set to multiples of this number.

The default is `tabstop=8`.

`taglength, tl`
If this variable is nonzero, tags are only compared for this number of characters.

The default is `taglength=0`.

`tags` The value of this variable should be a list of file names separated by a backslash (\) followed by a space. If there is no backslash before the space, **vi** treats the second and subsequent tags as part of an `option=value` combination. For example:

```
set tags=file1\ file2\ file3\
```

These are used by the **tag ex** command and the **Ctrl-] vi** command. The files are typically created with the **ctags** program.

The default is `tags=tags`.

`term` The value of this variable is the terminal type. The **TERM** environment variable specifies this variable's default value.

`terse` If this option is on, messages are displayed in a very abbreviated form. The default is `noterse`.

`warn` When this option is on, commands with an exclamation mark (`!`) print a warning message if the current file has been modified. No message is printed if this option is off.

The default is `warn`.

`window` This variable gives the number of text lines available in **vi** mode or the default number of lines to display for the command.

The default is given by the **–w** option. If it is not specified with the **–w** option, its value defaults to the environment variable **LINES** or the value found in the terminfo database for **TERM**.

`wrapmargin wm`
If this variable is nonzero in **vi** insert mode, when a line reaches this number of characters from the right of the screen, the current word moves down to the next line automatically; you don't have to press ENTER.

The default is `wramargin=0`.

`wrapscan, ws`
If this option is off, forward searches stop at the end of the file and backward searches stop at the beginning.

The default is `wrapscan`

`writeany, wa`
If this option is off, the editor does not let a file marked `[Not edited]` overwrite an existing file.

The default is `nowriteany`.

Editor Initialization

Initialization code consists of one or more **ex** commands that run when the editor starts up. Initialization code can be obtained in several ways:

1. If there is an environment variable named **EXINIT** with a nonnull value, it is assumed to hold initialization code. **vi** executes this code using an **ex source** command.
2. If **EXINIT** does not exist or has a null value, the editor tries to find a file named **.exrc.** If you have an environment variable named **HOME**, the value of this variable is assumed to be the name of your home directory. **vi** runs the **.exrc** file using an **ex source** command.
3. If **EXINIT** variable or the **$HOME/.exrc** file sets the option variable `exrc` and if there is a file named **.exrc** under the current directory, it is assumed to hold initialization code. **vi** runs this code using an **ex source** command.

All **.exrc** files must be owned by the same user ID that invoked the **vi** command, and must only be writable by that user ID. Typical permissions for a **.exrc** file would be 744.

The **.exrc** file is read as if it were a sequence of keystrokes typed at the beginning of an **ex** session. As a result, the contents of **.exrc** must be the same as the characters you would type if you were in **vi** or **ex**. In particular, if the input contains an unusual character (for example, a carriage return) that you would usually precede with Ctrl-V, there must be a Ctrl-V in the **.exrc** file. If you are creating an **.exrc** file with **vi**, you must type Ctrl-V Ctrl-V to put a Ctrl-V character into your initialization file, then Ctrl-V followed by the special character to put the special character into your initialization file. The **.exrc** file must show both Ctrl-V and the special character. A command specified in the **.exrc** file may be ignored (treated as a comment) by beginning that line with a double quote (") character.

Files

vi uses the following files:

/tmp Directory used for temporary files if **TMP_VI**, **TMPDIR** and **TMP** are not defined.

/tmp/VI*nnnnn.mmm*
Temporary files.

.exrc Startup file.

Environment Variables

vi uses the following environment variables:

CDPATH
Contains a list of directories to be searched.

COLUMNS
Contains the number of columns between the left and right margins (see option variable **wrapmargin**). This is also used as the horizontal screen size.

ENV Contains the pathname of a file containing KornShell commands. When you invoke **sh**, it executes this file before doing anything else.

EXINIT Contains a list of **vi** commands to be run when the editor is started up.

HOME Contains the directory to be searched for the editor startup file.

LINES Contains the number of lines in a screenful (see option variable **windows**). This is also used as the vertical screen size.

PATH Contains a list of directories to be searched for the shell command specified in the **ex** commands **read**, **write**, and **shell**.

READONLY
Specifies that files cannot be written to.

SHELL Contains the name of the command interpreter for use in **!**, **shell**, **read**, **write**, and other **ex** commands with an operand of the form **!***string*. The default is the **sh** utility.

TERM Contains the name of the terminal type.

TERMINFO
Contains the pathname of the terminfo database.

TMPDIR Contains the pathname that the shell uses as the directory for temporary files.

TMP_VI Contains a directory pathname that can be specified by an administrator as a location for **vi**'s temporary files. This is useful if the current default directory for these files (usually **/tmp**) is implemented as a TFS. In this case, all **vi**'s temporary files that the **exrecover** daemon uses for recovery would be gone after a system crash.

We recommend that this environment variable be set by a system administrator as opposed to a user setting it for their environment. If the latter occurs, and the user sets the **TMP_VI** directory to something different than what **exrecover** recognizes as **TMP_VI**, the user will need to run the **exrecover** daemon manually to allow the temporary files to be converted to the recoverable files used by **vi** (located in **/etc/recover/$LOGNAME**).

Note: A system administrator should NOT set **TMP_VI** to **/etc/recover/$LOGNAME**. Also, the administrator should not set **TMP_VI** to any directory where a pathname component is an environment variable with a user's value different than the init process's value (eg. $HOME). **vi**'s temporary files are converted into a form recoverable by **vi** when **exrecover** is run during IPL. Since **exrecover** is issued during IPL, it is owned by the init process and will therefore contain different values for certain environment variables, if those environment variables are set. Throughout the file system, there may exist some temporary files that can only be converted by **exrecover**. This conversion can be done manually by a system administrator (to recover files owned by all users) or by a single user (to recover only their own files).

Localization

vi uses the following localization environment variables:

- **LANG**
- **LC_COLLATE**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion

1 Failure due to any of the following:

- Unknown option
- No such command from open/visual
- Missing *lhs*
- Missing filename
- System does not support job control
- Write forms are **w** and **w>>**
- Internal error: bad seek pointer
- Internal error: Line out of range
- Internal error: line too long
- Nonzero address required on this command
- No lines in the buffer
- Nothing to undo

- Cannot escape a newline in global from visual
- Global command too long
- Argument list too long
- File is read only
- No previous command to substitute for **!**
- Command too long
- No previous regular expression
- Buffers are 1–9, a–z
- Line too long
- System does not support job control
- Digits required after **=**
- Nothing in buffer
- Missing *rhs*
- Too many macros
- Recursive map expansion
- Nothing to repeat
- Last repeatable command overflowed the repeat buffer
- Bad tag
- No tags file
- No such tag in tags file
- Negative address—first buffer line is 1
- Not an editor command
- Unimplemented **ex** command
- Wrong number of addresses for command
- Mark requires following letter
- Undefined mark referenced
- Global within global not allowed
- First address exceeds second
- Cannot use open/visual unless open option is set
- Regular expression \ must be followed by / or ?
- No address allowed on this command
- No more files to edit
- No current filename
- Extra characters at end of command
- Not that many lines in buffer
- Insufficient memory
- Restricted environment
- Command too long
- Trailing address required
- Destination cannot straddle source in **m** and **t**
- No filename to substitute for **%**
- No alternate filename to substitute for **#**
- Filename too long
- Too many filenames
- Argument buffer overflow
- Incomplete shell escape command
- Regular expressions cannot be delimited by letters or digits
- No previous scanning regular expression
- No previous substitute to repeat
- Cannot escape newlines into regular expressions
- Missing **[**
- Badly constructed regular expression
- No remembered regular expression
- Line overflow in substitute

- Replacement pattern contains **\d**—cannot use in regular expression
- Replacement pattern too long
- Regular expression too complicated
- Cannot escape newline in visual
- No such set option
- String too long in option assignment

Limits

- Maximum number of lines: 65279 (64K - 256 - 1).
- Length of longest line: 1K (1024) bytes including `\r\n`.
- Longest command line: 256 bytes.
- Length of filenames: 128 bytes.
- Length of string options: 64 bytes.
- Length of remembered regular expressions: 256 bytes.
- Number of **map**, **map!,** and **abbreviate** entries: 64 each.
- Number of saved keystrokes for **.** in **vi**: 128.
- Length of the *lhs* of **map**, **map!,** or **abbreviate**: 10 bytes.
- Max number of characters in a tag name: 30.
- Number of characters in a **:** escape from **vi**: 128.
- Requires 128K of memory plus the set option `maxbuffers` K of auxiliary memory. During startup, `maxbuffers` is changed to reflect available memory; at least 32K is required.
- Number of nested source files is 3.

Portability

POSIX.2 User Portability Extension, X/Open Portability Guide, UNIX systems.

Related Information

ed, **ex**, **fg**, **jobs**, **mesg**, **sed**, **talk**, **write**

For more information about **regexp**, see Appendix C.

---

wait — Wait for a child process to end

Format

wait [*pid*|*job-id ...*]

tcsh shell: **wait**

Description

wait waits for one or more jobs or child processes to complete in the background. If you specify one or more *job-id* arguments, **wait** waits for all processes in each job to end. If you specify *pid*, **wait** waits for the child process with that process ID (PID) to end. If no child process has that process ID, **wait** returns immediately.

If you specify neither a *pid* nor a *job-id*, **wait** waits for the process IDs known to the invoking shell to complete.

wait in the tcsh shell

The tcsh shell waits for all background jobs. If the shell is interactive, an interrupt will disrupt the **wait** and cause the shell to print the names and job numbers of all outstanding jobs. See “tcsh — Invoke a C shell” on page 556.

Localization

wait uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Usage Notes

wait is a built-in shell command.

Exit Values

If you specified a *job-id* that has terminated or is unknown by the invoking shell, an error message and a return code of 127 is returned. If you specified a *pid* that has terminated or is unknown to the shell, a return code of 127 is returned. If a signal ended the process abnormally, the exit status is a value greater than 128 unique to that signal. Otherwise, possible exit status values are:

`0` Successful completion

`1–126`
An error occurred

`127`
A specified *pid* or *job-id* has terminated or is unknown by the invoking shell

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

Related Information

sleep, **tcsh**

wall — Broadcast a message to logged-in users

Format

wall [*message*]

Description

wall sends a message to all logged-in users. Type each line, pressing Enter after each. After you finish typing the message, enter End-of-File or an interrupt (typically, <EscChar-D> for End-of-File or <EscChar-C> for an interrupt, where EscChar is normally the cent sign; if you use rlogin or telnet to enter the shell, you hold down the Ctrl key while you press either D or C).

You must be a superuser to ensure permission to write to all the ttys that are logged in. If you are not a superuser, then writes to all ttys will fail (except your own) and those users will not receive the message. Superusers can also get failures if the **/etc/utmpx** file does not correctly represent the users currently logged in.

Recipients of the message receive a beep announcing the message. The message is displayed in this form:

```
Broadcast Message from SWEHR@AQFT (ttyp0006) at 10:43:54 (EDT5EST)...

This is the text of the message line1.
This is line2.
```

Exit Values

`0` **wall** successfully sent the message to all users.

`1` Failure due to any of the following:

- No message was entered in response to the prompt.
- You do not have permission to write to a user's terminal.

wc — Count newlines, words, and bytes

Format

wc [**–c**|**–m**] [**–l**|**w**] [*file ...*]

Description

wc tells you how big a text document is. It counts the number of <newline>s, words, characters, and bytes in text files. If you specify multiple files, **wc** produces counts for each file, plus totals for all files.

Options

–c Prints a byte count. You cannot specify this option with **–m**.
–l Prints a <newline> count
–m Prints a character count. You cannot specify this option with **–c**.
–w Prints a word count

The order of options can dictate the order in which **wc** displays counts. For example, **wc –cwl** displays the number of bytes, then the number of words, then the number of <newline>s. If you do not specify any options, the default is **wc –lwc** (<newline> count, then words, then bytes).

A word is considered to be a character or characters delimited by white space.

Note: If you have a file containing doublebyte characters, the byte count is higher than the character count.

Localization

wc uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion
1 Failure because of an inability to open the input file
2 Failure because of an incorrect command-line option

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The way the order of options **–c**, **–l** and **–w** affects the order of display is an extension to traditional implementations of **wc**.

Related Information

awk, **ed**, **vi**

---

whence — Tell how the shell interprets a command name

Format

whence [**–v**] *name ...*

Description

whence tells how the shell would interpret each *name* if used as a command name. Shell keywords, aliases, functions, built-in commands, and executable files are distinguished. For executable files, the full pathname is given. If the executable file is a tracked alias, the string identifies it as *cached*.

Options

–v Gives a more verbose report.

Usage Notes

whence is a built-in shell command.

Localization

whence uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion
1 Command *name* could not be found
2 Failure due to an incorrect command-line argument

Portability

POSIX.2.

Related Information

command, **sh**

---

who — Display information about current users

Format

who[**–AabdHilmprsTtuw**] [*file*]
who –q[*file*]
who am Ili

Description

who displays information about users who are logged into the system. By default, the output contains the user's login name, terminal name, and the time that the user logged in. Normally, **who** consults the file **/etc/utmpx** for information, but you can use the *file* argument to specify another accounting file (such as **/etc/wtmp**).

When called as:

```
who am i
```

or

```
who am I
```

who displays your login name, terminal, and login time. This command works only in the POSIX locale.

Options

–A Displays all accounting entries.

–a Displays all types of entries. This is equivalent to specifying **–AbdHilprTtuw**.

–b Displays all entries written at system boot time.

–d Displays entries produced after the death of a process spawned from **/usr/sbin/init**.

–H Displays column headings above the output.

–i Displays idle time for users. The idle time is the *hours:minutes* since the last activity; a dot (.) means that the terminal has been used in the last minute, and the string *old* means that the terminal has not been used in more than 24 hours, or hasn't been used since boot time.

–l Displays logged-out user entries.

–m Displays information about current terminal only.

–p Displays entries for processes spawned from **/usr/sbin/init**.

–q Displays a quick list with the number of users and their names; other options are ignored.

–r Displays all run-level change entries.

–s Displays only the three fields *user name*, *terminal*, and *time of entry*.

–T Displays the state of each terminal as a plus sign (+) if the terminal allows write access to other users, and a minus sign (–) if write access is denied. **who** displays a question mark (?) if the write access cannot be determined.

–t Displays all time change entries (both old and new time).

–u Displays only entries associated with logged-in users. **who** enables this option when you do not provide any options on the command line.

–w Displays the terminal state; this indicates whether the terminal can be written to.

Files

who uses the following files:

/etc/utmpx
Current status file.

/etc/wtmp
Activity history file.

Localization

who uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_TIME**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion

2 Failure because of an incorrect command-line option, or because of too many command-line arguments.

Portability

POSIX.2 User Portability Extension, X/Open Portability Guide.

The **utmpx** file format, the options, and the output of **who** are totally compatible with UNIX System V.

The **–A**, **–a**, **–b**, **–d** **–i**, **–l**, **–p**, **–r**, **–s**, **–t**, **–w**, and **am I** options are extensions to the POSIX standard.

Related Information

See the **utmpx** file format description in Appendix H for more information.

---

whoami — Display your effective user name

Format

whoami

Description

whoami displays a user name associated with the effective user ID. To display your login name, use `who am i`.

For example, if you login as user1, then use the **su** command to change to user2:

command	**returned**
who am I	user1
whoami	user2

Exit Values

0 Successful completion

1 Incorrect command line argument

2 Error getting effective username; Displays effective UID

Related Information

who, **id**

---

write — Write to another user

Format

write *user_name* [*terminal*]

Description

write lets you send a message directly to the terminal of someone else logged in to the system.

Options

user_name
Specifies the user to whom you want to send your message.

terminal
Is an optional identifier for use when the other user is logged in on more than one terminal. The format of the terminal name is the same as returned by **who**.

Usage Notes

1. When you issue a **write** command to send a message to another user, the other user receives a message of the form:

   ```
   Message from your_name (terminal) [date] ...
   ```

   After the system establishes the connection to the other user, it sends two alert characters (usually beeps) to your terminal to tell you that it is ready to send your message. You can then type your message, which will appear on the other user's terminal. To end your message, enter end-of-file or an interrupt (typically, <EscChar-D> for end-of-file or <EscChar-C> for an interrupt, where EscChar is normally the cent sign; if you use rlogin or telnet to enter the shell, you hold down the Ctrl key while you press either D or C). When **write** receives an indication for end-of-message, it tells the other user that the message is over and breaks the connection.

   The other user can reply to your message with:

   ```
   write your_user_name
   ```

   However, if both of you are trying to write on each other's terminal at the same time, the messages may get interleaved on your screens, making them difficult to read. For two-way conversations, use **talk** instead of **write**.

2. You can add the output of a command to the material that you write. To do this, start a line with an exclamation mark (!) and put a standard system command on the rest of that line. **write** calls your shell to execute the command, and sends the standard output (**stdout**) from the command to the other user. The other user does not see the command itself or any input to the command. For example, you might write:

   ```
   Here is what my file contains:
   !cat file
   ```

3. The **mesg** command lets you refuse **write** messages. With:

   ```
   mesg n
   ```

   you can tell the system that you don't want to be interrupted by **write** messages. If people try to **write** to you, they are denied immediately; the system does not inform you about such attempts. For further details, see **mesg**.

Localization

write uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 **write** successfully wrote a message, or the intended recipient used **mesg** to refuse messages (either before you start sending a message or as you are sending the message).

1 Failure due to any of the following:

- *user_name* is not signed on
- You do not have permission to write on that user's terminal
- **write** cannot open the target terminal for writing
- The command line had an incorrect number of options

Portability

POSIX.2 User Portability Extension, X/Open Portability Guide, UNIX systems.

Related Information

mailx, **mesg**, **talk**, **who**

---

xargs — Construct an argument list and run a command

Format

xargs [**–I** *placeholder*] [**–i** *[placeholder]*] [**–L** *number*] [**–l** *[number]*] [**–n** *number*] [**–ptx**] [**–E** *[eofstr]*] [**–e** *[eofstr]*] [**–s** *size*] [*command* [*argument ...*]]

Description

The **xargs** command line typically contains the skeleton, or *template,* of another command. This template looks like a normal command, except that it lacks some arguments. **xargs** adds arguments from standard input (**stdin**) to complete the command, then runs the resulting command. If more input remains, it repeats this process.

In a doublebyte locale, some options may accept a doublebyte string as an argument. In these cases, an incorrect doublebyte string would be detected during command-line parsing.

Options

xargs gets the needed arguments from **stdin**. Different options tell how **stdin** is to be interpreted to obtain these arguments.

–I *placeholder*
Specifies that each line in **stdin** be considered as a single argument. The *placeholder* following the **–I** is a string that can appear multiple times in the command template. **xargs** strips the input line of any leading white-space characters and inserts it in place of the *placeholder* string. For example, with:

```
xargs –I '{}' mv dir1/'{}' dir2/ '{}'
```

stdin should consist of lines giving names of files that you want moved from `dir1` to `dir2`. **xargs** substitutes these names for the `{}` placeholder in each place that it appears in the command template.

When **xargs** creates arguments for the template command, no single argument can be longer than 255 characters after the input has replaced the placeholders. The **–x** option is automatically in effect if **–I** or **–i** is used. If you omit the *placeholder* string, it defaults to the string `{ }`. Thus we could write our preceding example as:

```
xargs –i mv dir1/ '{}' dir2/ '{}'
```

In a doublebyte locale, *placeholder* may contain doublebyte characters.

–i *placeholder*
Behaves like **–I**, except that the *placeholder* is optional. If you omit the *placeholder* string, it defaults to the string { }. Thus, the previous example could be written as:

```
xargs –i mv dir1/ '{} ' dir2/'{ } '
xargs –i /{}/ mv dir1/ '{}' dir2/ '{}
```

–L *number*
Specifies that **xargs** read *number* lines from **stdin** and concatenate them into one long string (with a blank separating each of the original lines). **xargs** then appends this string to the command template and runs the resulting command. This process is repeated until **xargs** reaches the end of **stdin** if there are fewer than *number* lines left in the file the last time the command is run, **xargs** just uses what is there.

With this option, a line must contain at least one nonblank character; blank lines are skipped and do not count toward the number of lines being added to the template. **xargs** considers a line to end at the first newline character, unless the last character of the line is a blank or a tab; in this case, the current line is considered to extend to the end of the next non-empty line.

If you omit the **–L** or **–I** option, the default number of lines read from **stdin** is `1`. The **–x** option is automatically in effect if **–I** is used.

–l *number*
Acts like the **–L** option, but the *number* argument is optional. *number* defaults to 1.

–n *number*
Specifies **xargs** is to read the given number of arguments from **stdin** and put them on the end of the command template. For example:

```
xargs –n 2 diff
```

obtains two arguments from **stdin**, appends them to the **diff** command, and then runs the command. It repeats this process until **stdin** runs out of arguments. When you use this option, **xargs** considers arguments to be strings of characters separated from each other by white-space characters (blanks, horizontal tabs, or newlines). Empty lines are always skipped (that is, they don't count as arguments). If you want an input argument to contain blanks or horizontal tabs, enclose it in double quotes or single quotes. If the argument contains a double-quote character ("), you must enclose the argument in single quotes. Conversely, if the argument contains a single quote (') (or an apostrophe), you must enclose the argument in double quotes. You can also put a backslash (\) in front of a character to tell **xargs** to ignore any special meaning the character may have (for example, white-space characters, or quotes).

xargs reads fewer than *number* arguments if:

- The accumulated command line length exceeds the *size* specified by the **–s** option (or {LINE_MAX} if you did not specify **–s**)
- The last iteration has more than zero, but less than *number* arguments remaining

If you do not specify the **–n** option, the default number of arguments read from **stdin** is 1.

Typically, an **xargs** command uses exactly one of the options just described. If you specify more than one, **xargs** uses the one that appears last on the command line. If the command has none of these options, **xargs** keeps reading input until it fills up its internal buffer, concatenating arguments to the end of the command template. When the buffer is full, **xargs** runs the resulting command, and then starts constructing a new command. For example:

```
ls | xargs echo
```

prints the names of files in the working directory as one long line. When you invoke **xargs** this way, the total length of all arguments must be less than the size specified by the **–s** option.

If no command template appears on the command line, **xargs** uses **echo** by default. When **xargs** runs a command, it uses your search rules to find the command; this means that you can run shell scripts as well as normal programs.

The command you want to execute should be in your search **$PATH**.

xargs ends prematurely if it cannot run a constructed command or if an executed command returns a nonzero status.

If an executed command is a shell program, it should explicitly contain an **exit** command to avoid returning a nonzero by accident; see **sh** for details.

Other Options

You can use the following options with any of the three main options.

–E *[eofstr]*

Defines *eofstr* to represent end-of-file on **stdin**. For example:

```
–E :::
```

tells **xargs** that `:::` represents the end of **stdin**, even if an input file continues afterward. If there is no **–E** or **–e** option, a single underscore (_) marks the end of the input.

In a doublebyte locale, *eofstr* may contain doublebyte characters.

–e *[eofstr]*
Acts like **–E** but the *eofstr* argument is optional. If you specify **–e** without *eofstr*, there is no end-of-file marker string, and _ is taken literally instead of as an end-of-file marker. **xargs** stops reading input when it reaches the specified end-of-file marker or the true end of the file.

–p Prompts you before each command. This turns on the **–t** option so that you see each constructed command before it is run. Then **xargs** displays `?...`, asking if you really want to run this command. If you type a string beginning with `y`, **xargs** runs the command as displayed; otherwise, the command is not run, and **xargs** constructs a new command.

–s *size*
Sets the maximum allowable size of an argument list to *size* characters (where *size* is an integer). The value of *size* must be less than or equal to the system variable LINE_MAX; if you omit the **–s** option, the default allowable size of an argument list is LINE_MAX. The length of the argument list is the length of the entire constructed command; this includes the length of the command name, the length of each argument, plus one blank for separating each item on the line.

–t Writes each constructed command to **stderr** just before running the command.

–x Kills **xargs** if it creates a command that is longer than the size given by the **–s** option (or {LINE_MAX} is **–s** was not specified). This option comes into effect automatically if you specify **–i** or **–l**.

Example

The following displays filenames in three columns:

```
ls | xargs –n 3 echo
```

Localization

xargs uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion of all commands
1–125 Failure due to any of the following:
- **xargs** could not assemble a command line
- One or more invocations of *command* returned a nonzero exit status.
- Some other error occurred

126 **xargs** found *command* but could not invoke it
127 **xargs** could not find *command*

Restriction

The maximum length of a constructed command is LINE_MAX bytes.

Portability

POSIX.2, X/Open Portability Guide, UNIX systems.

The **–e**, **–E**, **–i**, **–I**, **–l**, **–L**, and **–p** options are extensions of the POSIX standard.

Related Information

echo, **find**, **sh**

---

yacc — Use the yacc compiler

Format

yacc [**–dhlmqtv**] [**–b** *file.prefix*] [**–D** *file.h*] [**–o** *file.c*] [**–p** *prefix*] [**–P** *yyparse.c*] [**–V** *stats*] *gram.y*

Description

yacc converts a context-free LALR(1) grammar found in the input file *gram.y* into a set of tables that together with additional C code constitute a parser to recognize that grammar. If you specify an input file named -, **yacc** reads the grammar from the standard input. By default, **yacc** places the parsing tables and associated C code into the file **y.tab.c**.

You can find detailed information on writing parsers using **yacc** in *OS/390 UNIX System Services Programming Tools*.

Options

–b *file_prefix*
Uses *file_prefix* instead of y as the prefix for all output filenames. For example, **yacc** would name the parsing table *file_prefix***.tab.c** rather than **y.tab.c**.

–D *file.h*
Generates the file *file.h*, which contains the constant definition statements for token names. This lets other modules of a multimodule program access these symbolic names. This is the same as **–d**, except that the user specifies the include filename.

–d Generates the file **y.tab.h**, which contains the constant definition statements for token names. This lets other modules of a multimodule program access these symbolic names. This is the same as **–D**, except that the user does not specify the header filename.

–h Displays a brief list of the options and quits.

–l Disables the generation of **#line** statements in the parser output file, which are used to produce correct line numbers in compiler error messages from *gram.y*.

–m Displays memory usage, timing, and table size statistics on the standard output.

–o *file.c*
Places the generated parser tables into *file.c* instead of the default **y.tab.c**.

–P *yyparse.c*
Indicates that the C parser template is found in the file **yyparse.c**. If you do not specify this option, this parser template is located in **/etc/yyparse.c**.

–p *prefix*
By default, **yacc** prefixes all variables and defined parameters in the generated parser code with the two letters `yy` (or `YY`). In order to have more than one **yacc**-generated parser in a single program, each parser must have unique variable names. **–p** uses the string *prefix* to replace the `yy` prefix in variable names. *prefix* should be entirely in lowercase because **yacc** uses an uppercase version of the string to replace all `YY` variables. We recommend a short prefix (such as `zz`) because some C compilers have name length restrictions for identifiers. You can also set this identifier with a **%prefix** directive in the grammar file.

–q Disables the printing of warning messages.

–t Enables debugging code in the generated parser. **yacc** does not normally compile this code because it is under the control of the preprocessor symbol `YYDEBUG`.

This option is therefore equivalent to either setting `YYDEBUG` on the C compiler command line or specifying **#define YYDEBUG** statement in the first section of the grammar.

–V *stats*
Writes a verbose description of the parsing tables and any possible conflicts to the file *stats*.

This is the same as **-v** except that the user specifies the filename.

–v writes a verbose description of the parsing tables and any possible conflicts to the file **y.output**.

Files

yacc uses the following files:

y.output
Default statistics file when you specify **-v**.

y.tab.c
Default file for the generated parser.

y.tab.h
Default header file when you specify **–d**.

/etc/yyparse.c
Default parser template.

Localization

yacc uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_CTYPE**
- **LC_MESSAGES**
- **LC_SYNTAX**
- **NLSPATH**

See Appendix F for more information.

Usage Notes

In a doublebyte environment, **yacc** can use doublebyte characters, although this practice is possibly nonportable.

1. Comments and rule names can contain doublebyte characters.
2. Doublebyte characters can be used in symbolic token names (generated by *%token* statements only if the C preprocessor and compiler will interpret them correctly. Symbolic token names are converted directly into **#define** statements and are then interpreted by the preprocessor and the compiler.
3. You can use doublebyte characters as literal token definitions (a doublebyte character surrounded by apostrophes), although this will generate a warning and may create a conflict with an assigned token name.

Exit Values

`0` Successful completion

`1` Failure due to any of the following:

- *number* rules never reduced
- Reduce-reduce conflict
- Shift-reduce conflict
- *NAME* should have been defined earlier
- `\000` not permitted
- EOF encountered while processing *%union*
- EOF in string or character constant
- EOF inside comment
- Use of $*number* not permitted
- Nonterminal *number*, entry at *number*
- Action does not terminate
- Bad *%start* construction
- Bad syntax in *%type*
- Bad syntax on $<*ident*> clause
- Bad syntax on first rule
- Inability to find parser
- Inability to open input file
- Inability to open table file
- Inability to open temporary file
- Inability to open **y.output**
- Inability to place goto

- Inability to reopen action temporary file
- Default action causes potential type clash
- EOF before %}
- *%prec syntax* not permitted
- *nnn* construction not permitted
- Comment not permitted
- Option not permitted
- Incorrect or missing ' or "
- Incorrect rule: missing semicolon, or |?
- Internal **yacc** error
- Incorrect escape, or incorrect reserved word
- Item too big
- More than *number* rules
- Must return a value, since *LHS* has a type
- Must specify type for *name*
- Must specify type of $*number*
- Newline in string.
- No space in action table
- Nonterminal *symbol* not permitted after *%prec*
- Nonterminal *symbol* never derives any token string
- Nonterminal *symbol* not defined
- Optimizer cannot open temporary file
- Out of space in optimizer
- Out of state space
- Redeclaration of precedence of *symbol*
- Redeclaration of type of *symbol*
- Syntax error
- Token incorrect on *LHS* of grammar rule
- Too many characters in ID's and literals
- Too many look-ahead sets
- Too many nonterminals
- Too many states
- Too many terminals
- Type redeclaration of nonterminal *symbol*
- Type redeclaration of token *symbol*
- Unexpected EOF before %
- Unterminated < ...> clause
- Working set overflow
- **yacc** state or nolook error

Messages

Possible error messages include:

No input file
You did not specify a grammar file **gram.y** on the command line.

No parser produced
Analysis of the input grammar shows that it contains inaccessible or ungrounded nonterminal symbols. Check the preceding report and revise the grammar.

Out of memory at size bytes
The specified grammar is too complex to process within the memory resources of the current configuration.

Limits

yacc dynamically allocates all internal tables so that grammar size and complexity are limited only by available memory.

Portability

POSIX.2, POSIX.2 C-Language Development Utilities Option, X/Open Portability Guide, UNIX systems.

The **–D**, **–h**, **–m**, **–p**, **–q**, **–S**, **–s**, and **–V** options are extensions of the POSIX standard.

Related Information

lex

OS/390 UNIX System Services Programming Tools

---

zcat — Uncompress and display data

Format

zcat –DVv [*file ...*]

Description

zcat takes one or more compressed data files as input. The data files should be compressed with the **compress** command. If no data files are specified on the command line, **zcat** reads standard input (**stdin**). You can also pass **stdin** to **zcat** by specifying **–** as one of the files on the command line.

zcat uncompresses the data of all the input files, and writes the result on standard output (**stdout**). **zcat** concatenates the data in the same way **cat** does.

The names of compressed input files are expected to end in **.Z**. If a specified input file name does not end in this suffix, **zcat** automatically adds the **.Z**. For example, if the command line specifies file **abc**, **zcat** looks for **abc.Z**.

zcat is equivalent to:

```
uncompress -c
```

Options

–D Uncompresses files that were compressed using the dictionary option of **compress**.

–V Prints the version number of **uncompress** that **zcat** calls.

–v Prints the name of each file as it is uncompressed.

Localization

zcat uses the following localization environment variables:

- **LANG**
- **LC_ALL**
- **LC_MESSAGES**
- **NLSPATH**

See Appendix F for more information.

Exit Values

0 Successful completion

1 Failure due to any of the following:

- Unknown command line option
- File is not in compressed format
- File was compressed with a number of bits **zcat** cannot handle
- There is no space for decompress tables
- The compressed file is corrupt

Portability

UNIX systems.

Related Information

cat, **compress**, **uncompress**

Part 3. TSO/E Commands

This part describes the Time Sharing Option Extensions (TSO/E) OMVS command that you use to invoke the shell and the TSO/E commands that you can use to work with the OpenMVS file system.

The OMVS command invokes the OS/390 shell. You can enter the OMVS command from TSO/E or from the ISPF command processor panel.

You can use the **man** command to view manual descriptions of TSO/E commands. To do this, you must prefix all commands with **tso**. For example, to view a description of the MOUNT command, you would enter:

```
man tsomount
```

The commands for working with the file system are:

- BPXBATCH
- ISHELL
- MKDIR
- MKNOD
- MOUNT
- OBROWSE
- OCOPY
- OEDIT
- OGET
- OGETX
- OHELP
- OPUT
- OPUTX
- OSHELL
- OSTEPLIB
- UNMOUNT

You can enter these TSO/E commands from:

- TSO/E
- The Interactive System Productivity Facility (ISPF) command processor panel (typically, option 6 on the ISPF menu).

  Option 6 is usually preferable, because it does not convert into uppercase the commands that you enter. You should enter a TSO/E command from an ISPF panel that does not convert all the parameters into uppercase; some panels, such as the main ISPF panel, convert what you enter into uppercase. OS/390 UNIX System Services is case-sensitive.
- The shell

Notes:

1. The relative pathname is relative to the working directory (usually the **HOME** directory) of the TSO/E session, not the shell session.
2. You should use absolute pathnames when entering any TSO/E commands.
3. Avoid using spaces or single quotes within pathnames.

© Copyright IBM Corp. 1996, 2000

BPXBATCH — Run shell commands, shell scripts, or executable files

Format

```
BPXBATCH SH|PGM shell_command|program_name
```

Description

BPXBATCH makes it easy for you to run, from your TSO/E session, shell scripts or OS/390 C executable files that reside in hierarchical file system (HFS) files.

With BPXBATCH, you can allocate the MVS standard files **stdin**, **stdout**, and **stderr** as HFS files for passing input or saving output. If you do allocate these files, they must be HFS files. You can also allocate MVS data sets or HFS text files containing environment variables (**stdenv**). If you do not allocate them, **stdin**, **stdout**, **stderr**, and **stdenv** default to **/dev/null**. Allocate the standard files using the data definition PATH keyword options, or standard data definition options for MVS data sets, for**stdenv**. For more information about BPXBATCH, see Appendix D, “Running Shell Scripts or Executable Files under MVS Environments” on page 803.

In addition to using BPXBATCH, a user who wants to perform a local spawn without being concerned about environment set-up (that is, without having to set specific environment variables which could be overwritten if they are also set in the user's profile) can use BPXBATSL. BPXBATSL provides users with an alternate entry point into BPXBATCH, and forces a program to run using a local spawn instead of fork/exec as BPXBATCH does. This ultimately allows a program to run faster.

BPXBATSL is also useful when the user wants to perform a local spawn of their program, but also needs subsequent child processes to be fork/exec'ed. Formerly, with BPXBATCH, this could not be done since BPXBATCH and the requested program shared the same environment variables.

BPXBATSL is an alias of BPXBATCH.

Parameters

SH|PGM
Specifies whether BPXBATCH is to run a shell script or command, on the one hand, or a OS/390 C executable file located in an HFS file, on the other hand.

If neither SH nor PGM is specified, BPXBATCH assumes that the shell is to be started in order to run the shell script allocated by **stdin**.

SH
Specifies that the shell designated in your TSO/E user ID's security product profile is to be started and is to run shell commands or scripts provided from **stdin** or the specified program_name.

If SH is specified with no program_name information, BPXBATCH attempts to run anything read in from **stdin**.

SH is the default.

PGM
Specifies that the specified program_name is to be run as a called program from a shell environment. If you specify PGM, you must also specify program_name. BPXBATCH creates a process for the program to run in and then calls the program. If possible, the **HOME** and **LOGNAME** environment variables are set when the program is run.

program_name
Specifies the shell command name or the HFS pathname for the shell script or OS/390 C executable file that you want to run. program_name can also contain option information. program_name is in uppercase and lowercase letters.

When PGM and program_name are specified and the specified program name does not begin with a slash character (/), BPXBATCH prefixes the user's initial working directory information to the program pathname.

Examples

1. You want to run the shell script you specify with **stdin**.

```
ALLOCATE FILE(STDIN) PATH('/stdin_file_pathname')
PATHOPTS(ORDONLY)
BPXBATCH SH
```

2. You want to run the program **/usr/bin/payroll.**

```
BPXBATCH PGM /usr/bin/payroll
```

3. You want to run the script **shellscriptA** and put its output into the file **a.out** in a temporary directory.

```
BPXBATCH SH /u/usr/joe/shellscriptA > /tmp/a.out
```

ISHELL — Invoke the ISPF shell

Format

```
ISHELL
```

Note: An alias of ISHELL is:

```
ISH
```

Description

ISHELL invokes the OS/390 ISPF shell, a panel interface that helps you to set up and manage OS/390 UNIX System Services functions.

You can use the ISHELL command to:

- List files in a directory
- Create, delete, or rename directories, files, and special files
- Browse files
- Edit files
- Copy files
- Display file attributes
- Search files for text strings
- Compare files or directories
- Run executable files
- Display the attributes and contents of a symbolic link (symlink)

- Mount and unmount a hierarchical file system (HFS)
- Create a hierarchical file system (HFS)
- Set up character special files
- Set up standard directories for a root file system
- Set up existing users and groups for OS/390 UNIX System Services access

  For more information on setting up TSO/E users, see *OS/390 UNIX System Services Planning*.

Some of these tasks require either superuser authority (such as mounting, unmounting, setting up character special files, setting up existing users and groups for OS/390 UNIX System Services access) or the RACF SPECIAL attribute (such as setting up existing users and groups for OS/390 UNIX access).

Field level and panel help are available throughout the dialog. For additional information on ISHELL, see *OS/390 UNIX System Services User's Guide* and the online help panels.

MKDIR — Make a directory

Format

```
MKDIR 'directory_name'  MODE(directory_permission_bits) STICKY|NOSTICKY
```

Description

You can use the MKDIR command to create a directory in the file system.

Parameters

directory_name
Specifies the name of the directory to be created. The name can be a relative pathname or an absolute pathname. You must enclose it in single quotes. A relative pathname is relative to the working directory of the TSO/E session (usually the **HOME** directory). Therefore, you should usually specify an absolute pathname. The name can be up to 1023 characters long. The name is case-sensitive; the system stores each character in the case entered.

All directories in the pathname prior to the specified directory must already exist. If the specified directory already exists, no new directory is created.

MODE(directory_permission_bits)
Specifies the directory permission bits as three octal numbers, from `0` to 7, separated by commas or blanks. The octal values represent read (r), write (w), and search (x) access for: user, group, and other.

User permission is the permission given to the directory owner. Group permission is the permission given to the group the owner is a member of. Other permission is the permission given to any other user.

The access indicated by each of the numbers `0`–7 is:

0 No access
1 Search (x) access
2 Write-only (w) access
3 Write and search (wx) access
4 Read-only (r) access

5 Read and search (rx) access
6 Read and write (rw) access
7 Read, write, and search (rwx) access

The default permissions set when a directory is created are 755, representing:

7 User: read, write, and search permission.
5 Group: read and search permission.
5 Other: read and search permission.

STICKY
Specifies that the sticky bit is to be set on for a directory so a user cannot remove or rename a file in the directory unless one or more of these conditions are true:

- The user owns the file
- The user owns the directory
- The user has superuser authority

NOSTICKY
Specifies that the sticky bit is to be set off in the directory. NOSTICKY is the default.

Return Codes

0 Processing successful.
12 Processing unsuccessful. An error message has been issued.

Examples

1. You want to create a directory using an absolute pathname giving read, write, and search access to the directory owner and no access to the group and other classes. The new directory name is to be **/tmp/bin**. The directory **/tmp** already exists. You enter:

   ```
   MKDIR '/tmp/bin' MODE(7,0,0)
   ```

2. You want to create a new directory under the working directory of your TSO/E session; therefore you can specify a relative pathname. You want to name the new directory **u2**, and to set it up with the default permissions (755). You enter:

   ```
   MKDIR 'u2'
   ```

MKNOD — Create a character special file

Format

```
MKNOD 'pathname'
      MAJOR(device_major_number)
      MINOR(device_minor_number)
      MODE(file_permission_bits)
```

Description

MKNOD creates a character special file in a file system.

Note: MKNOD can be used only by a superuser.

Parameters

pathname
Specifies the name of the character special file to be created. The name can be a relative pathname or an absolute pathname. It must be enclosed in single quotes. A relative pathname is relative to the working directory of the TSO/E session (usually the **HOME** directory). Therefore, you should usually specify an absolute pathname. The name can be up to 1023 characters long. The name is case-sensitive; the system stores each character in the case entered. This operand is required.

All directories in the pathname must exist. If the specified file already exists, no new file is created.

MAJOR(device_major_number)
Specifies the device major number, which can be a decimal number between 0 and 65 535 (64K minus 1). See *OS/390 UNIX System Services Planning* for information on specifying the device major number. This operand is required.

MINOR(device_minor_number)
Specifies the device minor number, which can be a decimal number between 0 and 65 535 (64K minus 1). See *OS/390 UNIX System Services Planning* for information on specifying the device minor number. This operand is required.

MODE(file_permission_bits)
Specifies the file permission bits as three octal numbers, from `0` to 7, separated by commas or blanks. The octal values represent read (r), write (w), and execute (x) access for: user, group, and other.

User permission is the permission given to the file owner. Group permission is the permission given to the group the owner is a member of. Other permission is the permission given to any other user.

The access indicated by each of the numbers `0`–7 is:

0 No access
1 Search (x) access
2 Write-only (w) access
3 Write and execute (wx) access
4 Read-only (r) access
5 Read and execute (rx) access
6 Read and write (rw) access
7 Read, write, and execute (rwx) access

The default permissions set when a character special file is created are 666, representing:

6 User: read and write access
6 Group: read and write access
6 Other: read and write access

Examples

1. You want to create a character special file using an absolute pathname, giving read, write, and execute access to the file owner and no access to others. The filename is **tty1** in the existing directory **/dev**. The device major number is 2; the minor number is 1. You enter:

   ```
   MKNOD '/dev/tty1' MAJOR(2) MINOR(1) MODE(7,0,0)
   ```

2. You want to create a character special file named **ptty2** in the existing directory **/dev**. The device major number is 1; the device minor number is 457. You want the default permissions. You enter:

   ```
   MKNOD '/dev/ptty2' MAJOR(1) MINOR(457)
   ```

3. You want to create a new **tty** pair using an absolute pathname. The filename is **ttyp0042** in the existing directory **/dev**. The device minor number is 42. You want the default permissions. You enter:

   ```
   MKNOD '/dev/ptyp0042' MAJOR(1) MINOR(42)
   MKNOD '/dev/ttyp0042' MAJOR(2) MINOR(42)
   ```

MOUNT — Logically mount a file system

Format

```
MOUNT FILESYSTEM(file_system_name)
                 MOUNTPOINT(pathname)
                 TYPE(file_system_type)
                 MODE(RDWR|READ)
                 PARM(parameter_string)
                 SETUID|NOSETUID
                 WAIT|NOWAIT
                 SECURITY|NOSECURITY
                 SYSNAME (sysname)
                 AUTOMOVE | NOAUTOMOVE
```

Description

For hierarchical file systems, you can use the MOUNT command to logically mount, or add, a mountable file system to the file system hierarchy. You can unmount any mounted file system using the UNMOUNT command.

Note: A mount user must have UID (0) or at least have READ access to the BPX.SUPERUSER FACILITY class.

FILESYSTEM(file_system_name)
Specifies the name of the file system to be added to the file system hierarchy.

file_system_name
For the hierarchical file system (HFS), this is the fully qualified name of the MVS HFS data set that contains the file system. It cannot be a partitioned data set member.

The file system name specified must be unique among previously mounted file systems. The file system name supplied is changed to all uppercase characters. You can enclose it in single quotes, but they are not required.

If FILESYSTEM(''''file_system_name'''') is specified, the file system name will not be translated to uppercase.

MOUNTPOINT(pathname)
Specifies the pathname of the mount point directory, the place within the file hierarchy where the file system is to be mounted. This operand is required.

pathname
Specifies the mount point pathname. The pathname must be enclosed in single quotes. The name can be a relative pathname or an absolute pathname. A relative pathname is relative to the working directory of the TSO/E session (usually the **HOME** directory). Therefore, you should usually specify an absolute pathname. It can be up to 1023 characters long. Pathnames are case-sensitive, so enter the pathname exactly as it is to appear.

Notes:

1. The mount point must be a directory. ***Any files in that directory are inaccessible while the file system is mounted.***
2. Only one file system can be mounted to a mount point at any time.

TYPE(file_system_type)
Specifies the type of file system that will perform the logical mount request. The system converts the TYPE operand value to uppercase letters. This operand is required.

file_system_type
This name must match the TYPE operand of the FILESYSTYPE statement that activates this physical file system in the BPXPRMxx parmlib member. The file_system_type value can be up to 8 characters long.

MODE(RDWR|READ)
Specifies the type of access the file system is to be opened for.

RDWR
Specifies that the file system is to be mounted for read and write access. RDWR is the default if MODE is omitted.

READ
Specifies that the file system is to be mounted for read-only access.

The HFS allows a file system that is mounted using the MODE(READ) option to be shared as read-only with other systems that share the same DASD.

PARM(parameter_string)
Specifies a parameter string to be passed to the file system type. The format and content are specified by the physical file system that is to perform the logical mount. You can specify lowercase or uppercase characters. Enclose the string in single quotes.

parameter_string
Specifies a parameter string value that can be up to 1024 characters long. The parameter string must be enclosed in single quotes; it is case-sensitive.

For an HFS file system, the following can be specified:
`PARM('SYNC(t),NOWRITEPROTECT')`

SYNC(t)
`t` is a numeric value that specifies the number of seconds that should be used to override the sync interval default for this file system during a specific mount. If SYNC is not specified at mount time, then the sync interval default value will be used (a value of 60 seconds). The same rules apply to the argument to the SYNC keyword at mount time as apply to the argument of the SYNCDEFAULT keyword at HFS initialization time. For more information on the SYNCDEFAULT keyword, see *OS/390 UNIX System Services Planning*.

NOWRITEPROTECT
The HFS has a Write Protection mechanism that adds some overhead to HFS processing. This overhead can be avoided by turning off the write protection by specifying NOWRITEPROTECT in the PARM field of the MOUNT command.

SETUID|NOSETUID
Specifies whether the SETUID and SETGID mode bits on executables in this file system are respected. Also determines whether the APF extended attribute or the Program Control extended attribute is honored.

SETUID
Specifies that the SETUID and SETGID mode bits be respected when a program in this file system is run. SETUID is the default.

NOSETUID
Specifies that the SETUID and SETGID mode bits not be respected when a program in this file system is run. The program runs as though the SETUID and SETGID mode bits were not set. Also, if you specify the NOSETUID option on MOUNT, the APF extended attribute and the Program Control extended attribute are not honored.

WAIT|NOWAIT
Specifies whether to wait for an asynchronous mount to complete before returning.

WAIT
Specifies that MOUNT is to wait for the mount to complete before returning. WAIT is the default.

NOWAIT
Specifies that if the file system cannot be mounted immediately (for example, a network mount must be done), then the command will return with a return code indicating that an asynchronous mount is in progress.

SECURITY|NOSECURITY
Specifies whether security checks are to be enforced for files in this file system.

SECURITY
Specifies that normal security checking will be done. SECURITY is the default.

NOSECURITY
Specifies that security checking will not be enforced for files in this file system. A user may access or change any file or directory in any way.

Security auditing will still be performed if the installation is auditing successes.

The SETUID, SETGID, APF, and Program Control attributes may be turned on in files in this file system, but they will not be honored while it is mounted with NOSECURITY.

SYSNAME (sysname)
For systems participating in shared HFS, SYSNAME specifies the particular system on which a mount should be performed. This system will then become the owner of the file system mounted. This system must be IPLed with SYSPLEX(YES). The default for SYSNAME is the name of the system the mount is processed on.

sysname
sysname is a 1–8 alphanumeric name of a system participating in shared HFS.

AUTOMOVE | NOAUTOMOVE
The AUTOMOVE|NOAUTOMOVE parameters apply only in a sysplex where systems are participating in shared HFS. They indicate what happens if the system that owns a file system goes down. The default setting is AUTOMOVE.

AUTOMOVE
When AUTOMOVE is specified for a file system and the file system's owner goes down, AUTOMOVE indicates that ownership of the file system can be automatically moved to another system participating in shared HFS. AUTOMOVE is the default.

NOTAUTOMOVE
When NOAUTOMOVE is specified for a file system, this indicates that ownership should not be moved to another system participating in shared HFS if the file system's owner should crash.

For more information on shared HFS, see *OS/390 UNIX System Services Planning*.

Note: In OS/390 R9 and later, to ensure that the root is always available, use the defaults for SYSNAME and AUTOMOVE.

Usage Notes

1. The directory **/samples** contain sample MOUNT commands (called **mountx**).
2. When the mount is done asynchronously (NOWAIT was specified and return code 4 was returned), you can determine if the mount has completed with one of the following:
   - The **df** shell command
   - The **DISPLAY OMVS,F** operator command (see *OS/390 MVS System Commands*)
   - The MOUNT table option on the File Systems pulldown in the ISPF Shell (accessed by the ISHELL command)
   - In order to mount a file system as the system root file system, the caller must be a superuser. Also, a file system can only be mounted as the system root file system if the root file system was previously unmounted.

- If you have previously unmounted the root file system, a 'dummy file system' or SYSROOT will be displayed as the current root file system. During the time when SYSROOT is displayed as the root, any operation that requires a valid file system will fail. When you subsequently mount a new root file system on mountpoint **/**, that new file system will replace SYSROOT. When a new root file system has been mounted, you should terminate any current dubbed users or issue a **chdir** using a full pathname to the appropriate directory. This way, the users can access the new root file system. Otherwise, an error will occur when a request is made requiring a valid file system.
- Systems exploiting shared HFS will have I/O to an OMVS couple data set. Because of these I/O operations to the CDS, each mount request requires additional system overhead. You will need to consider the affect that this will have on your recovery time if a large number of mounts are required on any system participating in shared HFS.

Return Codes

0 Processing successful.
4 Processing incomplete. An asynchronous mount is in progress.
12 Processing unsuccessful. An error message has been issued.

Examples

1. To mount the HFS data set HFS.WORKDS on the directory **/u/openuser**, enter:

   ```
   MOUNT FILESYSTEM('HFS.WORKDS') MOUNTPOINT('/u/openuser') TYPE(HFS)
   ```

2. The following example mounts the HFS directory **/u/shared_data**, which resides on the remote host named **mvshost1**, onto the local directory **/u/jones/mnt**. The command may return before the mount is complete, allowing the mount to be processed in parallel with other work. The SETUID and SETGID bits are honored on any executable programs:

   ```
   MOUNT FILESYSTEM('MVSHOST1.SHARE.DATA') MOUNTPOINT('/u/jones/mnt')
         TYPE(NFSC) PARM('mvshost1:/hfs/u/shared_data') NOWAIT SETUID
   ```

OBROWSE — Browse an HFS file

Format

```
OBROWSE pathname
```

or

```
OBROWSE 'pathname'
```

or

```
OBROWSE
```

(The pathname is optional in the last example.)

Description

The OBROWSE command enables you to browse a file in the hierarchical file system (HFS). This command uses the ISPF/PDF Browse facility.

If you enter the OBROWSE command without specifying a pathname, the Browse Entry panel is displayed. From that panel, you can enter the directory name and filename of an existing file you want to browse. If you are browsing fixed-length records, you must also indicate the record length.

Parameters

pathname
Specifies the pathname of the file to be browsed. The pathname can be absolute or relative. It can be enclosed in single quotes. A relative pathname is relative to the working directory of the TSO/E session (usually the **HOME** directory). Therefore, you should usually specify an absolute pathname. If you enter the OBROWSE command from the shell, use the absolute pathname. Avoid using spaces or single quotes within the pathname.

OCOPY — Copy an MVS data set member or HFS file to another member or file

Format

```
OCOPY INDD(ddname1) OUTDD(ddname2)
BINARY | TEXT
CONVERT(character_conversion_table | YES | NO)
PATHOPTS (USE|OVERRIDE)
TO1047 | FROM1047
```

Description

You can use the OCOPY command to copy data between an MVS data set and the hierarchical file system (HFS), between two data sets, or between two files. For OCOPY, you would want to use CONVERT for these two situations:

- Conversion between code pages IBM-037 and IBM-1047
- Conversion between ASCII and code page IBM-1047

The OS/390 shell uses code page 1047, and MVS uses a Country Extended Code Page. You can convert data to or from code page 1047 while it is being copied.

If you are copying a file with doublebyte data, do not use the CONVERT option.

Before using the OCOPY command, you must allocate the data set or file you are working with. When using the TSO/E ALLOCATE command or a JCL DD statement to allocate a file or data set, you can specify PATHMODE and PATHOPTS parameters along with the PATH parameter. For information on the use of these parameters with the JCL statement, see *OS/390 MVS JCL Reference*. For information on the TSO/E ALLOCATE command, see *OS/390 TSO/E Command Reference*.

You can use OCOPY to copy:

- A member of an MVS partitioned data set (PDS or PDSE) to a file
- An MVS sequential data set to a file

- A file to a member of an MVS PDS or PDSE
- A file to an MVS sequential data set
- A file to a file
- A member of an MVS PDS or PDSE to another member of an MVS PDS or PDSE
- A member of an MVS PDS or PDSE to an MVS sequential data set
- An MVS sequential data set to another MVS sequential data set
- An MVS sequential data set to a member of an MVS PDS or PDSE

Both INDD and OUTDD can represent an MVS data set or a file. If the source (INDD) is an MVS data set and the target (OUTDD) is an HFS file, then OCOPY copies an MVS data set to a file; the operation is the same as the OPUT command. If the source (INDD) is an HFS file and the target (OUTDD) is an MVS data set, then OCOPY copies a file to an MVS data set; the operation is the same as the OGET command.

Both the target and source can be an MVS data set or member of a partitioned data set, or both can be a file. This function is typically used for code page conversion.

If PATHMODE, which sets the permission bits for a new file, is specified during allocation, it is used when creating a new file. If PATHMODE is not specified during the allocation of a new file, the allocation creates a file with the default permission of `000`, which means the user has no access to it.

Parameters

INDD(ddname1)
Specifies the ddname of the source. The ddname is up to 8 characters long.

OUTDD(ddname2)
Specifies the ddname of the target. The ddname is up to 8 characters long.

BINARY | TEXT
Specifies that the data to be copied is a binary file or text file.

BINARY
Specifies that the data to be copied is a binary file. The default is binary when copying a data set of undefined record format to a file.

When you specify BINARY, OCOPY operates without any consideration for <newline> characters or the special characteristics of DBCS data. For example, doublebyte characters might be split between MVS data set records, or a “shift-out” state might span records.

TEXT
Specifies that the data to be copied is a text file. The default is text except when copying a data set of undefined record format to a file.

If you are using a DBCS-supported terminal, you should use TEXT. It is assumed that doublebyte data in the file system includes the <newline> character in order to delineate line boundaries. Data within these lines that are delineated by <newline> characters must begin and end in the “shift-in” state.

CONVERT(character_conversion_table | YES | NO)
Specifies the character conversion table used to convert between the following:

- Code pages IBM-037 and IBM-1047
- The ASCII code page and IBM-1047

If this optional operand is omitted, the system copies the data without conversion.

Use this option for singlebyte data only.

Specify the CONVERT value as one of the following:

character_conversion_table
Specify one of the following:

- **data_set_name(member_name)**. Specifies the name of the partitioned data set (library) and the name of the member that contains the character conversion table.
- **data_set_name**. Specifies the name of the partitioned data set (library) that contains the character conversion table as the default member. The default member name is BPXFX000. (BPXFX000 is an alias; when shipped by IBM, it points to BPXFX111.) A fully qualified data set name must be enclosed in single quotes.
- **(member_name)**. Specifies the name of the conversion table to be used. It is a member of a PDS. Since the data_set_name is omitted, the standard library concatenation is searched for the table. (The default library is SYS1.LINKLIB.)

  The following list summarizes what you can specify when you want to convert data to a different code page when copying singlebyte data:

  – BPXFX100. Null character conversion table. Use this table if the square brackets at your workstation are at the same code points as the square brackets on code page 1047 (it is the default). Also use this table if you are using a DBCS terminal.

  – BPXFX111. Specifies a non-APL conversion table to convert between code pages IBM-037 and IBM-1047.

  – BPXFX211. Specifies an APL conversion table to convert between code pages IBM-037 and IBM-1047.

  – BPXFX311. Specifies an ASCII-EBCDIC conversion table to convert between code pages ISO8859-1 and IBM-1047.

YES
Specifies that the system is to perform conversion and use the default conversion table (BPXFX000) in the system library concatenation. (BPXFX000 is an alias; when shipped by IBM, it points to BPXFX111.)

NO
Specifies that conversion not be done. NO is the same as omitting the CONVERT operand.

PATHOPTS(USE | OVERRIDE)
Specifies whether the OCOPY should use or override the PATHOPTS value specified during allocation. If the PATHOPTS is not specified in the allocation, OCOPY will open the file with the appropriate PATHOPTS.

USE
Specifies that the PATHOPTS value is to be enforced. If a file that was identified as read-only when it was allocated is identified as the output file for OCOPY, OCOPY fails. Similarly, if a write-only file is specified as the input file, OCOPY fails. USE is the default.

OVERRIDE
Specifies that the PATHOPTS value specified during allocation is to be ignored.

TO1047 | FROM1047

TO1047
Specifies that the TO section of the character conversion table is to be used. This is usually used to convert from code page IBM-037 or ASCII to code page IBM-1047.

FROM1047
Specifies that the FROM section of the conversion table is to be used. This is usually used to convert from code page IBM-1047 to code page IBM-037 or ASCII.

If the CONVERT operand is specified and this operand is omitted,

- Data copied from an MVS data set to a file uses the TO1047 section of the table.
- Data copied from a file to an MVS data set uses the FROM1047 section of the table.

If the CONVERT operand is specified for a copy from a file to a file or an MVS data set to an MVS data set, you must specify either TO1047 or FROM1047.

Usage Notes

1. You can use OCOPY to copy a program object from a PDSE to the file system, and it will be executable there. If you have a load module in a partitioned data set, however, you must first use the IEBCOPY program to copy the load module from a partitioned data set to a PDSE and then subsequently use OCOPY to copy the module into the file system. The IEBCOPY converts the load module to a program object.

   Note: You can use the linkage editor to put the load module directly into the file system.

2. An executable file copied from the file system into an MVS data set is not executable under MVS. Some required directory information is lost during the copy. See *OS/390 UNIX System Services User's Guide* for a discussion of copying executable files.
3. Data sets with spanned records are not allowed.
4. When you are copying into an existing file, data is appended to the end of the file if OAPPEND is specified in PATHOPTS. Otherwise, the existing file is overwritten.
5. Copying from HFS text files to MVS data sets:

For text files, all <newline> characters are stripped during the copy. Each line in the file ending with a <newline> character is copied into a record of the MVS data set. You cannot copy a text file to an MVS data set in an undefined record format.

- **For an MVS data set in fixed record format:** Any line longer than the record size is truncated. If the line is shorter than the record size, the record is padded with blanks.
- **For an MVS data set in variable record format:** Any line longer than the largest record size is truncated and the record length is set accordingly. A change in the record length also occurs if the line is short.

6. Copying from HFS binary files to MVS data sets:

   For binary files, all data is preserved.

   - **For an MVS data set in fixed record format:** Data is cut into chunks of size equal to the record length. Each chunk is put into one record. The last record is padded with spaces or blanks.
   - **For an MVS data set in variable record format:** Data is cut into chunks of size equal to the largest record length. Each chunk is put into one record. The length of the last record is equal to the length of the data left.
   - **For an MVS data set in undefined record format:** Data is cut into chunks of size equal to the block size. Each chunk is put into one record. The length of the last record is equal to the length of the data left.

7. When you copy MVS data sets to HFS text files, a <newline> character is appended to the end of each record. If trailing blanks exist in the record, the <newline> character is appended after the trailing blanks.
8. When you copy MVS data sets to HFS binary files, the <newline> character is not appended to the record.

Return Codes

0 Processing successful.
12 Processing unsuccessful. An error message has been issued.

Examples

1. The following commands copy an MVS sequential data set to an HFS file. This is text data, and there is no code page conversion.

   - SYSUT1 is the ddname of the source data set, EMPLOYEE.DATA.
   - PATHNAME is the ddname of the target, which is the existing file **/u/admin/employee/data**.

   ```
   ALLOCATE FILE(sysut1) DATASET('employee.data')
   ALLOCATE FILE(pathname) PATH('/u/admin/employee/data')
   OCOPY INDD(sysut1) OUTDD(pathname) TEXT
   ```

2. The following commands copy a binary file into a member of a partitioned data set:

   - BINARY is the ddname of the source file, **bin/payroll**. This file is in the working directory.

- MVSPDS is the ddname of the target data set member, APPL.CODES(PAYROLL)

```
ALLOCATE FILE(binary) PATH('/bin/payroll')
ALLOCATE FILE(mvspds) DATASET('appl.codes(payroll)')
OCOPY INDD(binary) OUTDD(mvspds) BINARY
```

3. The following commands copy system input from the MVS SYSIN data set to the file system and perform code page conversion:
   - SYSIN is the ddname of the source, IBMUSR.EMPLOYEE.DATA.
   - PATHNAME is the ddname of the target, **/u/admin/employee/data**. This file does not currently exist and is created by ALLOCATE.
   - This is text data.
   - The character conversion table is the default table, member BPXFX000 of the SYS1.BPXLATE data set. (BPXFX000 is an alias; when shipped by IBM, it points to BPXFX111.)
   - Because this is a copy from an MVS data set to a file, the section TO1047 of the conversion table is used by default.

```
ALLOCATE FILE(sysin) DATASET('IBMUSR.EMPLOYEE.DATA')

ALLOCATE FILE(pathname) PATH('/u/admin/employee/data')
PATHMODE (sirwxu) PATHOPTS (ocreat, owronly)

OCOPY INDD(sysin) OUTDD(pathname) TEXT CONVERT((BPXFX000))
```

   (BPXFX000 is an alias; when shipped by IBM, it points to BPXFX111.)

4. The following OCOPY command copies data from one MVS sequential data set to another MVS sequential data set and performs code page conversion. This example shows just the OCOPY command; the necessary ALLOCATE commands are not included.
   - SYSUT1 is the ddname of the source data set.
   - TRANSDD is the ddname of the target data set.
   - This is text data.
   - The data is converted using the user-specified character conversion table and the TO1047 section of the table.

```
OCOPY INDD(sysut1) OUTDD(transdd) TEXT CONVERT('sys1.mylib(mytab)') TO1047
```

OEDIT — Edit an HFS file

Format

```
OEDIT pathname
```

or

```
OEDIT 'pathname'
```

Description

OEDIT enables you to edit a file in the hierarchical file system (HFS). This command uses the ISPF/PDF Edit facility.

If you enter OEDIT without specifying a pathname, the Edit Entry panel is displayed. From that panel, you can enter the directory name and filename of an existing file, or you can specify a directory name and filename for a new file. The Edit Entry panel also lets you specify an edit profile and an initial edit macro.

For an introduction to using ISPF File Edit, see *OS/390 UNIX System Services User's Guide*.

Parameters

pathname
Specifies the pathname of the file to be edited. The pathname can be absolute or relative. It can be enclosed in single quotes. A relative pathname is relative to the working directory of the TSO/E session (usually the **HOME** directory). Therefore, you should usually specify an absolute pathname. If you enter OEDIT from the shell, use the absolute pathname. Avoid using spaces or single quotes within pathnames.

Usage Notes

1. OEDIT attempts to load the file into a VB255 session. If this is an ISPF that supports wide edit (such as ISPF 4.1) and any line exceeds 235 characters, the width for the new session is the length of the longest line plus 25% to allow for some expansion.
2. The COPY command cannot copy in files that have records wider than the edit session.

OGET — Copy an HFS file into an MVS data set

Format

```
OGET  'pathname'
      mvs_data_set_name | mvs_data_set_name(member_name)
      BINARY | TEXT
      CONVERT(character_conversion_table | YES | NO)
```

Description

You can use the OGET command to copy an HFS file:

- To a member of an MVS partitioned data set (PDS or PDSE)
- To an MVS sequential data set

and convert the data from code page 1047 to code page IBM-037 or ASCII while it is being copied. Do not use the CONVERT option when copying files that contain doublebyte data. This option is used for singlebyte data only, not for doublebyte data.

Parameters

pathname
Specifies the pathname of the file that is being copied to a data set. This operand is required. The pathname is:

- A relative or absolute pathname. A relative pathname is relative to the working directory of the TSO/E session (usually the **HOME** directory). Therefore, you should usually specify an absolute pathname.
- Up to 1023 characters long.
- Enclosed in single quotes.
- In uppercase or lowercase characters, which are not changed by the system.

mvs_data_set_name | mvs_data_set_name(member_name)
Specifies the name of an MVS sequential data set or an MVS partitioned data set member to receive the file that is being copied. One of these two operands is required. The data set name is:

- A fully qualified name that is enclosed in single quotes, or an unqualified name
- Up to 44 characters long
- Converted to uppercase letters by the system

BINARY | TEXT
Specifies whether the file being copied contains binary data or text.

BINARY
Specifies that the file being copied contains binary data.

When you specify BINARY, OGET operates without any consideration for <newline> characters or the special characteristics of DBCS data. For example, doublebyte characters might be split between MVS data set records, or a “shift-out” state might span records.

TEXT
Specifies that the file being copied contains text. This is the default.

If you are using a DBCS-supported terminal, you should use TEXT. It is assumed that doublebyte data in the file system includes the <newline> character in order to delineate line boundaries. Data within these lines that are delineated by <newline> characters must begin and end in the “shift-in” state.

CONVERT(character_conversion_table | YES | NO)
Specifies that the data being copied is to be converted from IBM-1047 to IBM-037 or ASCII. This operand is optional. If is omitted, the system copies the data without conversion.

Use this option for singlebyte data only.

Specify the CONVERT value as one of the following:

character_conversion_table
Specify one of the following:

- **data_set_name(member_name)**. Specifies the name of the partitioned

data set (PDS) and the name of the member that contains the character conversion table.

- **data_set_name**. Specifies the name of the partitioned data set (PDS) that contains the character conversion table. The table is the FROM1047 part in member BPXFX000. (This is an alias; when shipped by IBM, it points to BPXFX111.)
- **(member_name)**. Specifies the name of the conversion table to be used. It is a member of a PDS. Since the data_set_name is omitted, the standard library concatenation is searched for the table. (The default library is SYS1.LINKLIB.)

  The following list summarizes what you can specify when you want to convert data to a different code page when copying singlebyte data:

  – BPXFX100. Null character conversion table. Use this table if the square brackets at your workstation are at the same code points as the square brackets on code page 1047 (it is the default). Also use this table if you are using a DBCS terminal.

  – BPXFX111. Specifies a non-APL conversion table to convert between code pages IBM-037 and IBM-1047.

  – BPXFX211. Specifies an APL conversion table to convert between code pages IBM-037 and IBM-1047.

  – BPXFX311. Specifies an ASCII-EBCDIC conversion table to convert between code pages ISO8859-1 and IBM-1047.

YES
Specifies that the system is to perform conversion and use the default conversion table (BPXFX000) in the standard library concatenation. (BPXFX000 is an alias; when shipped by IBM, it points to BPXFX111.)

NO
Specifies that conversion not be done. NO is the same as omitting the CONVERT operand.

Do not use the CONVERT parameter on files containing doublebyte data. Doublebyte data in the file system is in code page 939. If you need to convert to a code page other than 939, you use the **iconv** command.

Usage Notes

1. For text files, all <newline> characters are stripped during the copy. Each line in the file ending with a <newline> character is copied into a record of the MVS data set. You cannot copy a text file to an MVS data set in an undefined record format.

   - **For an MVS data set in fixed record format:** Any line longer than the record size is truncated. If the line is shorter than the record size, the record is padded with blanks.
   - **For an MVS data set in variable record format:** Any line longer than the largest record size is truncated; the record length is set to the length of the line. A change in the record length also occurs if the line is short.

   For text mode transfer, if the line is longer than the record size, the line is truncated (for DBCS, perhaps in the middle of a doublebyte character or in

"shift-in" state). If the line is shorter than the record size, the record is padded with blanks.

2. For binary files, all data is preserved.
   - **For an MVS data set in fixed record format:** Data is cut into chunks of size equal to the record length. Each chunk is put into one record. The last record is padded with spaces or blanks.
   - **For an MVS data set in variable record format:** Data is cut into chunks of size equal to the largest record length. Each chunk is put into one record. The length of the last record is equal to the length of the data left.
   - **For an MVS data set in undefined record format:** Data is cut into chunks of size equal to the block size. Each chunk is put into one record. The length of the last record is equal to the length of the data left.

   For binary mode transfers, doublebyte characters might be split between MVS data set records, or a "shift-out" state might span records.
3. If the MVS data set does not exist, OGET allocates a new data set, a sequential data set of variable record format. However, OGET does not allocate a new partitioned data set. The record length of the new data set is either 255 or the size of the longest line in the HFS file, whichever is larger. Dynamic allocation services determine the block size and space, based on installation-defined defaults. If the defaults are not sufficient, you should allocate a new MVS data set and then specify it on OGET.

   A simple method of allocating a sufficient size is to specify a primary extent size and a secondary extent size equal to the number of bytes in the file being copied.
4. An executable file copied into an MVS data set is not executable under MVS, because some required directory information is lost during the copy to the partitioned data set.
5. Data sets with spanned records are not allowed.
6. If you are using a DBCS-supported terminal, the target MVS data set should be defined or defaulted to variable record format. The record length of the data set must be greater than or equal to the longest line in the HFS file. (OGET can determine how long the longest line is if you ask it to allocate the target data set.)

Return Codes

0 Processing successful.
12 Processing unsuccessful. An error message has been issued.

Examples

1. The following command copies a text file to an MVS sequential data set, converting from code page 1047 to code page IBM-037 using the default table BPXFX000. (BPXFX000 is an alias; when shipped by IBM, it points to BPXFX111.)
   - The pathname of the file is **/u/admin/employee/data**.
   - The unqualified name of the sequential data set is EMPLOYEE.DATA.

   ```
   OGET '/u/admin/employee/data' EMPLOYEE.DATA TEXT CONVERT(YES)
   ```

2. The following command copies a text file to an MVS sequential data set, converting from code page 1047 to code page IBM-037 using conversion table BPXFX111 in the user's library data set.
   - The pathname of the file is **/u/admin/employee/data**.
   - The fully qualified name of the sequential data set is IBMUSR.EMPLOYEE.DATA.

```
OGET '/u/admin/employee/data' 'IBMUSR.EMPLOYEE.DATA'
         CONVERT('MY.LOADLIB(BPXFX111)')
```

OGETX — Copy HFS files from a directory to an MVS PDS or PDSE

Format

```
OGETX  hfs_directory | hfs_file_name
       mvs_PDS_name | mvs_data_set_name(member_name)
       ASIS
       BINARY | TEXT
       CONVERT(character_conversion_table | YES | NO)
       LC
       QUIET
       SUFFIX(suffix)
```

Description

You can use the OGETX command to:

- Copy files from an HFS (hierarchical file system) directory to a member of an partitioned data set (PDS) or PDSE
- Copy an individual file to a sequential data set or member of a partitioned data set

and convert the data from code page 1047 to code page IBM-037 or ASCII while it is being copied.

Do not use the CONVERT option when copying files that contain doublebyte data. This option is used for singlebyte data only, not for doublebyte data.

Parameters

hfs_directory | hfs_file_name
Specifies the pathname of the HFS directory or HFS filename that is being copied to an MVS PDS or PDSE. The HFS files are copied into members of the PDS or PDSE.

Use `hfs_directory` when a PDS is specified. When a sequential data set or PDS member is specified, then the filename must be used.

These limitations apply to an MVS data set name:

- It can use uppercase alphabetic characters *A* through *Z*, but not lowercase letters.
- It can use numeric characters `0` through `9`, and the special characters @, #, and $.
- It cannot begin with a numeric character.

- The member name cannot be more than 8 characters. If a filename is longer than 8 characters or uses characters that are not allowed in an MVS data set name, the file is not copied.

The LC operand lets you copy HFS filenames that are lowercase, mixed case, or uppercase.

Single quotes around the directory name or filename are optional.

mvs_PDS_name | mvs_data_set_name(member_name)
mvs_PDS_name specifies the name of an MVS PDS or PDSE to receive the HFS files that are being copied. mvs_data_set_name(member_name) specifies the name of an MVS partitioned data set member to receive the file that is being copied. The name is:

- A fully qualified name that is enclosed in single quotes, or an unqualified name
- Up to 44 characters long, with an additional 8 characters for the member name
- Converted to uppercase letters

ASIS
Specifies that the _ character in pathnames *not* be translated to the @ character in member names. (It is a common convention to use @ symbols in PDS member names to correspond with the _ symbol in pathnames.)

BINARY | TEXT
Specifies whether the files in the directory being copied contains binary data or text. For more information, see Note 7 on page 761.

BINARY
Specifies that the files in the directory being copied contains binary data.

When you specify BINARY, OGET operates without any consideration for <newline> characters or the special characteristics of DBCS data. For example, doublebyte characters might be split between MVS data set records, or a “shift-out” state might span records.

TEXT
Specifies that the files in the directory being copied contains text. This is the default.

If you are using a DBCS-supported terminal, you should use TEXT. It is assumed that doublebyte data in the file system includes the <newline> character in order to delineate line boundaries. Data within these lines that are delineated by <newline> characters must begin and end in the “shift-in” state.

CONVERT(character_conversion_table | YES | NO)
Specifies that the data being copied is to be converted from code page 1047 to code page IBM-237 or ASCII; that is,the FROM1047 part of the specified character conversion table is used. This operand is optional. If it is omitted, the system copies the data without conversion.

Use this option for singlebyte data only.

Specify the CONVERT value as one of the following:

character_conversion_table
Specify one of the following:

- **data_set_name(member_name)**. Specifies the name of the partitioned data set (PDS) and the name of the member that contains the character conversion table.
- **data_set_name**. Specifies the name of the partitioned data set (PDS) that contains the character conversion table. The table is the FROM1047 part in member BPXFX000. (This is an alias; when shipped by IBM, it points to BPXFX111.)
- **(member_name)**. Specifies the name of the conversion table to be used. It is a member of a PDS. Since the data_set_name is omitted, the standard library concatenation is searched for the table. (The default library is SYS1.LINKLIB.)

  The following list summarizes what you can specify when you want to convert data to a different code page when copying singlebyte data:

  – BPXFX100. Null character conversion table. Use this table if the square brackets at your workstation are at the same code points as the square brackets on code page 1047 (it is the default). Also use it if you are using a DBCS terminal.

  – BPXFX111. Specifies a non-APL conversion table to convert between code pages IBM-037 and IBM-1047.

  – BPXFX211. Specifies an APL conversion table to convert between code pages IBM-037 and IBM-1047.

  – BPXFX311. Specifies an ASCII-EBCDIC conversion table to convert between code pages ISO8859-1 and IBM-1047.

YES
Specifies that the system is to perform conversion and use the default conversion table (BPXFX000) in the standard library concatenation. (BPXFX000 is an alias; when shipped by IBM, it points to BPXFX111.)

NO
Specifies that conversion not be done. NO is the same as omitting the CONVERT operand.

Do not use the CONVERT parameter on files containing doublebyte data. Doublebyte data in the file system is in code page 939. If conversion to a code page other than 939 is required, you should use the **iconv** command.

LC
Specifies that the HFS filenames can be lowercase, uppercase, or mixed. If LC is not specified, the HFS filenames must be uppercase. Filenames are converted to uppercase member names.

QUIET
Turns off the echoing of the OGET command before a file is copied.

SUFFIX(suffix)
Specifies that files with the files created by (suffix) be copied and the suffix be dropped from the HFS filename when the PDS members are created.

A *suffix* is an optional additional file identifier that is appended to the filename following the first period (**.**). It is usually used to identify the type of file.

Usage Notes

1. Avoid using OGETX with pathnames containing single quotes or spaces.
2. For text files, all <newline> characters are stripped during the copy. Each line in the file ending with a <newline> character is copied into a record of the MVS data set. You cannot copy a text file to an MVS data set in an undefined record format.
   - **For an MVS data set in fixed record format:** Any line longer than the record size is truncated. If the line is shorter than the record size, the record is padded with blanks.
   - **For an MVS data set in variable record format:** Any line is longer than the largest record size is truncated; the record length is set accordingly. A change in the record length also occurs if the line is short.
3. For binary files, all data is preserved.
   - **For an MVS data set in fixed record format:** Data is cut into chunks of size equal to the record length. Each chunk is put into one record. The last record is padded with spaces or blanks.
   - **For an MVS data set in variable record format:** Data is cut into chunks of size equal to the largest record length. Each chunk is put into one record. The length of the last record is equal to the length of the data left.
   - **For an MVS data set in undefined record format:** Data is cut into chunks of size equal to the block size. Each chunk is put into one record. The length of the last record is equal to the length of the data left.
4. Data sets with spanned records are not allowed.
5. Before the copy, the OGET command for a file is echoed, unless you specify the QUIET option. If you did not specify QUIET and if the command is not echoed for a file, it has not met the copy criteria and is not copied.
6. If more than one filename is the same, the file is overwritten on each subsequent copy. For example, if you specify a copy of **Pgma** and **pgma** and use LC, the first file copied is overwritten. Or if you copy **pgma.h** and **pgma.c** and specify SUFFIX, the first file copied is overwritten.
7. If the target data set is a PDS with an undefined record format, the files may be treated as load modules. A load module is copied by link-editing it into the target library. For the program to be able to execute, the entry point must be at the beginning of the load module.

   For OGETX to treat the file as a load module, do not specify either TEXT nor BINARY.
8. If the source for the copy is a file, the target can be specified as a PDS. The member name used is the filename, which is in uppercase and has had any suffixes removed. Any remaining characters in the member name that are not valid in member names cause the copy to fail. You do not have to specify a file as a target with a sequential data set, or a directory as a target with a PDS. The ASIS option is not affected.

Examples

1. The following command copies the files in the HFS directory **/usr/sbllib** to the MVS PDS named DATAFILE, removing any suffixes appended to the HFS files and accepting lowercase filenames.

   ```
   OGETX /usr/sbllib/ DATAFILE LC SUFFIX
   ```

   The members **/usr/sbllib/program1.c**, **/usr/sbllib/list.prg**, and **usr/sbllib/program2.c** become DATAFILE(PROGRAM1), DATAFILE(LIST), and DATAFILE(PROGRAM2).

2. The following command copies the files with the suffix of **c** in the HFS directory **/usr/sbllib** to the MVS PDS named DATAFILE, removing the **.c** suffix appended to the HFS files and accepting lowercase filenames.

   ```
   OGETX /usr/sbllib/ DATAFILE LC SUFFIX(c)
   ```

   The members **/usr/sbllib/program1.c**, **/usr/sbllib/list.prg**, and **usr/sbllib/program2.c** become DATAFILE(PROGRAM1) and DATAFILE(PROGRAM2).

OHELP — Display online OS/390 UNIX System Services publications

Format

```
OHELP ref_id name
```

Description

You can use the OHELP command to display online publications to show information on commands, C functions, callable services, and messages issued by the shell and **dbx**.

To use OHELP, the OS/390 UNIX System Services must be installed and you must be defined as an OS/390 UNIX System Services user. The list of books that OHELP searches is defined by the installation. To obtain a complete list of books and their numbers, type OHELP without any operands. Your installation may have added more *ref_id*s and books to the list. *OS/390 UNIX System Services Planning* contains information on installing books for the OHELP command.

Parameters

ref_id
Specifies the number that identifies the book you want to search. The possible values are:

```
1 'IBMBOOKS.EZ239116.BPXA5080.BOOK'     OS/390 UNIX Command Reference
2 'IBMBOOKS.EZ239116.BPXB1080.BOOK'     OS/390 UNIX Callable Services
3 'IBMBOOKS.EZ239116.EDCLB020.BOOK'     OS/390 C/C++ Library Reference
4 'IBMBOOKS.EZ239116.BPXA8080.BOOK'     OS/390 UNIX Messages and Codes
5 'IBMBOOKS.EZ239116.AOPM0M10.BOOK'     OS/390 Infoprint Server Messages
6 'IBMBOOKS.EZ239116.AOPV0M10.BOOK'     OS/390 Infoprint Server User's Guide
* 'IBMBOOKS.EZ239116.BPXSH100.BKSHELF'  OS/390 UNIX OHELP Bookshelf
```

Note: The default is *ref_id=1*.

name
Specifies the name of the function you want information about. If you omit this operand, OHELP displays the table of contents of the book that contains the type of function specified by *ref_id*.

You can also specify a text string enclosed in double quotes. See Example 2.

Examples

1. To look at the description of the **kill** shell command, enter:

   ```
   OHELP 1 kill
   ```

2. To search for all occurrences of the phrase *environment variable* in the OHELP bookshelf, enter:

   ```
   OHELP * "environment variable"
   ```

Usage Notes

Before you can use OHELP, your system must have the BookManager READ product installed. The help information is displayed in a BookManager session, and you cannot work in the shell while viewing the information.

OMVS — Invoke the OS/390 shell

Format

```
OMVS ALARM | NOALARM
     AUTOSCROLL | NOAUTOSCROLL
     CONVERT(character_conversion_table)
     DBCS | NODBCS
     DEBUG(NO | YES | EVENT | DATA)
     ECHO | NOECHO
     ENDPASSTHROUGH(ATTN | CLEAR | CLEARPARTITION |
         ENTER | NO | PA1 | PA3 | PF1 | PF2 | PF3 ... PF24 | SEL)
     ESCAPE('escape-characters')
     LINES(n)
     PFn
         (ALARM | NOALARM |
          AUTOSCROLL | NOAUTOSCROLL |
          BACKSCR |
          BOTTOM |
          CLOSE |
          CONTROL |
          ECHO | NOECHO |
          FWDRETR |
          HALFSCR
          HELP |
          HIDE | NOHIDE |
          NEXTSESS |
          NO |
          OPEN |
          PFSHOW | NOPFSHOW |
          PREVSESS |
          QUIT |
          QUITALL |
          REFRESH |
          RETRIEVE
          RETURN |
          SCROLL |
          SUBCOMMAND |
          TOP |
```

```
     TSO )
PFSHOW | NOPFSHOW
RUNOPTS('LE/370-runtime-options')
SESSIONS(n)
SHAREAS | NOSHAREAS
WRAPDEBUG(n)
```

Description

Use the OMVS command to invoke the OS/390 shell. You can select options on the OMVS command to customize aspects of the shell interface, such as the function keys.

After you are working in a shell session, you can switch to subcommand mode, return temporarily to TSO/E command mode, or end the session by exiting the shell.

Parameters

ALARM | NOALARM
Controls the sounding of the 3270 alarm to alert you to particular events.

The default is ALARM.

ALARM
Causes the 3270 alarm to sound when the <alert> character is encountered in data being sent to the workstation.

NOALARM
Prevents the 3270 alarm from sounding when the <alert> character is encountered in data being sent to the workstation.

AUTOSCROLL | NOAUTOSCROLL
Controls the setting of the autoscroll function. The default is AUTOSCROLL.

AUTOSCROLL
Specifies automatic scrolling of input and output written to the screen.

NOAUTOSCROLL
Specifies that there not be automatic scrolling.

CONVERT(character_conversion_table)
Specifies the OS/390 UNIX System Services character conversion table used to convert between the OS/390 code page and the code page used in the shell.

data_set_name(member_name)
Specifies the name of the partitioned data set (PDS) and the name of the member that contains the character conversion table.

data_set_name
Specifies the name of the partitioned data set containing the character conversion table to be used.

(member_name)
Specifies the name of the character conversion table to be used. It is the name of a member in a partitioned data set.

If both the member_name and data_set_name are omitted, member FSUMQ000 in the default module search order is used as the character

conversion table. Figure 17 on page 765 lists the various formats of the OMVS CONVERT command:

Figure 17. Various Formats of the OMVS CONVERT Command (OMVS command)

Command Format	What It Does
OMVS CONV((BPXFX111))	See Note 1.
OMVS CONV('SYS1.XXXX')	Looks for SYS1.XXXX(FSUMQ000). See Note 2.
OMVS CONV('SYS1.XXXX(BPXFX111)')	Looks for SYS1.XXXX(BPXFX111)
OMVS CONV(XXXX)	Looks for *prefix*.XXXX(FSUMQ000)
OMVS CONV(XXXX(BPXFX111))	Looks for *prefix*.XXXX(BPXFX111)

Notes:

1. If the data_set_name is omitted, OS/390 locates member_name using the default search order for modules in the system library concatenation. The located member_name is used as the character conversion table. For example, if you specify:

   ```
   OMVS CONVERT((BPXFX111))
   ```

   the character conversion table is BPXFX111 in the default module search order in the system library concatenation.

   If the member name is omitted, the OMVS command looks in the specified data_set_name for member FSUMQ000, to use it as the character conversion table. For example, if you specify:

   ```
   OMVS CONVERT('SYS1.XLATE') ...
   ```

   OMVS uses SYS1.XLATE(FSUMQ000) as the character conversion table.

2. FSUMQ000 is an alias; when shipped by IBM, it points to BPXFX100, the default null character conversion table.

Figure 18 lists the character conversion tables supplied with the OMVS command. It shows the locale name, the conversion table to specify, and the default escape character for that table. If you are using the **De_CH.IBM-500** locale, you must specify BPXFX450 as the conversion table, and the default escape character for that particular table is the section sign, §. To specify BPXFX450 as the conversion table, issue:

```
CONVERT((BPXFX450))
```

Figure 18. Locales, Their Conversion Tables, and Default Escape Characters (OMVS command)

Shell and Utilities Locale	3270 Code Page	Shell Code Page	Conversion Table	Default Escape Character
De_CH.IBM-500	IBM-500	IBM-500	BPXFX450	§
De_DE.IBM-273	IBM-273	IBM-273	BPXFX473	§
De_DK.IBM-277	IBM-277	IBM-277	BPXFX477	¤
En_GB.IBM.285	IBM-285	IBM-285	BPXFX485	¯
En_JP.IBM-1027	IBM-1047	IBM-1047	BPXFX100	¢
En_US.IBM-037	IBM-037	IBM-037	BPXFX437	¢
En_US.IBM-1047(For APL terminals)	IBM-037	IBM-1047	BPXFX211	¢
En_US.IBM-1047	IBM-037	IBM-047	BPXFX111	¢
Es_ES.IBM-284	IBM-284	IBM-284	BPXFX484	¨
Fi_FI.IBM-278	IBM-278	IBM-278	BPXFX478	§
Fr_BE.IBM-500	IBM-500	IBM-500	BPXFX450	§
Fr_CA.IBM-037	IBM-037	IBM-037	BPXFX437	¢
Fr_CA.IBM-1047	IBM-037	IBM-1047	BPXFX111	¢
Fr_CH.IBM-500	IBM-500	IBM-500	BPXFX450	§
Fr_FR.IBM-297	IBM-297	IBM-297	BPXFX497	§
Is_IS.IBM-871	IBM-871	IBM-871	BPXFX471	¨
It_IT.IBM-280	IBM-280	IBM-280	BPXFX480	§
Ja_JP.IBM-939	IBM-939	IBM-939	BPXFX100	¢
Ja_JP.IBM-1027	IBM-1027	IBM-1027	BPXFX100	¢
Nl_BE.IBM-500	IBM-500	IBM-500	BPXFX450	§
Nl_NL.IBM-037	IBM-037	IBM-037	IBM-037	¢
Nl_NL.IBM-1047	IBM-037	IBM-1047	BPXFX111	¢
No_NO.IBM-277	IBM-277	IBM-277	BPXFX477	¤
Pt_PT.IBM-037	IBM-037	IBM-037	BPXFX437	¢
Pt_PT.IBM-1047	IBM-037	IBM-1047	BPXFX111	¢
Sv_SE.IBM-278	IBM-278	IBM-278	BPXFX478	§
Xx_XX.**IBM-1047**	IBM-1047	IBM-1047	BPXFX100	¢

DBCS | NODBCS

Specifies whether to use DBCS on 3270-type terminals. The default is DBCS processing.

DBCS

Causes OMVS to automatically determine whether the terminal supports DBCS. If so, DBCS processing takes place. It also enables the OMVS command to handle doublebyte data in translated messages. This operand is ignored if you're not using a DBCS terminal.

Doublebyte data, including escape character strings, cannot be supplied for any of the OMVS command operands. The following data strings used by OMVS must contain singlebyte characters only:

- Escape characters
- Conversion table data set name
- Conversion table member name
- Password used to access the conversion table, if one is required

Also, the ddname for the OMVS debug data set is always SYSFSUMO.

Note: OMVS supports only code pages 939, 1027, and 1047 on DBCS. The null character conversion table (BPXFX100) should be used with DBCS terminals. (It is the default.)

NODBCS
Specifies that OMVS operate in SBCS mode only. If you are logged on to a terminal that supports DBCS, this operand allows you to bypass DBCS processing.

DEBUG(NO | YES | EVENT | DATA)
Controls the collection and output of debugging information. The default is NO; change the default setting only if IBM requests it.

NO
Indicates that no debugging information is to be written.

YES
Indicates that debugging information is collected while the OMVS command runs.

EVENT
Causes additional debugging information to be written whenever certain internal events occur in the OMVS command.

DATA
Causes any data received from or sent to the workstation to be written. Also, debug information for internal events is recorded.

ECHO | NOECHO
Enables OMVS to control the visibility of the input area. The default is NOECHO.

ECHO
Allows OMVS to hide or unhide the input area.

NOECHO
Prevents OMVS from hiding and unhiding the input area.

ENDPASSTHROUGH(ATTN | CLEAR | CLEARPARTITION | ENTER | NO | PA1 | PA3 | PF1 | PF2 | PF3 ... PF24 | SEL)
Specifies a 3270 key that ends TSO/3270 passthrough mode and forces OMVS to return to the shell session. Because this key would be used only during application development, the default is ENDPASSTHROUGH(NO); all 3270 keys can be used by the 3270 application.

ATTN
Specifies the 3270 <Attention> key. In some 3270 applications, this key may be changed to <PA1> before it is seen by the TSO/E OMVS command. If so, OMVS will never see the <Attention> key; specify <PA1>instead of <ATTN>.

With some terminal connections, the <ATTN> key may not be available.

CLEAR
Specifies the 3270 CLEAR key. In some TS0/3270 applications, the TSO/E OMVS command will not see <CLEAR> when the CLEAR key is pressed. In these cases, specifying ENDPASSTHROUGH(CLEAR) will have no effect.

CLEARPARTITION
Specifies the 3270 <Clear Partition> key. This key is effective only if the application is using explicit 3270 partitions.

ENTER
Specifies the 3270 ENTER key. This key is useful only if the 3270 application is completely driven by PF or PA keys.

NO
No breakout key; this is the default.

PA1
Specifies the 3270 <PA1>key. For some TSO/3270 applications, <PA1> is changed to <ATTN> before OMVS sees it. In these cases, you should specify ENDPASSTHROUGH(ATTN).

In general, the provider of the TSO/3270 application needs to tell the user whether <PA1>, <ATTN>, or <CLEAR> can be used for ENDPASSTHROUGH.

PA3
Specifies the 3270 <PA3> key. The <PA3> key may not be available on some keyboards.

PF*n*
Specifies the 3270 function keys 1–9.

PF*nn*
Specifies the 3270 function keys 10–24.

SEL
Specifies the 3270 Cursor Select key. This key is useful only when the 3270 application creates fields on the 3270 screen that can be selected by a light pen.

ESCAPE('escape-characters')
Specifies an escape character as the first character in a two-character sequence that is the EBCDIC equivalent of an ASCII control character (for example, the EBCDIC “ød” is the equivalent of the ASCII “Ctrl-D”). When an escape character is typed in the input area, the next character typed is converted into a special character before it is passed to the shell.

You can enter a string up to eight escape characters, enclosed in single quotes with no space between them. (Do not use nonprintable EBCDIC characters.)

The default escape character depends on the character conversion table being used. (See Figure 18 on page 765 for a list of default characters and the conversion tables they are used with.) To enter <Ctrl-D>, for example, type in `ød` or `øD` in the input area.

If the last character in the input area is one of the escape characters, the <newline> character normally appended to the input data is suppressed. For example, to enter only a <Ctrl-Q> with no final <newline>, type the string `øQø` in the input area, and press <Enter>.

LINES(n)
Controls the amount of output data the OMVS command keeps for scrolling. The default is roughly four screenfuls. You can specify that between 25 and 3000 lines should be kept in the output buffer.

PFn(ALARM | NOALARM | AUTOSCROLL | NOAUTOSCROLL | BACKSCR | BOTTOM | CLOSE | CONTROL | ECHO | NOECHO FWDRETR | HALFSCR | HELP | HIDE | NOHIDE NEXTSESS | NO | OPEN | PFSHOW | NOPFSHOW | PREVSESS | QUIT | QUITALL | REFRESH | RETRIEVE | RETURN | SCROLL | SUBCOMMAND | TOP | TSO)
Customizes the settings for the function keys that you use while working in the OS/390 shell or in subcommand mode. in <PF*n*> The *n* is a one- or two-digit function key number from 1 to 24. Do not use a leading zero for a one-digit number. More than one function key can be assigned the same function. For example, both <PF1> and <PF13> are assigned the Help function by default.

All PF keys can be abbreviated using the usual TSO/E rules. For example,

- OPEN can be abbreviated as O, OP, or OPE.
- NEXTSESS can be abbreviated as NE, NEX, NEXT, NEXTS, NEXTSE, or NEXTSES.
- PFSHOW can be abbreviated as PF, and NOPFSHOW can be abbreviated as NOPF.

ALARM | NOALARM
A toggle key used to turn on and off the 3270 alarm that sounds when an <alert> character is written to the output area (also available in subcommand mode).

The label for this PF key (in the PF key lines at the bottom of the screen) shows up as either ALARM or NOALARM, depending on the current toggle setting. If it is ALARM, pressing this PF key turns the alarm on. If it is NOALARM, pressing this PF key turns the alarm off.

AUTOSCROLL | NOAUTOSCROLL
A toggle key used to turn the autoscroll function on and off (also available in subcommand mode). The screen automatically scrolls forward when new input is written to the screen.

The label for this PF key (in the PF key lines at the bottom of the screen) shows up as either AUTOSCROLL or NOAUTOSCROLL, depending on the current toggle setting. If it is AUTOSCROLL, pressing this PF key turns the autoscroll function on. If it is NOAUTOSCROLL, pressing this PF key turns the autoscroll function off.

BACKSCR
Scrolls the screen backward one full screen, redisplaying previously displayed output lines. The scrolling ends when the oldest available saved line is reached. (This option is also available in subcommand mode.)

If you first move the cursor into the output area, the line with the cursor becomes the top line.

BOTTOM
Scrolls help information forward to the last panel of information, and scrolls output forward the last full screen (also available in subcommand mode).

CLOSE
Ends the displayed session and switches to another one, or returns to TSO/E if the only session was closed (also available in subcommand mode).

CONTROL
Treats all characters in the input area as if they were preceded by an escape character. Also, no trailing <newline> is appended to the data.

ECHO | NOECHO
A toggle key used to control whether the shell command can hide or unhide the OMVS command input area.

The label for this PF key (in the PF key lines at the bottom of the screen) shows up as either ECHO or NOECHO, depending on the current toggle setting. If it is ECHO, pressing this PF key allows the current shell command to hide or unhide the OMVS command input area. If it is NOECHO, pressing this PF key prevents the current shell command from hiding or unhiding the OMVS input area.

FWDRETR
Retrieves the oldest available input line from a stack of saved input lines, starting with the oldest and moving up to the most recent line (also available in subcommand mode).

HALFSCR
Scrolls half the displayed screen forward, allowing room for more output data. If the output area on the screen is not full, half the displayed lines are scrolled off the screen. If you first move the cursor into the output area, the line with the cursor becomes the middle line. (This option is also available in subcommand mode.)

HELP
Temporarily suspends the session and displays the help information for the OMVS command. The scrolling function keys can be used to look at the help information. To exit the help information, press the Return function key. (This option is also available in subcommand mode.)

HIDE | NOHIDE
Temporarily hides or unhides the input data you type on the shell command line. If you press this PF key while the input area is hidden, the input area is made visible. If it is not hidden, the input area is hidden.

The input area stays hidden or unhidden until:

- You press <Enter>.
- You press the HIDE | NOHIDE PF key.
- You switch to another session, escape to TSO/E and return, or enter subcommand mode and return.

If OMVS is running in NOECHO mode, the input area will be visible after you take one of these actions. If OMVS is running in ECHO mode, the visibility of the input area depends on the shell command you are running.

NEXTSESS
Switches to the next (higher-numbered) session (also available in subcommand mode).

NO
Deactivates a function key so that it doesn't do anything (also available in subcommand mode).

OPEN
Starts a new shell session and switches to it (also available in subcommand mode).

PFSHOW | NOPFSHOW
Toggles on and off the display of the active function key settings at the bottom of the screen (also available in subcommand mode, and can be used as PF and NOPF).

PREVSESS
Switches to the previous (lower-numbered) session (also available in subcommand mode).

QUIT
Ends the displayed session and switches to another one, or returns to TSO/E if the only session was closed (also available in subcommand mode).

QUITALL
Ends all shell sessions and causes OMVS to end and to return to TSO/E (also available in subcommand mode).

REFRESH
Updates the screen with the latest output data. Use this function key if the display of output is incomplete, but the session is now displaying `INPUT` status. For more information on the status field, see *OS/390 UNIX System Services User's Guide*. (This option is also available in subcommand mode.)

RETRIEVE
Retrieves the most recently entered input line from a stack of saved input lines, starting with the most recent and moving down to the oldest available line (also available in subcommand mode).

RETURN
If help information is displayed, returns you to the session you were in. If you are in subcommand mode, returns you to the shell. (This option is also available in subcommand mode.)

SCROLL
Scrolls the last line of output data to the top of the screen, making room for more output data. If Help information is displayed, its data is scrolled. If you first move the cursor into the output area, the line with the cursor becomes the top line. (This option is also available in subcommand mode.)

SUBCOMMAND
If you press this key when the command line is blank, it leaves the shell session and enters subcommand mode.

To run a subcommand without switching to subcommand mode, type the subcommand at the command line and then press the function key. You can enter the OMVS subcommands at the command line when you are in subcommand mode.

TOP
Scrolls help information backward to the first panel, and scrolls output backward to a screenful of the oldest available output (also available in subcommand mode).

TSO
If you press this key when the command line is blank, it temporarily suspends a shell session or subcommand mode, and you are in a TSO/E session. You can enter TSO/E commands. Press <PA1> or the <Attention> key to exit TSO/E command mode and return to the session you were in. (This option is also available in subcommand mode.)

To run a TSO/E command without suspending the shell session or subcommand mode, type the command at the command line and then press the function key. When the command completes, you can continue working in the shell session or subcommand mode.

Function Key Defaults:

PF1(HELP)
PF2(SUBCOMMAND)
PF3(RETURN)
PF4(TOP)
PF5(BOTTOM)
PF6(TSO)
PF7(BACKSCR)
PF8(SCROLL)
PF9(NEXTSESS)
PF10(REFRESH)
PF11(FWDRETR)
PF12(RETRIEVE)
PF13(HELP)
PF14(SUBCOMMAND)
PF15(RETURN)
PF16(TOP)
PF17(BOTTOM)
PF18(TSO)
PF19(BACKSCR)
PF20(SCROLL)
PF21(NEXTSESS)
PF22(REFRESH)
PF23(FWDRETR)
PF24(RETRIEVE)

PFSHOW | NOPFSHOW
Specifies that the PF keys be shown at the bottom of the screen. The default is PFSHOW.

PFSHOW
Specifies that PF keys be shown at the bottom of the screen.

NOPFSHOW
Specifies that PF keys not be shown at the bottom of the screen.

RUNOPTS('LE/370–runtime-options')
Specifies a string containing LE/370 runtime options, which are passed to LE/370 when the TSO/E OMVS command starts up, and to the initial login shell

program in the _CEE_RUNOPTS environment variable. These options are the same as those passed to other LE/370 programs run from the TSO READY prompt.

The options string can be from 1 to 1000 characters in length, and should contain valid LE/370 runtime options. It should not contain options such as POSIX(OFF), TRAP(OFF), or MSGFILE(); or characters such as slashes, unbalanced parentheses or quotes, or imbedded NULL characters. Specifying such options or using these characters will cause unpredictable problems when the TSO/E OMVS command runs.

If the RUNOPTS operand is omitted, OMVS uses the RUNOPTS string defined in the BPXPRMxx SYS1.PARMLIB member that is active for the OMVS kernel. If no RUNOPTS string was defined in BPXPRMxx, no LE/370 runtime options are used when the TSO/E OMVS command starts up.

SESSIONS(n)

Specifies the initial number of sessions to be started. The default is 1, and the allowed range is 1 to 100; most users will use two or three sessions.

Note: You can specify a number from 1 to 100 without getting a syntax error on the command. Normally, you cannot start more than several sessions before getting an error message. If you try to start too many sessions (the limit depends on the size of your TSO/E address space), your TSO/E user ID runs out of storage and various unpredictable errors may occur. You may have to log off your TSO/E user ID before you can continue.

SHAREAS | NOSHAREAS

Specifies whether to run the shell program in a separate address space. Both OMVS and the shell will run in the TSO/E address space when OMVS is invoked with the SHAREAS parameter.

OMVS will use SHAREAS as the default if the shell program is not a SETUID or SETGID program and the owning UID or GID is not the same as the current user.

SHAREAS

Runs the shell program in the same TSO/E address space as OMVS. SETUID and SETGID shell programs cannot be run with the SHAREAS option unless your UID or GID owns the shell program.

Note: If you end OMVS while in SHAREAS mode, the shell process ends immediately. (It may get killed, but it will usually end by itself when the TTY is closed.)

NOSHAREAS

Runs the shell program in a separate address space. SETUID and SETGID shell programs usually require this option.

WRAPDEBUG(n)

Controls how many lines of debug data OMVS writes out before wrapping around to the top of the debug data set. This option is effective only if the **DEBUG(YES) DEBUG(EVENT)**, or **DEBUG(DATA)** options are used.

The **WRAPDEBUG(n)** value specifies how many lines of debug data OMVS writes out before wrapping around to the top of the debug data set. The default number of lines is 10 000. The value of **n** must be between 100 and 1 000 000 000. The debug data set must be large enough to hold **n** 80-byte

lines of debug data. If the debug data set is too small, debug recording stops when the data set fills up.

Subcommands

When the shell is active, you can enter subcommand mode by pressing the Subcommand function key. While in subcommand mode, you can enter subcommands on the command line or use function keys.

ALARM
Turns on the 3270 alarm which sounds when an <alert> character is written to the output area.

AUTOSCROLL
Activates automatic forward scrolling of output as new input is written to the screen.

BACKSCR
Scrolls the screen backward one full screen, redisplaying previously deleted output lines. The scrolling ends when the oldest available saved line is reached.

BOTTOM
If the help information is displayed, it is scrolled forward to the last panel of information. If output is displayed, it is scrolled forward to the last screen of output.

CLOSE
Ends the displayed session and switches to another one, or returns to TSO/E if the only session was closed.

ECHO
Allows the current shell command to control whether the OMVS input area is visible or hidden. The HIDE subcommand, NOHIDE subcommand, and HIDE | NOHIDE PF keys can temporarily override the input area visibility set by the current shell command.

HALFSCR
Scrolls half the displayed screen forward, allowing room for more output data.

HELP
Displays help information for the OMVS command. To view the help information, use the scrolling function keys. To return from Help to the session, press the Return function key.

? is a short form for the Help subcommand.

HIDE
Temporarily hides the input data you type on the shell command line. The input area stays hidden until:

- You press <Enter>.
- You press the HIDE | NOHIDE PF key.
- You switch to another session, escape to TSO and return, or enter subcommand mode and return.

If OMVS is running in NOECHO mode, the input area will be visible after you take one of these actions. If it is running in ECHO mode, whether you can see the input area depends on the shell command you are running.

NEXTSESS
Switches to the next (higher-numbered) session.

NOALARM
Prevents the 3270 alarm from sounding when the <alert> character is encountered in data being sent to the workstation.

NOAUTOSCROLL
Turns off the automatic scrolling (AUTOSCROLL) function.

NOECHO
Causes the OMVS input area to remain visible regardless of the current shell command. You can use the HIDE subcommand and the PF key to temporarily hide the input area.

NOHIDE
Temporarily unhides the input data you type on the shell command line. The input area remains visible until:

- You press <Enter>.
- You press the HIDE | NOHIDE PF key.
- You switch to another session, escape to TSO and return, or enter subcommand mode and return.

If OMVS is running in NOECHO mode, the input area remains visible after you take one of these actions. If OMVS is running in ECHO mode, the visibility of the input area depends on the shell command you are running.

NOPFSHOW
Turns off the display of the function key settings and escape characters at the bottom of the screen.

OPEN
Starts a new shell session and switches to it.

PFSHOW
Displays the current function key settings and escape characters on the bottom two lines of the display screen. A maximum of two screen lines is used. If some function key settings do not fit on the two lines, they are not displayed.

PREVSESS
Switches to the previous (lower-numbered) session.

QUIT
Ends the displayed session and switches to another one, or returns to TSO/E if the only session was closed.

QUITALL
Ends all shell sessions and causes OMVS to end and to return to TSO/E.

RETURN
Returns from subcommand mode to the shell session. If help information is being displayed, the session returns to subcommand mode and you must enter the RETURN command again to return to the shell.

SCROLL
Scrolls forward the data displayed on the screen, approximately one full screen.

TOP
Scrolls help information backward to the first panel. Scrolls output backward to a screenful of the oldest available output.

TSO
Invokes TSO/E command mode. In this mode, you can enter TSO/E commands. Press <PA1>or the <Attention> key to return to subcommand mode.

Usage Notes

1. The OMVS command is a Language Environment application. OMVS overrides the default Language Environment; MSGFILE ddname (SYSOUT) and uses ddname SYSFSUMM. (See the Language Environment; books for more information about the MSGFILE runtime option.)

   Normally, any Language Environment; error messages from the OMVS command are displayed on the TSO/E terminal. If you want to redirect these messages, you need to allocate the SYSFSUMM ddname instead of the SYSOUT ddname, as is usual with Language Environment; applications.

2. The language of the OMVS command messages is determined by the PROFILE PLANGUAGE setting when OMVS is invoked. Do not change PROFILE PLANGUAGE while OMVS is running

Return Codes

0 Processing successful.
12 Processing unsuccessful. An error message has been issued.

Examples

These examples explain how to use the multisession capability of OMVS:

1. To start *2* sessions automatically when starting OMVS, enter:

   ```
   OMVS SESSIONS(2)
   ```

2. To assign the NEXTSESS function to a PF key, enter:

   ```
   OMVS PF1(NEXTSESS)
   ```

OPUT — Copy an MVS data set member into an HFS file

Format

```
OPUT  mvs_data_set_name | mvs_data_set_name(member_name)
      'pathname'
      BINARY | TEXT
      CONVERT(character_conversion_table | YES | NO)
```

Description

You can use the OPUT command to:

- Copy a member of an MVS partitioned data set (PDS or PDSE) to a file
- Copy an MVS sequential data set to a file

and convert the data from code page IBM-037 or ASCII to code page IBM-1047.

Do not use the CONVERT option when copying files that contain doublebyte data. This option is used for singlebyte data only, not for doublebyte data.

Parameters

mvs_data_set_name | mvs_data_set_name(member_name)
Specifies the name of an MVS sequential data set or an MVS partitioned data set member that is being copied.

- A fully qualified name that is enclosed in single quotes, or an unqualified name (an unqualified name is not enclosed in single quotes)
- Up to 44 characters long, with an additional 8 characters for the member name
- Converted to uppercase characters by the system

pathname
Specifies the pathname of the file to receive the data set member that is being copied. The target file cannot be a directory. All directories in the pathname prior to the filename directory must already exist. The pathname is:

- A relative or absolute pathname. A relative pathname is relative to the working directory of the TSO/E session (usually the **HOME** directory). Therefore, you should usually specify an absolute pathname.
- Up to 1023 characters long.
- Enclosed in single quotes.
- In uppercase or lowercase characters, which are not changed by the system.

BINARY | TEXT
specifies that the data set being copied contains binary data or text.

BINARY
Specifies that the data set being copied contains binary data. This is the default for a data set of undefined record format.

TEXT
Specifies that the data set being copied contains text. This is the default for a data set of fixed record format or variable record format.

CONVERT(character_conversion_table | YES | NO)
Specifies that the data being copied is to be converted from IBM-037 or ASCII to EBCDIC Latin 1/Open Systems Interconnection code page 01047—that is, that the TO1047 part of the specified character conversion table will be used. This operand is optional. If this operand is omitted, the system copies the data without conversion.

You can use this option for singlebyte data, but not for doublebyte data.

Specify the CONVERT value as one of the following:

character_conversion_table
Specify one of the following:

- **data_set_name(member_name)**. Specifies the name of the partitioned data set (PDS) and the name of the member that contains the character conversion table.
- **data_set_name**. Specifies the name of the partitioned data set (PDS) that contains the character conversion table. The table is the FROM1047 part in member BPXFX000. (This is an alias; when shipped by IBM, it points to BPXFX111.)
- **(member_name)**. Specifies the name of the conversion table to be used. It is a member of a PDS. Since the data_set_name is omitted, the standard library concatenation is searched for the table. (The default library is SYS1.LINKLIB.)

  The following list summarizes what you can specify when you want to convert data to a different code page when copying singlebyte data:

  – BPXFX100. Null character conversion table. Use this table if the square brackets at your workstation are at the same code points as the square brackets on code page 1047 (it is the default). Also use it if you are using a DBCS terminal.

  – BPXFX111. Specifies a non-APL conversion table to convert between code pages IBM-037 and IBM-1047.

  – BPXFX211. Specifies an APL conversion table to convert between code pages IBM-037 and IBM-1047.

  – BPXFX311. Specifies an ASCII-EBCDIC conversion table to convert between code pages ISO8859-1 and IBM-1047.

YES
The system will perform conversion and use the default conversion table (BPXFX000) in the system library concatenation. (BPXFX000 is an alias; when shipped by IBM, it points to BPXFX111.)

NO
Specifies no conversion. NO is the same as omitting the CONVERT operand.

Usage Notes

1. If the specified file does not exist, OPUT creates a new file. For a new text file, the mode (permission bits) is 600. When the mode is 600, the user has read and write access; others have nothing. For a new binary file, the mode (permission bits) is 700. When the mode is 700, the user has read, write, and search access; others have nothing.
2. If the specified file exists, the new data overwrites the existing data. The mode of the file is unchanged.
3. You can use OPUT to copy a program object from a PDSE to the file system, and it will be executable there. If you have a load module in a partitioned data set, however, you must first use the IEBCOPY program to copy the load module from a partitioned data set to a PDSE and then subsequently use OPUT to copy the module into the file system. IEBCOPY converts load

modules to a program object. See *OS/390 UNIX System Services User's Guide* for a discussion of copying executable files.

4. Data sets with spanned record lengths are not allowed.
5. When you copy MVS data sets to HFS text files, a <newline> character is appended to the end of each record. If trailing blanks exist in the record, the <newline> character is appended after the trailing blanks.
6. When you copy MVS data sets to HFS binary files, the <newline> character is not appended to the record.

Return Codes

0 Processing successful.
12 Processing unsuccessful. An error message has been issued.

Examples

1. This command copies an MVS sequential data set to a file, converting from code page IBM-037 to code page 1047.
   - The unqualified name of the sequential data set is EMPLOYEE.DATA.
   - The pathname of the file is **/u/admin/employee/data**.

   ```
   OPUT EMPLOYEE.DATA '/u/admin/employee/data' TEXT CONVERT(YES)
   ```

2. This command copies an MVS sequential data set to a file converting to code page 1047 using the conversion table BPXFX000 in the user's library data set.
   - The fully qualified name of the sequential data set is IBMUSR.EMPLOYEE.DATA.
   - The pathname of the file is **/u/admin/employee/data**.

   ```
   OPUT 'IBMUSR.EMPLOYEE.DATA' '/u/admin/employee/data'
          TEXT CONVERT(MY.LOADLIB(BPXFX000))
   ```

3. This command copies a binary file from a PDSE to a file in the file system.
   - APPL.LOADLIB(PAYROLL) is the fully qualified name of the member of the PDSE.
   - **bin/payroll** is the pathname of the file; the directory **bin** is in the working directory.
   - There is no code page conversion.

   ```
   OPUT 'APPL.LOADLIB(PAYROLL)' '/bin/payroll' binary
   ```

OPUTX — Copy members from an MVS PDS or PDSE to an HFS directory

Format

```
OPUTX  mvs_PDS_name | mvs_data_set-name(member_name)
       hfs_directory | hfs_file_name
       ASIS
       BINARY | TEXT
       CONVERT(character_conversion_table | YES | NO)
       LC
       MODE
       QUIET
       SUFFIX(suffix)
```

Description

You can use the OPUTX command to:

- Copy members from an MVS partitioned data set (PDS) or PDSE to an HFS directory
- Copy a sequential data set or member of a PDS to a file

and convert the data from code page IBM-037 or ASCII to code page IBM-1047 while it is being copied.

Do not use the CONVERT option when copying files that contain doublebyte data. This option is used for singlebyte data only, not for doublebyte data.

Parameters

hfs_directory | HFS_file_name
Specifies the directory name or filename of the HFS files to receive the PDS members that are being copied. The name can be up to 1023 characters long. Single quotes around the directory name or filename are optional.

Use `hfs_directory` when a PDS is specified. When a sequential data set or PDS member is specified, then the filename must be used.

mvs_PDS_name | mvs_data_set_name(member_name)
Specifies the name of an MVS partitioned data set or an MVS partitioned data set member that is being copied into an HFS. The data set name is:

- A fully qualified name that is enclosed in single quotes, or an unqualified name (an unqualified name is not enclosed in single quotes)
- Up to 44 characters long, with an additional 8 characters for the member name
- Converted to uppercase letters

ASIS
Specifies that the @ character in member names *not* be translated to the _ character in pathnames. (It is a common convention to use @ symbols in PDS member names to correspond with the _ symbol in pathnames.)

BINARY | TEXT
Specifies whether the data set being copied contains binary data or text.

BINARY
Specifies that the data set being copied contains binary data. This is the default for a data set of undefined record format.

TEXT
Specifies that the data set being copied contains text. This is the default for a data set of fixed record format or variable record format.

CONVERT(character_conversion_table | YES | NO)
Specifies that the data being copied be converted from code page IBM-037 to EBCDIC Latin 1/Open Systems Interconnection code page 01047—that is, that the TO1047 part of the specified character conversion table is used. This operand is optional. If this operand is omitted, the system copies the data without conversion.

You can use this option for singlebyte data, but not for doublebyte data.

Specify the CONVERT value as one of the following:

character_conversion_table
Specify one of the following:

- **data_set_name(member_name)**. Specifies the name of the partitioned data set (PDS) and the name of the member that contains the character conversion table.
- **data_set_name**. Specifies the name of the partitioned data set (PDS) that contains the character conversion table. The table is the FROM1047 part in member BPXFX000. (This is an alias; when shipped by IBM, it points to BPXFX111.)
- **(member_name)**. Specifies the name of the conversion table to be used. It is a member of a PDS. Since the data_set_name is omitted, the standard library concatenation is searched for the table. (The default library is SYS1.LINKLIB.)

  The following list summarizes what you can specify when you want to convert data to a different code page when copying singlebyte data:

  – BPXFX100. Null character conversion table. Use this table if the square brackets at your workstation are at the same code points as the square brackets on code page 1047 (it is the default). Also use it if you are using a DBCS terminal.

  – BPXFX111. Specifies a non-APL conversion table to convert between code pages IBM-037 and IBM-1047.

  – BPXFX211. Specifies an APL conversion table to convert between code pages IBM-037 and IBM-1047.

  – BPXFX311. Specifies an ASCII-EBCDIC conversion table to convert between code pages ISO8859-1 and IBM-1047.

YES
Specifies that the system is to perform conversation and use the default conversion table (BPXFX000) in the standard library concatenation. (BPXFX000 is an alias; when shipped by IBM, it points to BPXFX111.)

NO
Specifies that conversion not be done. NO is the same as omitting the CONVERT operand.

LC
Specifies that the member name be converted to a lowercase filename.

MODE
Specifies the file mode for any members copied into the HFS. The mode can be specified as three or four octal digits. (The digits can be separated by commas.) Invalid mode specifications are ignored.

If the specified file does not exist, OPUTX creates a new file. For a new text file, the mode (permission bits) is 600. When the mode is 600, the user has read and write access; others have none. For a new binary file, the mode (permission bits) is 700. When the mode is 700, the user has read, write, and search access; others have none.

For more information, see *OS/390 UNIX System Services User's Guide*.

QUIET
Turns off the echoing of the OPUTX commands before the member or data set is copied.

SUFFIX(suffix)
Specifies that a suffix specified by (suffix) be appended to the member names in creating the HFS filenames.

A suffix is an optional additional file identifier that is appended to the filename following a period (**.**). It is usually used to identify the type of file. For example, **.c** usually indicates a C language source file and **.h** indicates a C language header file. Suffixes can be any length and you can append as many as you want, but the filename, including suffixes, cannot exceed 255 characters for OS/390 UNIX System Services.

Usage Notes

1. Avoid using OPUTX with pathnames containing quotes or spaces.
2. If the specified file does not exist, OPUTX creates a new file. For a new text file, the mode (permission bits) is 600. When the mode is 600, the user has read and write access; others have nothing. For a new binary file, the mode (permission bits) is 700. When the mode is 700, the user has read, write, and search access; others have nothing.
3. If the specified file exists, the new data overwrites the existing data. The mode of the file is unchanged.
4. Data sets with spanned records are not allowed.
5. When you copy MVS data sets to HFS text files, a <newline> character is appended to the end of each record. If trailing blanks exist in the record, the <newline> character is appended after the trailing blanks.
6. When you copy MVS data sets to HFS binary files, the <newline> character is not appended to the record.
7. Prior to the copy, the OPUTX command for a data set or member is echoed, unless you specify the QUIET option. If you did not specify QUIET and if the command is not displayed, the data set or member is not copied.
8. If the source data set is a PDS with an undefined record format, the members may be treated as load modules. A load module is copied by link-editing it into the target file in the file hierarchy. For the program to be able to run from the file hierarchy, the entry point must be at the beginning of the load module.

   For OPUTX to treat the file as a load module, neither BINARY or TEXT can be specified.

9. If the source for the copy is a sequential data set or a PDS member and the target is a directory, the filename used is the last qualifier of the data set name or the member name. You do not have to specify a file as the target with a sequential data set, or a directory as the target with a PDS. The LC and ASIS options are not affected.

Examples

The following command copies files in a PDS into a HFS directory and specifies that:

- The name of the partitioned data set (PDS) is DATAFILE
- The HFS directory is **/usr/sbllib**
- The files are given a suffix of **.c**

```
OPUTX DATAFILE '/usr/sbllib/' LC SUFFIX(c)
```

Assuming the PDS has members PROGRAM1, PROGRAM2, and PROGRAM3, these members are copied as **/usr/sbllib/program1.c**, **/usr/sbllib/program2.c**, and **/usr/sbllib/program3.c**.

OSHELL — Invokes BPXBATCH from TSO/E

Format

```
OSHELL
```

Description

OSHELL uses BPXBATCH to run the shell command or shell script:

```
oshell shell_command
```

For example, to display process information, enter:

```
oshell ps –ej
```

When you use OSHELL, do not use an & to run a shell command in the background. For more information about BPXBATCH, see Appendix D, “Running Shell Scripts or Executable Files under MVS Environments” on page 803

Some examples of using the OSHELL command are:

- List files in a directory
- Create, delete, or rename directories, files, and special files
- Display contents of a file
- Copy files
- Display file attributes
- Search files for text strings
- Compare files or directories
- Run executable files
- Display the attributes and contents of a symbolic link (symlink)
- Set up character special files
- Set up standard directories for a root file system

Some of these tasks may require superuser authority.

OSTEPLIB — Build a list of files

Format

```
OSTEPLIB pathname
```

Description

Use the OSTEPLIB command to build a list of files that are sanctioned by your installation as valid step libraries for programs that have the set-user-ID or set-group-ID bit set. This permission setting allows a program to have temporary access to files that are not normally accessible to other users. Step libraries have many uses; for example, selected users can test new versions of runtime libraries before the new versions are made generally available.

You must have superuser authority to issue OSTEPLIB.

The sanctioned list is valid if it conforms to the following rules:

- You can include comment lines in the list. Each comment line must start with /* and end with */.
- You must follow standard MVS data set naming conventions in naming the files in the list.
- Each data set name must be fully qualified and cannot be enclosed in quotation marks.
- Each data set name must be on a line by itself, with no comments.
- You can put blanks before and after each data set name. Entirely blank lines in the list are ignored.
- You can use the * character to specify multiple files that begin with the same characters. For example, if you list SYS1.*, you are sanctioning any file that begins with *SYS1.* as a step library.

Following is an example of a file that contains a correctly formatted list of sanctioned step libraries:

```
/*****************************************************************/
/*                                                               */
/*Name: Sample Sanctioned List for set-user-ID and set-group-ID  */
/*      files                                                    */
/*                                                               */
/*Updated by:   May only be updated by OSTEPLIB TSO/E command    */
/*                                                               */
/*Description:  Contains a list of data set names that may       */
/*              be used as STEPLIB libraries for SETUID          */
/*              programs                                         */
/*                                                               */
/*              Wild cards may be used to specify multiple       */
/*              data set names that have the same prefix         */
/*              characters.                                      */
/*                                                               */
/*****************************************************************/

/*****************************************************************/
/*Sanction all data set names beginning with SYS1.CEE            */
/*****************************************************************/
SYS1.CEE*

/*****************************************************************/
/*Sanction data set containing vers. 2 of the C run time library */
/*****************************************************************/
ADMIN.CEE.RTLV2
```

Parameters

pathname

Specifies the pathname of the file to contain the list of sanctioned step libraries. The pathname can be absolute or relative to the root. Avoid using the space character or single quotation mark (apostrophe) within the pathname. The pathname cannot be enclosed in single quotes.

If you omit the pathname operand, the new sanctioned list file is created with the same filename as the old one and replaces it when it has been validated.

UNMOUNT — Remove a file system from the file hierarchy

Format

```
UNMOUNT FILESYSTEM(file_system_name)
        DRAIN | FORCE | IMMEDIATE | NORMAL | REMOUNT(RDWR | READ) | RESET
```

Description

The UNMOUNT command removes a file system from the file system hierarchy. The alias for this command is UMOUNT.

Note: An UNMOUNT user must have UID (0) or at least have READ access to the BPX.SUPERUSER FACILITY class.

Parameters

FILESYSTEM(file_system_name)
Specifies the name of the file system to be removed from the file system. The name supplied is changed to all uppercase characters. This operand is required.

file_system_name
For the hierarchical file system (HFS), this is the fully qualified name of the MVS HFS data set that contains the file system. The file system name supplied is changed to all uppercase characters.

Specify the name of file system exactly as it was specified when the file system was originally mounted. You can enclose it in single quotes, but they are not required.

If FILESYSTEM(''''file_system_name'''') is specified, the file system name will not be translated to uppercase.

DRAIN
Specifies that an unmount drain request is to be made. The system will wait for all use of the file system to be ended normally before the unmount request is processed or until another UNMOUNT command is issued.

Note: UNMOUNT can be specified with IMMEDIATE to override a previous UNMOUNT DRAIN request for a file system. If this is used in the foreground, your TSO/E session waits until the UNMOUNT request has completed. <ATTN> (or <PA1>) does not terminate the command.

UNMOUNT DRAIN is not supported in a sysplex environment. If an UNMOUNT DRAIN is issued in a sysplex, the following behavior is exhibited:

- If there is no activity in the file system, UNMOUNT DRAIN will perform the unmount, but it will behave like an UNMOUNT NORMAL.
- If there is activity in the file system, UNMOUNT DRAIN will return a Return_value of -1 with Return_code EINVAL and Reason_code JrNotSupInSysplex.

FORCE
Specifies that the system is to unmount the file system immediately. Any users accessing files in the specified file system receive failing return codes. All data changes to files in the specified file system are saved, if possible. If the data changes to the files cannot be saved, the unmount request continues and data is lost.

Note: An UNMOUNT IMMEDIATE request must be issued before you can request a UNMOUNT FORCE of a file system. Otherwise, UNMOUNT FORCE fails.

IMMEDIATE
Specifies that the system is to unmount the file system immediately. Any users accessing files in the specified file system receive failing return codes. All data changes to files in the specified file system are saved. If the data changes to files cannot be saved, the unmount request fails.

NORMAL
Specifies that if no user is accessing any of the files in the specified file system, the system processes the unmount request. Otherwise, the system rejects the unmount request. This is the default.

REMOUNT(RDWR|READ)
Specifies that the specified file system be remounted, changing its mount mode. REMOUNT takes an optional argument of RDRW or READ. If you specify either argument, the filesystem is remounted in that mode if it is not already in that mode. If you specify REMOUNT without any arguments, the mount mode is changed from RDWR to READ or READ to RDWR.

Note: REMOUNT is not supported in a sysplex. An alternate way to remount a file system in a sysplex is to UNMOUNT the file system and then MOUNT it again in the desired mode.

RESET
A reset request stops a previous UNMOUNT DRAIN request.

Note: UNMOUNT RESET is not supported in a sysplex since UNMOUNT DRAIN is unsupported in a sysplex environment (see description above).

Usage Notes

1. The directory **/samples** contain sample UNMOUNT commands (called **unmountx**).
2. If you unmount a TFS file system, all data stored in that file system is discarded. For more information on TFS, see *OS/390 UNIX System Services Planning*.
3. If desired, the root file system can be umounted. If this is done, the IMMED operand must be specified. Unmounting the root will force all file system activity to be halted. Therefore, if this is done, a subsequent mount of a root file system should be done as soon as possible.
4. While the root file system is unmounted, a 'dummy file system' root named SYSROOT will be displayed as the current root file system. During this time, any operation that requires a valid file system will fail. When the new root file system is subsequently mounted, you should terminate any currently dubbed users or issue a **chdir** using a full pathname to the appropriate directory so that the users can access the new root file system.
5. If the file system that you are unmounting is an NFS-supported file system, the UNMOUNT command may receive an EAGAIN return code if the request was made before an internal caching clock has expired. That is, there is a 60-second delay from last use before termination is possible. Retry the request.
6. A file system that has file systems mounted on it cannot be unmounted. Any children file systems must be unmounted first.

Return Codes

0 Processing successful.
12 Processing unsuccessful. An error message has been issued.

Examples

1. The following command specifies a normal unmount by default:

```
UNMOUNT FILESYSTEM('HFS.WORKDS')
```

2. Before you request a forced unmount of a file system, you must issue an immediate unmount request:

```
UNMOUNT FILESYSTEM('HFS.WORKDS') IMMEDIATE
UNMOUNT FILESYSTEM('HFS.WORKDS') FORCE
```

3. To unconditionally change the mount mode of a file system:

   ```
   UNMOUNT FILESYSTEM(HFS.OMVS.BIN) REMOUNT
   ```

4. To change the mount mode of a file system to read/write, provided it is currently read-only:

   ```
   UNMOUNT FILESYSTEM(HFS.OMVS.BIN) REMOUNT(RDWR)
   ```

Appendix A. OS/390 Shell Command Summary

The following list presents OS/390 shell commands and utilities grouped by the task a user might want to perform. Similar tasks are organized together. Stub commands (**cancel**, **cu** and **lpstat**) are not listed because their functions are not supported by OS/390 UNIX System Services.

The list also shows the command name, the standard or specification it satisfies, and its function. XPG4.2 refers to "X/Open CAE Issue 4 Version 2 Specifications." XPG5.0 refers to "X/Open CAE Issue 5 Specifications."

General Use

at	POSIX.2	XPG4.2	Run a command at a specified time
batch	POSIX.2	XPG4.2	Run commands when the system is not busy
bpxmtext	—	—	Display reason code text
command	POSIX.2	XPG4.2	Run a simple command
confighfs	—	—	Invoke vfs_pfsctl HFS functions
date	POSIX.2	XPG4.2	Display the date and time
echo	POSIX.2	XPG4.2	Write arguments to standard output
exec	POSIX.2	XPG4.2	Run a command and open, close, or copy the file descriptors
man	POSIX.2	XPG4.2	Print sections of the online reference manual
nice	POSIX.2	XPG4.2	Run a command at a different priority
passwd	—	—	Change user passwords
print	—	—	Return arguments from the shell
printf	POSIX.2	XPG4.2	Write formatted output
sh	POSIX.2	XPG4.2	Invoke a shell (command interpreter)
tcsh	—	—	Invoke a tcsh shell
time	POSIX.2	XPG4.2	Display processor and elapsed times for a command
wall	—	—	Broadcast a message to logged-in users
whence	—	—	Tell how the shell interprets a command name
whoami	—	—	Display your effective username

Controlling Your Environment

alias	POSIX.2	XPG4.2	Display or create a command alias
asa	POSIX.2	XPG4.2	Interpret ASA/Fortran carriage control
automount	—	—	Configure the automount facility
cal	—	XPG4.2	Display a calendar for a month or year
calendar	—	XPG4.2	Display all current appointments
captoinfo	—	—	Prints terminal entries in the termcap file
chcp	—	—	Set or query ASCII/EBCDIC code pages for the terminal
configstk	—	—	Configure the AF_UENT stack
env	POSIX.2	XPG4.2	Display environments, or set an environment for a process
export	POSIX.2	XPG4.2	Set the export attributes for variables, or show currently exported variables
fc	POSIX.2	XPG4.2	Process a command history list
hash	—	XPG4.2	Create a tracked alias
history	—	—	Process a command history list
id	POSIX.2	XPG4.2	Return the user identity
infocmp	—	—	Compare and print the terminal description
ipcrm	—	—	Remove message queue, semaphore set, or shared memory identifiers

© Copyright IBM Corp. 1996, 2000

ipcs	—	—	Report status of the interprocess communication facility
lm	—	—	Start the login monitor for OCS support
locale	POSIX.2	XPG4.2	Get locale-specific information
localedef	POSIX.2	XPG4.2	Define the locale environment
logger	POSIX.2	XPG4.2	Log messages
logname	POSIX.2	XPG4.2	Return a user's login name
newgrp	POSIX.2	XPG4.2	Change to a new group
ocsconfig	—	—	Configure, unconfigure, or query an OCS object
printenv	—	—	Display the value of environment variables
r	—	—	Process a command history list
readonly	POSIX.2	—	Mark a variable as read-only
return	POSIX.2	XPG4.2	Return from a shell function or **.** (dot) script
set	POSIX.2	XPG4.2	Set or unset command options and positional parameters
shift	POSIX.2	XPG4.2	Shift positional parameters
stty	POSIX.2	XPG4.2	Set or display terminal options
su	—	—	Change the user ID connected with a session
sysvar	—	—	Display static system symbols
tic	—	—	Compile term descriptions into terminfo database entries
touch	POSIX.2	XPG4.2	Change the file access and modification times
tput	POSIX.2	XPG4.2	Change characteristics of terminals
tso	—	—	Run a TSO command from the shell
tty	POSIX.2	—	Return the user's terminal name
uconvdef	—	—	Create binary conversion tables
unalias	POSIX.2	XPG4.2	Remove alias definitions
uname	POSIX.2	XPG4.2	Display the name of the current operating system
unset	POSIX.2	XPG4.2	Unset values and attributes of variables and functions
who	POSIX.2	XPG4.2	Display information about current users

Daemons

cron	—	—	Run commands at specified dates and times
inetd	—	—	Handle login requests
rlogind	—	—	Validate rlogin requests
uupd	—	—	Invoke **uucico** for TCP/IP connections from remote UUCP systems

Managing Directories

basename	POSIX.2	XPG4.2	Return the nondirectory components of a pathname
cd	POSIX.2	XPG4.2	Change the working directory
chgrp	POSIX.2	XPG4.2	Change the group owner of a file or directory
chmod	POSIX.2	XPG4.2	Change the mode of a group or directory
chown	POSIX.2	XPG4.2	Change the owner or group of a file or directory
chroot	—	—	Change the root directory for the execution of a command
dircmp	—	XPG4.2	Compare directories
dirname	POSIX.2	XPG4.2	Return the directory components of a pathname
ls	POSIX.2	XPG4.2	List file and directory names and attributes
mkdir	POSIX.2	XPG4.2	Make a directory
mount	—	—	Logically mount a file system
mv	POSIX.2	XPG4.2	Rename or move a file or directory
pathchk	POSIX.2	XPG4.2	Check a pathname
pwd	POSIX.2	XPG4.2	Return the working directory name
rm	POSIX.2	XPG4.2	Remove a directory entry
rmdir	POSIX.2	XPG4.2	Remove a directory

unlink	—	XPG5.0	Removes a directory entry

Managing Files

cat	POSIX.2	XPG4.2	Concatenate or display text files
chaudit	—	—	Change audit flags for a file
cksum	POSIX.2	XPG4.2	Calculate and write checksums and byte counts
cmp	POSIX.2	XPG4.2	Compare two files
col	—	XPG4.2	Remove reverse line feeds
comm	POSIX.2	XPG4.2	Show and select or reject lines common to two files
compress	—	XPG4.2	Lempel-Ziv file compression
cp	POSIX.2	XPG4.2	Copy a file
csplit	POSIX.2	XPG4.2	Split text files
ctags	POSIX.2	XPG4.2	Create tag files for **ex**, **more**, and **vi**
dot or **.**	—	XPG4.2	Run a shell file in the current environment
cut	POSIX.2	XPG4.2	Cut out selected fields of each line of a file
dd	POSIX.2	XPG4.2	Convert and copy a file
df	POSIX.2	XPG4.2	Display the amount of free space in the file system
diff	POSIX.2	XPG4.2	Compare two text files and show the differences
du	POSIX.2	XPG4.2	Summarize usage of file space
ed	POSIX.2	XPG4.2	Use the **ed** line-oriented text editor
egrep	—	XPG4.2	Search a file for a specified pattern
ex	POSIX.2	XPG4.2	Use the **ex** text editor
exrecover	—	—	**vi** file recovery daemon
extattr	—	—	Set, reset, or display extended attributes for files
expand	POSIX.2	XPG4.2	Expand tabs to spaces
fgrep	—	XPG4.2	Search a file for a specified pattern
file	POSIX.2	XPG4.2	Determine file type
filecache	—	—	Manage file caches
find	POSIX.2	XPG4.2	Find a file meeting specified criteria
fold	POSIX.2	XPG4.2	Break lines into shorter lines
head	POSIX.2	XPG4.2	Display the first part of a file
iconv	—	XPG4.2	Convert characters from one code set to another
join	POSIX.2	XPG4.2	Join two sorted, textual relational databases
line	—	XPG4.2	Copy one line of standard input
link	—	XPG5.0	Create a hard link to a file
ln	POSIX.2	XPG4.2	Create a link to a file
mkfifo	POSIX.2	XPG4.2	Make a FIFO special file
mknod	—	—	Make a FIFO or character special file
mount	—	—	Logically mount a file system
more	POSIX.2	XPG4.2	Display files on a page-by-page basis
mv	POSIX.2	XPG4.2	Rename or move a file or directory
nl	—	XPG4.2	Number lines in a file
nm	POSIX.2	XPG4.2	Display symbol table of object, library, and executable files
obrowse	—	—	Browse an HFS file
od	POSIX.2	XPG4.2	Dump a file in a specified format
oedit	—	—	Edit an HFS file
pack	—	XPG4.2	Compress files by Huffman coding
paste	POSIX.2	XPG4.2	Merge corresponding or subsequent lines of a file
patch	POSIX.2	XPG4.2	Change a file using diff output
pcat	—	XPG4.2	Display Huffman-packed lines on standard output
pg	—	XPG4.2	Display files interactively
sed	POSIX.2	XPG4.2	Start the **sed** noninteractive stream editor
sort	POSIX.2	XPG4.2	Start the sort-merge utility
spell	—	XPG4.2	Detect spelling errors in files
split	POSIX.2	XPG4.2	Split a file into manageable pieces

strings	POSIX.2	XPG4.2	Display printable strings in binary files
sum	—	XPG4.2	Compute checksum and block count for file
tabs	POSIX.2	XPG4.2	Set tab stops
tail	POSIX.2	XPG4.2	Display the last part of a file
tee	POSIX.2	XPG4.2	Duplicate the output stream
tr	POSIX.2	XPG4.2	Translate characters
tsort	—	XPG4.2	Sort files topologically
umask	POSIX.2	XPG4.2	Set or return the file mode creation mask
uncom-press	—	XPG4.2	Undo Lempel-Zev compression of a file
unexpand	POSIX.2	XPG4.2	Compress spaces into tabs
uniq	POSIX.2	XPG4.2	Report or filter out repeated lines in a file
unmount	—	—	Remove a file system from the file hierarchy
unpack	—	XPG4.2	Decode Huffman packed files
uudecode	POSIX.2	XPG4.2	Decode a transmitted binary file
uuencode	POSIX.2	XPG4.2	Encode a file for safe transmission
vi	POSIX.2	XPG4.2	Use the display-oriented interactive text editor
wc	POSIX.2	XPG4.2	Count newlines, words, and bytes
zcat	—	XPG4.2	Uncompress and display data

Printing Files

cancel	—	—	Cancel print queue requests (stub command)
infocmp	—	—	Compare and print the terminal description
lp	POSIX.2	XPG4.2	Send a file to a printer
lpstat	—	—	Show status of print queues (stub command)
pr	POSIX.2	XPG4.2	Format a file in paginated form and send it to standard output

Computing and Managing Logic

bc	POSIX.2	XPG4.2	Use the arbitrary-precision arithmetic calculation language
break	POSIX.2	XPG4.2	Exit from a for, while, or until loop in a shell script
colon or **:**	POSIX.2	XPG4.2	Do nothing, successfully
continue	POSIX.2	XPG4.2	Skip to the next iteration of a loop in a shell script
dot or **.**	POSIX.2	XPG4.2	Run a shell file in the current environment
eval	POSIX.2	XPG4.2	Construct a command by concatenating arguments
exec	POSIX.2	XPG4.2	Run a command and open, close, or copy the file descriptors
exit	POSIX.2	XPG4.2	Return to the parent process from which the shell was called or to TSO/E
expr	POSIX.2	XPG4.2	Evaluate arguments as an expression
false	POSIX.2	XPG4.2	Return a nonzero exit code
grep	POSIX.2	XPG4.2	Search a file for a specified pattern
left bracket or **[**	—	XPG4.2	Test for a condition
let	—	—	Evaluate an arithmetic expression
test	POSIX.2	XPG4.2	Test for a condition
trap	POSIX.2	XPG4.2	Intercept abnormal conditions and interrupts
true	POSIX.2	XPG4.2	Return a value of `0`

Controlling Processes

bg	POSIX.2	XPG4.2	Move a job to the background
crontab	POSIX.2	XPG4.2	Schedule regular background jobs
fg	POSIX.2	XPG4.2	Bring a job into the foreground
fuser	—	XPG5.0	List process IDs of processes with open files
jobs	POSIX.2	XPG4.2	Return the status of jobs in the current session
kill	POSIX.2	XPG4.2	End a process or job, or send it a signal
nohup	POSIX.2	XPG4.2	Start a process that is immune to hangups
ps	POSIX.2	XPG4.2	Return the status of a process
renice	POSIX.2	XPG4.2	Change priorities of a running process
sleep	POSIX.2	XPG4.2	Suspend execution of a process for an interval of time
stop	POSIX.2	XPG4.2	Suspend a process or job
suspend	POSIX.2	XPG4.2	Send a SIGSTOP to the current shell
time	POSIX.2	XPG4.2	Display processor and elapsed times for a command
times	—	XPG4.2	Get process and child process times
wait	POSIX.2	XPG4.2	Wait for a child process to end
ulimit	—	XPG4.2	Set process limits

Writing Shell Scripts

autoload	—	—	Indicate function name not defined
dspmsg	—	—	Display selected messages from message catalogs
functions	—	—	Display or assign attributes to functions
getconf	POSIX.2	XPG4.2	Get configuration values
getopts	POSIX.2	XPG4.2	Parse utility options
integer	—	—	Mark each variable with an integer value
read	POSIX.2	XPG4.2	Read a line from standard input
type	—	XPG4.2	Tell how the shell interprets a name
typeset	—	—	Assign attributes and values to variables
xargs	POSIX.2	XPG4.2	Construct an argument list and run a command

Developing or Porting Application Programs

ar	POSIX.2	XPG4.2	Create or maintain library archives
awk	POSIX.2	XPG4.2	Process programs written in the **awk** language
c89	POSIX.2	XPG4.2	Compile OS/390 C source code and create an executable file
c++	—	—	Compile C++/MVS and C/MVS and create an executable file
cc	—	XPG4.2	Compile common usage source code and create an executable file
dbx	—	—	Use the debugger
dspcat	—	—	Display all or part of a message catalog
gencat	—	XPG4.2	Create or edit message catalogs
lex	POSIX.2	XPG4.2	Generate a program for lexical tasks
make	POSIX.2	XPG4.2	Maintain program-generated and interdependent files
mkcatdefs	—	—	Preprocess a message source file
runcat	—	—	Pipe output from mkcatdefs to gencat
strip	POSIX.2	XPG4.2	Remove unnecessary information from an executable file
yacc	POSIX.2	XPG4.2	Use the **yacc** compiler

Communicating with the System or Other Users

mail	—	XPG4.2	Read and send mail messages
mailx	POSIX.2	XPG4.2	Send or receive electronic mail
mesg	POSIX.2	XPG4.2	Allow or refuse messages
talk	POSIX.2	XPG4.2	Talk to another user
write	POSIX.2	XPG4.2	Write to another user

Working with Archives

ar	POSIX.2	XPG4.2	Create or maintain library archives
cpio	—	XPG4.2	Copy in/out file archives
pax	POSIX.2	XPG4.2	Interchange portable archives
tar	—	XPG4.2	Manipulate the **tar** archive files to copy or back up a file

Working with UUCP

uucc	—	—	Compile UUCP configuration files
uucico	—	—	Process UUCP file transfer requests
uucp	—	XPG4.2	Copy files between remote UUCP systems
uucpd	—	—	Invoke **uucico** for TCP/IP connections from remote UUCP systems
uulog	—	XPG4.2	Display log information about UUCP events
uuname	—	XPG4.2	Display list of remote UUCP systems
uupick	—	XPG4.2	Manage files sent by uuto and uucp
uustat	—	XPG42	Display status of pending UUCP transfers
uuto	—	XPG42	Copy files to users on remote UUCP systems
uux	—	XPG42	Request command execution on remote UUCP systems
uuxqt	—	—	Carry out command requests from remote UUCP systems

Appendix B. tcsh Shell Command Summary

The following list presents the built-in tcsh shell commands, grouped by the task a user might want to perform, and their functions. Similar tasks are organized together.

General Use

alloc	—	—	Show the amount of dynamic memory acquired
builtins	—	—	Print the names of all built-in commands
bye	—	—	Terminate the login shell
echo	—	—	Write arguments to standard output
echotc	—	—	Exercise the terminal capabilities in args
exec	—	—	Run a command and open, close, or copy the file descriptors
glob	—	—	Write each word to standard output
hashstat	—	—	Print a statistic line on hash table effectiveness
login	—	—	Terminate a login shell
logout	—	—	Terminate a login shell
nice	—	—	Run a command at a different priority
notify	—	—	Notify user of job status changes
repeat	—	—	Execute command count times
source	—	—	Read and execute commands from name
time	—	—	Display processor and elapsed times for a command
where	—	—	Report all instances of command
which	—	—	Display next executed command

Controlling Your Environment

@ (at)	—	—	Print the value of tcsh shell variables, or assign a value
alias	—	—	Display or create a command alias
bindkey	—	—	List all bound keys, or change key bindings
complete	—	—	List completions
history	—	—	Display a command history list
hup	—	—	Run command so it exits on a hang-up signal
newgrp	—	—	Change to a new group
onintr	—	—	Control the action of the tcsh shell on interrupts
printenv	—	—	Display the values of environment variables
rehash	—	—	Recompute internal hash table
sched	—	—	Print scheduled event list
set	—	—	Set or unset command options and positional parameters
setenv	—	—	Set environment variable name to value
settc	—	—	Tell tcsh shell the terminal capability cap value
setty	—	—	Control tty mode changes
shift	—	—	Shift positional parameters
telltc	—	—	List terminal capability values
unalias	—	—	Remove alias definitions
uncomplete	—	—	Remove completions whose names match pattern
unhash	—	—	Disable use of internal hash table
unlimit	—	—	Remove resource limitations
unset	—	—	Unset values and attributes of variables and functions
unsetenv	—	—	Remove environment variables that match pattern

© Copyright IBM Corp. 1996, 2000

watchlog	—	—	Report on users who are logged in.

Managing Directories

cd	—	—	Change the working directory
chdir	—	—	Change the working directory
dirs	—	—	Print the directory stack
popd	—	—	Pop the directory stack
pushd	—	—	Make exchanges within directory stack

Computing and Managing Logic

break	—	—	Exit from a loop in a shell script
breaksw	—	—	Cause a break from a switch
continue	—	—	Skip to the next iteration of a loop in a shell script
default	—	—	Label default case in a switch statement
eval	—	—	Construct a command by concatenating arguments
exec	—	—	Run a command and open, close, or copy the file descriptors
exit	—	—	Return to the shell's parent process or to TSO/E
filetest	—	—	Apply a file inquiry operator to a file

Managing Files

ls-F	—	—	List files

Controlling Processes

bg	—	—	Move a job to the background
fg	—		Bring a job into the foreground
jobs	—	—	Return the status of jobs in the current session
kill	—	—	End a process or job, or send it a signal
limit	—	—	Limit consumption of processes
nohup	—	—	Start a process that is immune to hangups
stop	—	—	Suspend a process or job
suspend	—	—	Send a SIGSTOP to the current shell
time	—	—	Display processor and elapsed times for a command
wait	—		Wait for a child process to end

Appendix C. Regular Expressions (regexp)

Many OS/390 shell commands match strings of text in text files using a type of pattern known as a *regular expression*. A regular expression lets you find strings in text files not only by direct match, but also by extended matches, similar to, but much more powerful than the filename patterns described in **sh**.

The newline character at the end of each input line is never explicitly matched by any regular expression or part thereof.

expr and **ed** take *basic regular expressions*; all other shell commands accept *extended regular expressions*. **grep** and **sed** accept basic regular expressions, but will accept extended regular expressions if the **–E** option is used.

Regular expressions can be made up of normal characters or special characters, sometimes called *metacharacters*. Basic and extended regular expressions differ only in the metacharacters they can contain.

The basic regular expression metacharacters are:

```
¬ $ . * \( \) [ \{ \} \
```

The extended regular expression metacharacters are:

```
| ¬ $ . * + ? ( ) [ { } \
```

These have the following meanings:

. A dot character matches any single character of the input line.

¬ The ¬ character does not match any character but represents the beginning of the input line. For example, ¬A is a regular expression matching the letter A at the beginning of a line. The ¬ character is only special at the beginning of a regular expression, or after a (or |.

$ This does not match any character but represents the end of the input line. For example, A$ is a regular expression matching the letter *A* at the end of a line. The $ character is only special at the end of a a regular expression, or before a) or |.

[*bracket-expression*]
A bracket expression enclosed in square brackets is a regular expression that matches a single character, or collation element. This bracket expression applies not only to regular expressions, but also to pattern matching as performed by the **fnmatch()** function (used in filename expansion).

- If the initial character is a circumflex (^), then this bracket expression is complemented. It matches any character or collation-element except for the expressions specified in the bracket expression. For pattern matching, as performed by the **fnmatch** function, this initial character is instead ! (the exclamation mark).
- If the first character after any potential circumflex is either a dash (-), or a closing square bracket (]), then that character matches exactly that character—that is, a literal dash or closing square bracket.
- You can specify collation sequences by enclosing their name inside square brackets and periods. For example, **[.ch.]** matches the multicharacter col-

© Copyright IBM Corp. 1996, 2000

lation sequence `ch` (if the current language supports that collation sequence). Any single character is itself. Do not give a collation sequence that is not part of the current locale.

- Equivalence classes can be specified by enclosing a character or collation sequence inside square bracket equals. For example, `[=a=]` matches any character in the same equivalence class as `a`. This normally expands to all the variants of `a` in the current locale—for example, `a`, `\(a:`, `\(a'`, ... On some locales it might include both the uppercase and lowercase of a given character. In the POSIX locale, this always expands to only the character given.
- Within a character class expression (one made with square brackets), the following constructs can be used to represent sets of characters. These constructs are used for internationalization and handle the different collation sequences as required by POSIX.

  `[:alpha:]` Any alphabetic character.

  `[:lower:]` Any lowercase alphabetic character.

  `[:upper:]` Any uppercase alphabetic character.

  `[:digit:]` Any digit character.

  `[:alnum:]` Any alphanumeric character (alphabetic or digit).

  `[:space:]` Any white-space character (blank, horizontal tab, vertical tab).

  `[:graph:]` Any printable character, except the blank character.

  `[:print:]` Any printable character, including the blank character.

  `[:punct:]` Any printable character that is not white space or alphanumeric.

  `[:cntrl:]` Any nonprintable character.

  For example, given the character class expression:

  ```
  [:alpha:]
  ```

  you need to enclose the expression within another set of square brackets, as in:

  ```
  /[[:alpha:]]/
  ```

- Character ranges are specified by a dash (–), between two characters, or collation sequences. These indicates all character or collation sequences that collate between two characters or collation sequences. It does not refer to the native character set. For example, in the POSIX locale, `[a-z]` means all the lowercase alphabetics, even if they don't agree with the binary machine ordering. However, since many other locales do not collate in this manner, use of ranges are not recommended, and are not used in strictly conforming POSIX.2 applications. An endpoint of a range may explicitly be a collation sequence; for example, `[[.ch.]-[.ll.]]` is valid. However, equivalence classes or character classes are not: `[[=a=]-z]` is not permitted.

\ This character turns off the special meaning of metacharacters. For example, `\.` only matches a dot character. Note that `\\` matches a literal \ character. Also note the special case of "`\`*d*" described in the following paragraph.

\d For *d* representing any single decimal digit (from 1 to 9), this pattern is equivalent to the string matching the *d*th expression enclosed within the () characters (or \(\) for some commands) found at an *earlier point* in the regular expression. Parenthesized expressions are numbered by counting (characters from the left.

Constructs of this form can be used in the replacement strings of substitution commands (for example, the `sub` function of **awk**), to stand for constructs matched by parts of the regular expression.

*regexp**
A regular expression *regexp* followed by * matches a string of *zero* or more strings that would match *regexp*. For example, `A*` matches `A`, `AA`, `AAA` and so on. It also matches the null string (zero occurrences of `A`).).

regexp+
A regular expression *regexp* followed by + matches a string of *one* or more strings that would match *regexp*.

regexp?
A regular expression *regexp* followed by ? matches a string of one or zero occurrences of strings that would match *regexp*.

char{n} | *char\{n\}*
In this expression (and the ones to follow), *char* is a regular expression that stands for a single character—for example, a literal character or a period (.). Such a regular expression followed by a number in brace brackets stands for that number of repetitions of a character. For example, `X\{3\}` stands for `XXX`. In basic regular expressions, in order to reduce the number of special characters, { and } must be escaped by the \ character to make them special, as shown in the second form (and the ones to follow).

char{min,} | *char\{min,\}*
When a number, *min*, followed by a comma appears in braces following a single-character regular expression, it stands for at least *min* repetitions of a character. For example, `X\{3,\}` stands for at least three repetitions of `X`.

char{min,max} | *char\{min,max\}*
When a single-character regular expression is followed by a pair of numbers in braces, it stands for at least *min* repetitions and no more than *max* repetitions of a character. For example, `X\{3,7\}` stands for three to seven repetitions of `X`.

regexp1 | *regexp2*
This expression matches either regular expression *regexp1* or *regexp2*.

(regexp) | \(*regexp*\)
This lets you group parts of regular expressions. Except where overridden by parentheses, concatenation has the highest precedence. In basic regular expressions, in order to reduce the number of special characters, (and) must be escaped by the \ character to make them special, as shown in the second form.

Several regular expressions can be concatenated to form a larger regular expression.

Summary

The commands that use basic and extended regular expressions are as follows:

Basic **ed**, **expr**, **grep**, **sed**
Extended **awk**, **grep** with **-E** option, **sed** with the **-E** option.

Figure 19 summarizes the features that apply to which shell commands:

Figure 19. Regular Expression Features (regexp)

Notation	awk	ed	grep -E	expr	sed
.	Yes	Yes	Yes	Yes	Yes
	Yes	Yes	Yes	No	Yes
$	Yes	Yes	Yes	Yes	Yes
[...]	Yes	Yes	Yes	Yes	Yes
[::]	Yes	Yes	Yes	Yes	Yes
*re**	Yes	Yes	Yes	Yes	Yes
re+	Yes	No	Yes	No	No
re?	Yes	No	Yes	No	No
re|*re*	Yes	No	Yes	No	No
d	Yes	Yes	Yes	Yes	Yes
(...)	Yes	No	Yes	No	No
\(...\)	No	Yes	No	Yes	Yes
\<	No	No	No	No	No
\>	No	No	No	No	No
\{ \}	Yes	No	Yes	No	Yes

Examples

The following patterns are given as illustrations, along with descriptions of what they match:

`abc` Matches any line of text containing the three letters `abc` in that order.

`a.c` Matches any string beginning with the letter `a`, followed by *any* character, followed by the letter `c`.

`^.$` Matches any line containing exactly one character (the newline is not counted).

`a(b*|c*)d` Matches any string beginning with a letter `a`, followed by either zero or more of the letter `b`, or zero or more of the letter `c`, followed by the letter `d`.

`.* [a-z]+ .*`
Matches any line containing a *word*, consisting of lowercase alphabetic characters, delimited by at least one space on each side.

`(morty).*\1`

`morty.*morty`
: These expressions both match lines containing at least two occurrences of the string `morty`.

`[[:space:][:alnum:]]`
: Matches any character that is either a white-space character or alphanumeric.

Appendix D. Running Shell Scripts or Executable Files under MVS Environments

This appendix describes the IBM-supplied BPXBATCH program.

BPXBATCH

BPXBATCH makes it easy for you to run shell scripts and executable files that reside in hierarchical file system (HFS) files through the MVS job control language (JCL). If you do most of your work from TSO/E, using BPXBATCH saves you the trouble of going into the shell to run your scripts and executable files. REXX execs can also use BPXBATCH to run shell scripts and executable files.

In addition to using BPXBATCH, a user who wants to perform a local spawn without being concerned about environment set-up (that is, without having to set specific environment variables which could be overwritten if they are also set in the user's profile) can use BPXBATSL. BPXBATSL provides users with an alternate entry point into BPXBATCH, and forces a program to run using a local spawn instead of fork/exec as BPXBATCH does. This ultimately allows a program to run faster.

BPXBATSL is also useful when the user wants to perform a local spawn of their program, but also needs subsequent child processes to be fork/exec'ed. Formerly, with BPXBATCH, this could not be done since BPXBATCH and the requested program shared the same environment variables.

BPXBATSL is an alias of BPXBATCH.

Format

JCL:
EXEC PGM=BPXBATCH,PARM='SH|PGM program_name'

TSO/E:
BPXBATCH SH|PGM program_name

Description

The BPXBATCH program allows you to submit MVS batch jobs that run shell commands or scripts, or OS/390 C executable files in hierarchical file system (HFS) files from a shell session. You can invoke BPXBATCH from a JCL job or from TSO/E (as a command, through a CALL command, or from a CLIST or REXX EXEC).

With BPXBATCH, you can allocate the MVS standard files **stdin**, **stdout**, and **stderr** as HFS files for passing input. If you do allocate the foregoing standard files, they must be HFS files. You can also allocate MVS data sets or HFS text files containing environment variables (**stdenv**). If you do not allocate them, **stdin**, **stdout**, **stderr**, and **stdenv** default to **/dev/null**. Allocate the standard files using the data definition PATH keyword options, or standard data definition options for MVS data sets, for **stdenv**.

For JCL jobs, you specify PATH keyword options on DD statements:

© Copyright IBM Corp. 1996, 2000

```
//jobname JOB ...

//stepname EXEC PGM=BPXBATCH,PARM='SH|PGM program_name'

//STDIN   DD  PATH='/stdin-file-pathname',PATHOPTS=(ORDONLY)
//STDOUT  DD  PATH='/stdout-file-pathname',PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//            PATHMODE=SIRWXU
//STDERR  DD  PATH='/stderr-file-pathname',PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//            PATHMODE=SIRWXU
⋮
```

Your application in the executable file can also allocate **stdin**, **stdout**, **stderr**, and **stdenv** dynamically through the use of SVC 99.

For TSO/E, you specify PATH keyword options on the ALLOCATE command:

```
ALLOCATE FILE(STDIN) PATH('/stdin-file-pathname') PATHOPTS(ORDONLY)
ALLOCATE FILE(STDOUT) PATH('/stdout-file-pathname')
         PATHOPTS(OWRONLY,OCREAT,OTRUNC) PATHMODE(SIRWXU)
ALLOCATE FILE(STDERR) PATH('/stderr-file-pathname')
         PATHOPTS(OWRONLY,OCREAT,OTRUNC) PATHMODE(SIRWXU)

BPXBATCH SH|PGM program_name
```

stdin and **stdenv** must always be allocated as read. **stdout** and **stderr** must always be allocated as write.

Parameters

BPXBATCH accepts one parameter string as input, the combination of SH|PGM and program_name. At least one blank character must separate the parts of the parameter string. The total length of the parameter string must *not* exceed 500 characters.

Note: BPXBATCH was first created for use with JCL, which had a max parm length of 100. Now, BPXBATCH can also be called form CLIST, REXX, and TSO. These additional environments do not have the 100 character parameter limit, hence the increase to 500.

If neither SH nor PGM is specified as part of the parameter string, BPXBATCH assumes that the shell is to be started to run the shell script allocated by STDIN.

SH|PGM
Specifies whether BPXBATCH is to run a shell script or command or a OS/390 C executable file located in an HFS file.

SH
Specifies that the shell designated in your TSO/E user ID's security product profile is to be started and is to run shell commands or scripts provided from **stdin** or the specified program_name.

Note: If SH is specified with no program_name information, BPXBATCH attempts to run anything read in from STDIN.

PGM
Specifies that the specified program_name is to be run as a called program from a shell environment.

If PGM is specified, program_name must also be specified. BPXBATCH creates a process for the program to run in and then calls the program. All environment variables read from the **stdenv** file are set when the program is run, if **stdenv** was allocated. If possible, the **HOME** and **LOGNAME** environment variables are also set when the program is run, if they were not specified in the **stdenv** file, or if **stdenv** was not allocated. Refer to "Usage Notes" for more information on environment variable processing.

program_name
Specifies the shell command name or the HFS pathname for the shell script or OS/390 C executable file to be run. In addition, program_name can contain option information.

The program_name is interpreted as case-sensitive.

Note: When PGM and program_name are specified and the specified program name does not begin with a slash character (/), BPXBATCH prefixes the user's *initial* working directory information to the program pathname.

Usage Notes

1. BPXBATCH is an alias for the program BPXMBATC, which resides in the SYS1.LINKLIB data set.
2. BPXBATCH must be invoked from a user address space running with a program status word (PSW) key of 8.
3. BPXBATCH does not translate characters on the supplied parameter information. You should supply parameter information, including HFS pathnames, using only the POSIX portable character set. For information on the POSIX portable character set, see *OS/390 UNIX System Services Programming Tools*.
4. If your BPXABATCH job returns ABEND 4093 reason code 0000001c, you need to expand the region size. For example:

   ```
   //SHELLCMD  EXEC PGM=BPXBATCH,REGION=8M,PARM='SH shell_cmd'
   ```

5. Allocations for any files other than **stdin**, **stdout**, **stderr**, or **stdenv** cannot be used by a program run by BPXBATCH.
6. BPXBATCH does not close file descriptors other than `0–2`. Other file descriptors that are open and not defined as "marked to be closed" remain open when you call BPXBATCH and BPXBATCH runs the specified script or executable file.
7. BPXBATCH uses write-to-operator (WTO) routing code 11 to write error messages to either the JCL job log or your TSO/E terminal. Your TSO/E user profile must specify WTPMSG so that messages can be displayed at the terminal.
8. BPXBATCH (with the SH parameter) must not be used to run an executable file, shell command, or shell script in the background (by specifying the shell & symbol) unless the shell **nohup** command is also used. If the shell ampersand (&) symbol is used without **nohup**, the results are unpredictable.
9. BPXBATCH, when used with the PGM parameter, sets up environment variables for the program to be run. If the **stdenv** file is not allocated, the **HOME** and **LOGNAME** environment variables are set. If **stdenv** is allocated, the environment variables read from the file it represents are set, with **HOME** or **LOGNAME** or both environment variables added if they are not specified in the **stdenv** file. The following types of files can be allocated to **stdenv**:

- HFS text file
- Sequential format MVS data set (including SYSIN data set)
- Member of a partitioned data set (PDS)

Other forms of MVS data sets, such as DUMMY, TERMINAL, SYSOUT, or PDS/E, are not supported for **stdenv**.

The **stdenv** file consists of one or more records, where *record* is defined as a string terminated with a <newline> character (X'15') in an HFS file, or a fixed or variable (nonspanned) format record in an MVS data set. Other MVS record formats are not supported for **stdenv**. The following rules apply to the specification of environment variables in **stdenv** files:

- Only one environment variable can be specified per record.
- Each environment variable is specified as *variable*=*value*.
- Environment variable names must begin in column 1, unless names beginning with blanks are used.
- Environment variable records should not be terminated with null characters (X'00'). BPXBATCH automatically appends a null character to the end of each environment variable, and the lengths of environment variables as seen by the program include the null characters.
- Trailing blanks (X'40') are truncated for MVS SYSIN data sets, but are not truncated for any other type of file.
- Be careful that sequence numbers are not present in MVS data sets, because they will be treated as part of the environment variables. ISPF edit users should always set number mode off when creating environment variables, including JCL data sets with environment variables specified as SYSIN.

Some environment variables are release-dependent. If BPXBATCH is executed on a system that does not support the environment variable, you will not get an error message and the variable will be ignored. Use the **uname** shell command to determine the release number of the operating system that BPXBATCH is running on.

Environment variables (including **PATH**) are established at the start of the executable program, not for BPXBATCH itself. Thus, **PATH** is not searched to locate the program, but instead is used if the program invokes other executable programs. In the following example, **someprogram** may be found only in the initial working directory defined by the user's profile, not by the **PATH** environment variable:

```
//jobname JOB ...

//stepname EXEC PGM=BPXBATCH,PARM='PGM someprogram parm1 parm2'

//STDOUT  DD  PATH='/tmp/pgmout',PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//            PATHMODE=SIRWXU
//STDENV  DD  *
PATH=/bin:/u/usr/joeuser
STEPLIB=SYS1.JOE.STEPLIB
/*
```

10. BPXBATCH uses two more environment variables for execution that are specified by STDENV:
    - **_BPX_BATCH_UMASK=0755**

- **_BPX_BATCH_SPAWN=YES|NO**

_BPX_BATCH_UMASK allows the user the flexibility of modifying the permission bits on newly created files instead of using the default mask (when PGM is specified).

Note: This variable will be overriden by umask (usually set from within /etc/profile) if BPXBATCH is invoked with the 'SH' option (SH is the default). SH causes BPXBATCH to execute a login shell which runs the /etc/profile script (and runs the user's .profile) and which may set the umask before execution of the intended program.

_BPX_BATCH_SPAWN causes BPXBATCH to use SPAWN instead of fork/exec and allows data definitions to be carried over into the spawned process. When **_BPX_BATCH_SPAWN** is set to YES, spawn will be used. If it is set to NO, which is equivalent to the default behavior, fork/exec will be used to execute the program.

If **_BPX_BATCH_SPAWN** is set to YES, then you must consider two other environment variables that affect spawn (BPX1SPN):

- **_BPX_SHAREAS=YES|NO|REUSE**

  When YES or REUSE, the child process created by spawn will run in the same address space as the parent's under these conditions:

  – The child process is not setuid or setgid to a value different from the parent

  – The spawned filename is not an external link or a sticky bit file

  – The parent has enough resources to allow the child process to reside in the same address space

  – The NOSHAREAS extended attribute is not set

  When no, the child and parent run in separate address spaces.

- **_BPX_SPAWN_SCRIPT=YES**

  Beginning in Release 8 of OS/390, spawn will recognize a header in the first line of a HFS file that indicates the file to be executed and its first set of arguments. This header will only be recognized when a HFS file is not found in an executable format. The format of the header is as follows:

  ```
  #! Path String
  ```

  where **#!** is the file magic number. The magic number indicates that the first line of a file is a special header that contains the name of the program to be executed and any argument data to be supplied to it.

  When **_BPX_SPAWN_SCRIPT=yes**, spawn will first recognize the file magic number and will process the file accordingly. If the file magic number is not found in the file's first line, spawn will treat the specified file as a shell script and will invoke the shell to run the shell script.

For more information about spawn, see BPX1SPN in *OS/390 UNIX System Services Programming: Assembler Callable Services Reference*.

Files

- SYS1.LINKLIB(BPXMBATC) is the BPXBATCH program location.
- The **stdin** default is **/dev/null**.
- The **stdout** default is **/dev/null**.
- The **stderr** default is the value of **stdout**. If all defaults are accepted, **stderr** is **/dev/null**.
- **stdenv** default is **/dev/null**.

Return Codes

`0` Processing successful.

`254`
Processing unsuccessful. BPXBATCH requires OMVS to be started.

`255`
Processing unsuccessful. An error message has been issued.

`32000`
BPXBATCH invoked the BPX1FRK (fork) callable service. This is usually invoked only by a TSO/E user. One of the following conditions may have resulted:

- BPXBATCH failed to open specified files after the program fork. Files are normally opened and closed prior to a fork. Try running BPXBATCH again.
- The program_name or the shell exited with an exit status of 125.

32512
One of the following conditions may have resulted:

- The PGM keyword was specified for BPXBATCH and no program_name could be found.

  Message BPXM008I was written to the job log or **stderr**.
- The SH keyword was specified for BPXBATCH and either **/bin/login** or the shell did not exist.
- The SH keyword was specified with a program_name value for BPXBATCH and no program_name could be found. The shell exited with an exit status of 127. **stdout** contains a shell message indicating the program was not found.
- The program_name or the shell exited with an exit status of 127.

BPXYWAST
BPXBATCH invoked the BPX1FRK (fork) callable service. This is usually invoked only by a TSO/E user. Processing was successful with wait() status containing a nonzero value. The wait status was mapped by BPXYWAST and returned by BPX1WAT (wait). If the value returned is greater than `255`, the exit status can be determined by dividing the value of BPXYWAST by `256`.

No error messages were issued by BPXBATCH.

Using OSHELL to Run Shell Commands and Scripts from MVS

You can use the OSHELL REXX exec to run a shell command or shell script from the TSO/E `READY` prompt and display the output to your terminal. This exec uses BPXBATCH to run the shell command or shell script:

```
oshell shell_command
```

For example, to display process information, enter:

```
oshell ps –ej
```

Note: With this exec, do not use an & to run a shell command in the background.

Appendix E. BPXCOPY: Copying a Sequential or Partitioned Data Set or PDSE Member into an HFS File

This appendix describes the BPXCOPY program.

BPXCOPY

BPXCOPY allows you to copy a sequential data set or a partitioned data set or a PDSE member into a hierarchical file system (HFS) file.

Format

```
JCL:
EXEC PGM=BPXCOPY,PARM='ELEMENT HEADID LINK TYPE PATHMODE SYMLINK
SYMPATH APF | NOAPF PROGCTL | NOPROGCTL SHAREAS | NOSHAREAS UID GID'
SHARELIB | NOSHARELIB
```

Description

BPXCOPY can copy a sequential data set or partitioned data set or PDSE member into an HFS file. You can invoke BPXCOPY in several ways:

- From JCL using `EXEC PGM=BPXCOPY`. BPXCOPY does not need the Terminal Monitor Program (TMP) to be started when it is invoked from JCL.
- From LINK, XCTL, ATTACH, a TSO/E CALL command with the asis option, or by a CALL after a LOAD.

BPXCOPY provides similar function to the OPUT command, but differs from OPUT in these ways:

- There is no code page conversion available.
- The specified filename cannot be longer than 8 characters.
- The pathname of the directory specified cannot be longer than 255 characters.
- You can define hard links to the file.
- You can define symbolic links to the file.
- You can set the permission access bits of the file.
- You can set the extended attributes of the file.
- You can set the owning UID and GID of the file.

A DD statement allocates a data set or file and sets up a ddname. For BPXCOPY:

- The input ddname is associated with an MVS data set, either a sequential data set or a member of a partitioned data set or PDSE. When you invoke BPXCOPY from JCL, you must use SYSUT1 as the input ddname. If BPXCOPY is invoked from LINK, XCTL, or ATTACH, a TSO/E CALL command with the asis option, or by a CALL after a LOAD, you can specify an alternative ddname.
- The output ddname is associated with the pathname of the directory in which the HFS file resides. The absolute pathname for the HFS file is this pathname combined with the name specified with the ELEMENT parameter. When you invoke BPXCOPY from JCL, you must use SYSUT2 as the output ddname. If BPXCOPY is invoked from LINK, XCTL, or ATTACH, a TSO/E CALL command with the asis option, or by a CALL after a LOAD, you can specify an alternative ddname.

© Copyright IBM Corp. 1996, 2000

- The message output ddname is associated with an MVS data set. The default ddname is SYSTSPRT, which typically directs messages to SYSOUT. When you invoke BPXCOPY from JCL, you must use SYSTSPRT as the message output ddname. If BPXCOPY is invoked from LINK, XCTL, or ATTACH, a TSO/E CALL command with the asis option, or by a CALL after a LOAD, you can specify an alternative ddname.

Parameters

You can specify the following keyword parameters with BPXCOPY. The parameters can be separated by any delimiter (space, comma, tab, or comment (/*)).

ELEMENT(element_name)
element_name is a simple 1-to-8-character filename of the output file. The element_name specified is converted to uppercase characters.

The directory pathname for the output file is specified with the PATH keyword on a JCL DD statement.

The pathname of the output file consists the directory pathname appended with the element_name.

This parameter is required.

HEADID('character_string')
An 8-byte character string, enclosed in single quotes, that will appear on the header of each page of output created.

This optional parameter is provided for SMP/E usage, not for a typical user.

LINK('linkname','linkname',...)
The names of hard links to the file. Each linkname is concatenated with the directory pathname. On the JCL DD statement for the directory, the maximum length for a pathname (before concatenation) is 255 characters. Pathnames with a length of up to 1023 characters can be specified only if BPXCOPY is invoked from LINK, XCTL, or ATTACH, a TSO/E CALL command, or by a CALL after a LOAD.

If you specify this parameter, you create one or more hard links to the file when the data is copied into a file. The linkname must be enclosed in single quotes. You can specify up to 64 linknames, and each must be enclosed in single quotes. Specifying LINK is optional.

SYMLINK('linkname','linkname',...)
The names of symbolic links to the file. Each linkname is concatenated with the directory pathname. On the JCL DD statement for the directory, the maximum length for a pathname (before concatenation) is 255 characters. Pathnames with a length of up to 1023 characters (after concatenation) can be specified if BPXCOPY is involved from LINK, XCTL, or ATTACH, a TSO/E CALL command, or by a CALL after a LOAD.

If you specify this parameter, you create one or more symbolic links to the file. The linkname must be enclosed in single quotes. You can specify up to 64 linknames, and each must be enclosed in single quotes. Specifying SYMLINK is optional. If you specify SYMLINK, you must also specify SYMPATH.

SYMPATH('pathname','pathname',...)
The pathnames of the file for which the symbolic link is created. Each pathname may be an absolute pathname (beginning with a slash) or a relative

pathname (not beginning with a slash). When an absolute pathname is used, the symbolic link will be resolved starting at the root directory. When a relative pathname is used, the symbolic link will be resolved starting at the parent directory of the symbolic link.

For JCL, the maximum length for a pathname is limited by the 100 character limit on the entire PARM string (including other parameters) on the EXEC statement. Pathnames with a length of up to 1023 characters can be specified if BPXCOPY is invoked from LINK, XCTL, or ATTACH, a TSO/E CALL command, or by a CALL after a LOAD.

Specifying SYMPATH is optional, but if you specify SYMPATH, you must also specify SYMLINK. Each SYMLINK linkname must be matched with a corresponding SYMPATH pathname. The first linkname will define a symbolic link to the first pathname, the second linkname will define a symbolic link to the second pathname, etc. If there are fewer pathnames than linknames, the last pathname will be used for the remaining linknames.

PATHMODE (mode_bits)

Changes the access permissions, or *modes,* of the specified file or directory. Modes determine who can read, write, or search a directory. The bits are used to set execution and permission access of the output file. On BPXCOPY, you can specify PATHMODE as an absolute mode; it must consist of four octal numbers separated by commas or blanks.

Absolute modes are four octal numbers specifying the complete list of attributes for the files; you specify attributes by ORing together the bits for each octal number.

```
4,0,0,0   Set-user-ID bit
2,0,0,0   Set-group-ID bit
1,0,0,0   Sticky bit
0,4,0,0   Individual read
0,2,0,0   Individual write
0,1,0,0   Individual execute (or list directory)
0,0,4,0   Group read
0,0,2,0   Group write
0,0,1,0   Group execute
0,0,0,4   Other read
0,0,0,2   Other write
0,0,0,1   Other execute
```

Specifying PATHMODE is optional.

For more information on permission bits, see the **chmod** command.

TYPE (TEXT|BINARY)

The format for the HFS file. The default is BINARY for U-format data sets and TEXT for all others. (U-format means undefined-length records.) Specifying TYPE is optional.

APF|NOAPF

Specifies whether the APF extended attribute is set or unset. When this attribute is set (APF) on an executable program file (load module), it behaves as if loaded from an APF-authorized library. For example, if this program is exec()ed at the job step level and the program is linked with the AC = 1 attribute, the program will be executed as APF-authorized.

To be able to set APF, you must have at least READ access to the BPX.FILEATTR.APF FACILITY class profile. For more information, see *OS/390 UNIX System Services Planning*.

Specifying APF or NOAPF is optional. If not specified, the attribute will be defined as NOAPF.

PROGCTL|NOPROGCTL
Specifies whether the PROGCTL extended attribute is set or unset. When this is set (PROGCTL) on an executable program file (load module), it causes the program to behave as if an RDEFINE had been done for the load module to the PROGRAM class. When this program is brought into storage, it does not cause the enviroment to be marked dirty.

To be able to set PROGCTL, you must have at least READ access to the BPX.FILEATTR.PROGCTL FACILITY class . For more information, see *OS/390 UNIX System Services Planning*.

Specifying PROGCTL or NOPROGCTL is optional. If not specified, the attribute will be defined as NOPROGCTL.

SHAREAS | NOSHAREAS
Specifies whether the SHAREAS extended attribute is set or unset. When this attribute is set (SHAREAS) on an executable program file (load module), the _BPX_SHAREAS environment variable is honored when the file is spawn()ed. When this attribute is not set (NOSHAREAS), the _BPX_SHAREAS environment variable is ignored when the file is spawn()ed.

Specifying SHAREAS or NOSHAREAS is optional. If not specified, the attribute will be defined as SHAREAS.

SHARELIB | NOSHARELIB
Specifies whether the st_ShareLib extended attribute is set or unset in the target file.

Note: In order to use BPXCOPY with this keyword parameter, you must have at least READ access to the BPX.FILEATTR.SHARELIB FACILITY class.

UID(owner)
Specifies the owner of the file. Owner can be a user name or a numeric user ID (UID). However, if a numeric owner exists as a user name in the user data base, the UID number associated with that user name is used.

If a mixed case user name is specified, it must be enclosed in single quotes.

To be able to set the UID of the file, the user must have UID 0 or have at least READ access to the BPX.SUPERUSER FACILITY class.

Specifying UID is optional. If not specified, the UID of the user running BPXCOPY will be used.

GID(group)
Specifies the group owner of the file. group can be a group name or a numeric group ID (GID). However, if a numeric group exists as a group name in the group data base, the GID number associated with that group name is used.

If a mixed case group name is specified, it must be enclosed in single quotes.

To be able to set the GID of the file, the user must have UID 0 or have at least READ access to the BPX.SUPERUSER FACILITY class.

Specifying GID is optional. If not specified, the GID of the directory pathname will be used.

Return Codes

0 Processing successful

12 Processing unsuccessful. An error message has been issued.

Examples

1. JCL and BPXCOPY are used to copy a PDSE member into a directory.

   Known:

   - The name of the PDSE member is REGEREX.
   - The directory name is **/u/turbo/llib**.
   - Output messages are to be directed to SYSOUT.
   - Type of data: binary.

```
//TEST JOB MSGLEVEL=(1,1)
//STEP EXEC PGM=BPXCOPY,
//  PARM='ELEMENT(REGEREX) LINK('../erex'') TYPE(BINARY)'
//SYSUT1   DD DSN=TURBO.LOADLIB(REGEREX),DISP=SHR
//SYSUT2   DD PATH='/u/turbo/llib'
//SYSTSPRT DD SYSOUT=*
```

   The LINK name is concatenated with the directory name from SYSUT2, yielding **/u/turbo/llib/../erex**. The file system treats this as **/u/turbo/erex**, making this an alias for **/u/turbo/llib/REGEREX**.

2. JCL and BPXCOPY are used to copy a PDS member into a directory.

   Known:

   - The name of the PDS member is TABLE1.
   - The directory name is **/u/carbon/data**.
   - Output messages are to be directed to SYSOUT.
   - Type of data: text.

```
//TEST JOB MSGLEVEL=(1,1)
//STEP EXEC PGM=BPXCOPY,
//  PARM='ELEMENT(TABLE1) TYPE(TEXT) PATHMODE(0,7,6,4)'
//SYSUT1   DD DSN=CARBON.DATA(TABLE1),DISP=SHR
//SYSUT2   DD PATH='/u/carbon/data'
//SYSTSPRT DD SYSOUT=*
```

   This results in a file **/u/carbon/data/TABLE1** with read, write, and execute authority for the user; read and write authority for the group; and read authority for other users.

3. A member of an MVS partitioned data set is copied to an HFS file from a program using the LINK macro.

   Known:

   - The ddname of the source: INDD. INDD can be any sequential data set and is defined by an ALLOCATE command issued outside the program.
   - The ddname of the directory to copy into: OUTDD. OUTDD can be any directory name and is defined by an ALLOCATE command issued outside the program.
   - Three link names—**DATA**, **link1**, and **link2**—for the target file.

- Output messages are directed to SYSOUT.
- Type of data: text.

```
*
COPYEX   CSECT
         STM   14,12,12(13)            Entry linkage
         LR    12,15
         USING COPYEX,12
         LA    10,SAVEAREA
         ST    10,8(13)
         ST    13,SAVEAREA+4
         LR    13,10
*
         LINK  EP=BPXCOPY,PARAM=(OPT_LIST,DD_LIST),VL
*
         L     13,SAVEAREA+4           Exit linkage
         L     14,12(13)
         LM    0,12,20(13)
         BR    14
*
SAVEAREA DS    18F
*
OPT_LIST DC    H'80'                   Length of option string
         DC    CL80'ELEMENT(DATA) HEAD('0001'') TYPE(TEXT)            X
               LINK('link1'', ''link2'')'
*
DD_LIST  DC    H'72'                   Length of DDNAME list
         DC    CL56' '
         DC    CL8'INDD'               Logical SYSUT1  input
         DC    CL8'OUTDD'              Logical SYSUT2  output directory
*
         END   COPYEX
```

Appendix F. Localization

Internationalization enables you to work in a cultural context that is comfortable for you through locales, character sets, and a number of special environment variables. The process of adapting an internationalized application or program, particular to a language or cultural milieu, is termed *localization*.

A *locale* is the subset of your environment that deals with language and cultural conventions. It is made up of a number of categories, each of which is associated with an environment variable and controls a specific aspect of the environment. The following list shows the categories and their spheres of influence:

LC_COLLATE
Collating (sorting) order.

LC_CTYPE
Character classification and case conversion.

LC_MESSAGES
Formats of informative and diagnostic messages and interactive responses.

LC_MONETARY
Monetary formatting.

LC_NUMERIC
Numeric, nonmonetary formatting.

LC_TIME
Date and time formats.

LC_SYNTAX
EBCDIC-variant character encodings used by some C functions and utilities.

To give a locale control over a category, set the corresponding variable to the name of the locale. In addition to the environment variables associated with the categories, there are two other variables which are used in conjunction with localization, **LANG** and **LC_ALL**. All of these variables affect the performance of the shell commands. The general effects apply to most commands, but certain commands such as **sort**, with its dependence on **LC_COLLATE**, require special attention to be paid to one or more of the variables; this manual discusses such cases in the *Localization* section of the command. The effects of each environment variable is as follows:

LANG
Determines the international language value. Utilities and applications can use the information from the given locale to provide error messages and instructions in that locale's language. If **LC_ALL** variable is not defined, any undefined variable is treated as though it contained the value of **LANG**.

LC_ALL
Overrides the value of **LANG** and the values of any of the other variables starting with **LC_**.

LC_COLLATE
Identifies the locale that controls the collating (sorting) order of characters and determines the behavior of ranges, equivalence classes, and multicharacter collating elements.

© Copyright IBM Corp. 1996, 2000

LC_CTYPE
Identifies the locale that defines character classes (for example, *alpha, digit, blank*) and their behavior (for example, the mapping of lowercase letters to uppercase letters). This locale also determines the interpretation of sequences of bytes as characters (such as singlebyte versus doublebyte characters).

LC_MESSAGES
Identifies the locale that controls the processing of affirmative and negative responses. This locale also defines the language and cultural conventions used when writing messages.

LC_MONETARY
Determines the locale that controls monetary-related numeric formatting (for example, currency symbol, decimal point character, and thousands separator).

LC_NUMERIC
Determines the locale that controls numeric formatting (for example, decimal point character and thousands separator).

LC_TIME
Identifies the locale that determines the format of time and date strings.

LC_SYNTAX
Identifies the locale that defines the encodings for the variant characters in the portable character set.

The **NLSPATH** localization variable specifies where the message catalogs are to be found.

For example,

```
NLSPATH="/system/nlslib/%N.cat"
```

specifies that the OS/390 shell is to look for all message catalogs in the directory **/system/nlslib**, where the catalog name is to be constructed from the *name* parameter passed to the OS/390 shell with the suffix **.cat**.

Substitution fields consist of a % symbol, followed by a single-letter keyword. These keywords are currently defined:

%N The value of the *name* parameter
%L The value of the **LC_MESSAGES** category, or **LANG**, depending on how the catopen() function that opens this catalog is coded. For more information, refer to catopen() in *OS/390 C/C++ Run-Time Library Reference*.
%l The *language* element from the **LC_MESSAGES** category
%t The *territory* element from the **LC_MESSAGES** category
%c The *codeset* element from the **LC_MESSAGES** category

Templates defined in **NLSPATH** are separated by colons (:). A leading colon or two adjacent colons (::) are equivalent to specifying **%N**. For example:

```
NLSPATH=":%N.cat:/nlslib/%L/%N.cat"
```

specifies that the OS/390 shell should look for the requested message catalog in *name*, *name***.cat**, and **/nlslib**/*category*/*name***.cat**, where *category* is the value of the **LC_MESSAGES** or **LANG** category of the current locale.

Do not set the **NLSPATH** variable unless you need to override the default system path. Otherwise the commands may behave unpredictably.

Appendix G. Stub Commands

OS/390 UNIX has some stub commands. Stub commands are those commands that are recognized by OS/390 UNIX but whose functions are not supported.

They are:

- **cancel**
- **cu**
- **lpstat**

© Copyright IBM Corp. 1996, 2000

Appendix H. File Formats

This appendix gives more detailed information on the formats of the files used by the shell commands. The file formats are:

- **cpio**
- **magic**
- **pax**
- **queudefs**
- **tags**
- **tar**
- **utmpx**
- **uucp**

cpio — Format of cpio archives

You can use the **cpio** command to back up or restore files. The **cpio** command reads and writes either a compact binary format header or an ASCII format header. The **tar** command reads and writes headers in either the original TAR format from UNIX systems or the USTAR format defined by the POSIX 1003.1 standard.

The **pax** command reads and writes headers in any of the **cpio** formats.

Description

A **cpio** archive consists of one or more concatenated member files. Each member file contains a header optionally followed by file contents as indicated in the header. The end of the archive is indicated by another header describing an (empty) file named **TRAILER!!**.

There are two types of **cpio** archives, differing only in the style of the header:

- ASCII archives have totally printable header information; thus, if the files being archived are also ASCII files, the whole archive is ASCII.
- By default, **cpio** writes archives with binary headers. However, binary archive files cannot usually be ported to other operating systems, so you should not use these.

The information in an ASCII archive header is stored in fixed-width, octal (base 8) numbers padded with zeros on the left. Figure 20 gives the order and field width for the information in the ASCII header:

© Copyright IBM Corp. 1996, 2000

Figure 20. Archive File: ASCII Header

Field Width	Field Name	Meaning
6	magic	Magic number 070707
6	dev	Device where file resides
6	ino	I-number of file
6	mode	File mode
6	uid	Owner user ID
6	gid	Owner group ID
6	nlink	Number of links to file
6	rdev	Device major/minor for special file
11	mtime	Modify time of file
6	namesize	Length of filename
11	filesize	Length of file

After the header information, *namesize* bytes of pathname are stored. *namesize* includes the null byte of the end of the pathname. After this, *filesize* bytes of the file contents are recorded.

Binary headers contain the same information in 2-byte (short) and 4-byte (long) integers as follows:

Bytes	Field Names
2	magic
2	dev
2	ino
2	mode
2	uid
2	gid
2	nlink
2	rdev
2	mtime
2	namesize
2	filesize

After the header information comes the filename, with *namesize* rounded up to the nearest 2-byte boundary. Then the file contents appear as in the ASCII archive. The byte ordering of the 2- and 4-byte integers in the binary format is machine-dependent and thus portability of this format is not easily guaranteed.

Related Information

The **compress**, **cpio**, **pax**, and **tar** commands

Also see the description of the **tar** file archive on page 827.

magic — Format of the /etc/magic file

Description

The **file** command uses the **/etc/magic** file in its attempt to identify the type of a binary file. Essentially, **/etc/magic** contains templates showing what different types of files look like.

The **magic** file contains lines describing magic numbers, which identify particular types of files. Lines beginning with a > or & character represent continuation lines to a preceding main entry:

> If the **file** command finds a match on the main entry line, these additional patterns are checked. Any pattern that matches is used. This may generate additional output; a single blank separates each matching line's output if any output exists for that line.

& If the **file** command finds a match on the main entry line, and a following continuation line begins with this character, that continuation line's pattern must also match, or neither line is used. Output text associated with any line beginning with the & character is ignored.

Each line consists of four fields, separated by *one or more tabs:*

(a) The first field is a byte offset in the file, consisting of an optional offset operator and a value. In continuation lines, the offset immediately follows a continuation character.

If no offset operator is specified, then the offset value indicates an offset from the beginning of the file.

The * offset operator specifies that the value located at the memory location following the operator be used as the offset. Thus, `*0x3C` indicates that the value contained in `0x3C` should be used as the offset.

The + offset operator specifies an incremental offset, based on the value of the last offset. Thus, `+15` indicates that the offset value is 15 bytes from the last specified offset.

(b) The next field is a type: `byte`, `short`, or `string`. It can be followed by an optional mask which is bitwise ANDed to the value prior to comparison. For example, `byte &0x80` looks at the high bit.

(c) The next field is a value, preceded by an optional operator. Operators only apply to nonstring types: `byte`, `short` and `long`. The default operator is = (exact match). The operators are:

=	Equal.
!	Not equal.
>	Greater than.
<	Less than.
&	All bits in pattern must match.
^	Any bits in pattern may match.
x or ?	Any value matches (must be the only character in the field). ? is an extension to traditional implementations of **magic**.

The `string` values to be matched can contain any valid ANSI C backslash sequence. Thus, to match a single backslash, \\ must be entered in the **magic** file.

Note: Because of its format, the **magic** file must use `–t` to match a tab character.

(d) The rest of the line is a string to be printed if the particular file matches the template. Note that the contents of this field is ignored if the line begins with the & continuation character. The fourth field may contain a **printf ()**-type format indicator to output the magic number (see **printf** for more details on format indicators).

Usage Notes

Characters from a code page other than IBM-1047 should not be added to the **/etc/magic** file (the default **magic** file).

Characters from a code page other than IBM-1047 can be used in alternate **magic** files that are specified by the **–m** option on the **file** command. These characters should only be used in the third field of the **magic** file template when the field type is `string`. They will only match files containing these characters when the **file** command is invoked in the non-IBM-1047 locale.

Examples

Here are some sample entries:

```
0   short                0x5AD4              DOS executable
*0x18                    Short               0x40
>*0x3c                   Short               0x6584C OS/2 linear
                                             executable
>*0x3C                   Short               0x454e
>+54byte                 1                   OS/2 format
>+54byte                 2                   Windows format
0   short                0xFDF0              DOS library
0   string               AH                  Halo bitmapped font file
0   short                0x601A              Atara ST contiguous
                                             executable
>14 long                 >0                  - not stripped
0 byte                   0X1F
>1 byte                  0x1E                Packed file
>1 byte                  0x9D                Compressed file
```

Related Information

The **file** command. Also, see "Enabling the **file** Utility" in *OS/390 UNIX System Services Planning*.

---

pax — Format of pax archives and special header summary files

Description

pax uses the USTAR archive format described in the **tar** file format description.

(For more information about the **tar** file format, see Appendix H.)

Special Header Summary File: example format

```
#00#IBMOS390_USTAR_VERS=1
#
# Archive Name: /tmp/spec.pax
#
# This file was created by the IBM OS/390 pax or tar utility.
# During the process of creating the archive from which this
# file was extracted, one or more of the source files to be
# stored in the archive was determined to have names or
# attributes that are not supported by the standard USTAR
# format (as described by POSIX.2 IEEE std 1003.2-1992).
# To preserve these files or these characteristics,
# one or more special header files (having the same name as
# this file) were inserted into the archive.  Those files
# are recognized by OS/390 pax and tar utilities and are
# used during extraction to restore the files to their
# original state.
#
# The purpose of this file is to summarize the information
# described by all OS/390 special header files stored in
# the archive so that users with versions of pax or tar
# that do not support these special header files can
# manually restore some or all of the files and file
# attributes described by them.  Note that some file
# attributes are specific to OS/390 and cannot be restored
# on other platforms.
#
# The remainder of this file consists of a set of records
# corresponding to each special header file stored in the
# archive.  Each set consists of a record describing the
# pathname, one or more reasons explaining why the file or
# attribute could not be stored, and the unix command,
# or commands, that would be used to restore the file or
# attribute.  Note that these commands use the pathnames
# of the file as they existed when archived and may not
# correspond to the current pathnames on your system.
#
```

Figure 21 (Part 1 of 2). Example of a Special Header Summary File

```
# Pathname: level0/longsymlink
# Reason:   1. FSUMF076 target of symbolic link
("level1/level2/level3/level4/level5/level6/level7/level8/
level9/level10/level11/level12/level13/level14/linkbase")
exceeds 100 chars.
# Unix restore command(s):
ln -s level1/level2/level3/level4/level5/level6/level7/
level8/level9/level10/level11/level12/level13/level14/
linkbase level0/longsymlink
#
# Pathname: level0/level1/level2/level3/level4/level5/
level6/level7/level8/level9/level10/level11/level12/
level13/level14/longhardlink
# Reason:   1. FSUMF076 target of hard link ("level0/level1/
level2/level3/level4/level5/level6/level7/level8/level9/
level10/level11/level12/level13/level14/linkbase")
exceeds 100 chars.
# Unix restore command(s):
ln level0/level1/level2/level3/level4/level5/level6/level7/
level8/level9/level10/level11/level12/level13/level14/
linkbase level0/level1/level2/level3/level4/level5/level6/
level7/level8/level9/level10/level11/level12/level13/level14
/longhardlink
```

Figure 21 (Part 2 of 2). Example of a Special Header Summary File

Portability

POSIX.2, X/Open Portability Guide.

Related Information

The **cpio**, **pax**, and **tar** commands.

Also see the descriptions of the **cpio** and **tar** file formats.

---

queuedefs — Queue description for at, batch, and cron

Description

The **queudefs** file describes the characteristics of the queues managed by the clock daemon **cron**. Each line in the file that is not a comment uses the following format to describe a queue:

```
q . [njobj] [nicen] [nwaitw]
```

where the fields are:

q Specifies the name of the queue. Jobs started by **at** default to queue *a*; jobs started by **batch** default to queue *b*, and **crontab** files default to queue *c*. Queue names can be any singlebyte character except a space, tab, newline, null, or number sign (#).

njob Specifies the maximum number of jobs that can be run in the queue simultaneously. If more than *njob* jobs are ready to run, **cron** runs the first *njob* jobs immediately, and runs the others as current jobs terminate. The default value is 100.

nice Specifies the nice value (see **nice**) that **cron** assigns to all jobs in the queue that are not run by a user ID with appropriate privileges. The default value is 2.

nwait Specifies the number of seconds that **cron** is to wait before it reschedules a job that was deferred because there were more than *njob* jobs running in that job's queue, or because more than 25 jobs were running in all queues. The default value is 60.

Lines beginning with a number sign (#) are comments, and are ignored.

Examples

Here is a sample **queudefs** file:.

```
#
# Sample queuedefs file
#
a.5j3n
b.3j1n90w
```

This file indicates that the *a* queue, for **at** jobs, can have a maximum of five jobs running simultaneously. **crontab** runs the jobs with a **nice** value of 3. Because there is no `nwait` field for this queue, if **cron** cannot run a job because too many other jobs are running, it waits 60 seconds before trying to run it again.

This file also states that the *b* queue, for **batch** jobs, can have a maximum of three jobs running simultaneously. **cron** runs the jobs with a **nice** value of 1. If **cron** cannot run a job because too many other jobs are running, it waits 90 seconds before trying to run it again. All other queues can run up to 100 jobs simultaneously; **cron** runs these jobs with a **nice** value of 2 and, if it cannot run a job because too many other jobs are running, it waits 60 seconds before trying to run it again.

Related Information

The **at**, **batch**, and **crontab** commands.

---

tags — Format of the tags file

Description

When you use the **vi :tag** or **ex :tag** command, or the **ex –t**, **more –5**, **vi –t**, option, that utility looks for a file called **tags** in the current directory. This lets you quickly locate various points of interest in a C program which may span more than one source file. These points of interest are *tags*.

The **tags** file contains tags for function definitions, preprocessor macro definitions, and `typedef` definitions.

For each tag, the **tags** file contains one line in the following form:

tagname sourcefile address

The tagname field is the name of the C function, macro, or typedef. The sourcefile field has the name of the source file containing the tag named **tagname**. The address field is an editor address within sourcefile to reach the tag definition. This is either a line number in the file or a regular expression (enclosed in ? or / characters) that uniquely matches the line of source code where the tag appears. A tab character separates each field.

For **vi** or **more** to use the **tags** file correctly, it must be sorted by tagname using the POSIX locale's collation sequence.

Related Information

The **more**, **sort**, and **vi** commands.

---

tar — Format of tar archives

Description

tar reads and writes headers in either the original TAR format from UNIX systems or the USTAR format defined by the POSIX 1003.1 standard.

The **pax** command reads and writes headers in any of the **tar** formats.

The **tar** command supports both the older UNIX-compatible **tar** formats and the new USTAR format. The USTAR format allows more information to be stored and supports longer pathnames.

A **tar** archive, in either format, consists of one or more blocks, which are used to represent member files. Each block is 512 bytes long; you can use the **–b** option with **tar** to indicate how many of these blocks are read or written (or both) at once.

Each member file consists of a header block, followed by zero or more blocks containing the file contents. The end of the archive is indicated by two blocks filled with binary zeros. Unused space in the header is left as binary zeros.

The header information in a block is stored in a printable ASCII form, so that **tar** archives are easily ported to different environments. If the contents of the files on the archive are all ASCII, the entire archive is ASCII.

Figure 22 shows the UNIX format of the header block for a file:

Figure 22. Archive File: UNIX-Compatible Format

Field Width	Field Name	Meaning
100	name	Name of file
8	mode	File mode
8	uid	Owner user ID
8	gid	Owner group ID
12	size	Length of file in bytes
12	mtime	Modify time of file
8	chksum	Checksum for header
1	link	Indicator for links
100	linkname	Name of linked file

- A directory is indicated by a trailing /(slash) in its `name`.
- The `link` field is: `1` for a linked file, `2` for a symbolic link, `0` otherwise.

tar determines that the USTAR format is being used by the presence of the null-terminated string USTAR in the `magic` field. All fields before the `magic` field correspond to those of the UNIX format, except that `typeflag` replaces the link field.

Figure 23. Archive File: USTAR Format

Field Width	Field Name	Meaning
100	name	Name of file
8	mode	File mode
8	uid	Owner user ID
8	gid	Owner group ID
12	size	Length of file in bytes
12	mtime	Modify time of file
8	chksum	Checksum for header
1	typeflag	Type of file
100	linkname	Name of linked file
6	magic	USTAR indicator
2	version	USTAR version
32	uname	Owner user name
32	gname	Owner group name
8	devmajor	Device major number
8	devminor	Device minor number
155	prefix	Prefix for filename

Description of the Header Files

In the headers:

- The `name` field contains the name of the archived file. On USTAR format archives, the value of the prefix field, if non-null, is prefixed to the `name` field to allow names longer than 100 characters.
- The `magic`, `uname`, and `gname` fields are null-terminated character strings
- The `name`, `linkname`, and `prefix` fields are null-terminated unless the full field is used to store a name (that is, the last character is not null).
- All other fields are zero-filled octal numbers, in ASCII. Trailing nulls are present for these numbers, except for the `size`, `mtime`, and `version` fields.
- `prefix` is null unless the filename exceeds 100 characters.
- The `size` field is zero if the header describes a link.
- The `chksum` field is a checksum of all the bytes in the header, assuming that the chksum field itself is all blanks.
- For USTAR, the `typeflag` field is a compatible extension of the `link` field of the older **tar** format. The following values are recognized:

Flag	File Type
0 or null	Regular file
1	Link to another file already archived
2	Symbolic link
3	Character special file
4	Block special file (not supported)
5	Directory

6	FIFO special file
7	Reserved
S	OS/390 extended USTAR special header
T	OS/390 extended USTAR special header summary (**S** and **T** are OS/390 extensions. See "OS/390 Extended USTAR Support" on page 441 for more information.)
A–Z	Available for custom usage

- In USTAR format, the `uname` and `gname` fields contain the name of the owner and group of the file, respectively.

Compressed **tar** archives are equivalent to the corresponding archive being passed to a 14-bit **compress** command.

Related Information

The **cpio** and **tar** commands

Also see the description of the **cpio** file archive on page 821.

---

utmpx — Format of login accounting files

Description

Login accounting information is stored in two files:

- **/etc/utmpx** holds the current state of each item being accounted
- **/etc/wtmp** maintains the history of changes to each accounting item

Both files are arrays of the following binary records described in the form of a C data structure:

```
#include <sys/types.h>

struct  utmpx {
        char    ut_user[8];   /* Login user name */
        char    ut_id[4];     /* ID from &[.ETCDIR]/inittab */
        char    ut_line[12];  /* Login terminal name */
        short   ut_pid;       /* Process id */
        short   ut_type;      /* Entry type (see below) */
        struct  exit_status {
          /* Process termination+exit status*/
                short    e_termination;
                short    e_exit;
        } ut_exit;            /* Used with DEAD_PROCESS */
        time_t   ut_time;     /* Entry created */
};

#define EMPTY         0   /* Unused */
#define RUN_LVL       1   /* Set new run level */
#define BOOT_TIME     2   /* System boot */
#define OLD_TIME      3   /* Time of date change - delta */
#define NEW_TIME      4   /* Time of date change + delta */
#define INIT_PROCESS  5   /* Process started by &[.ETCDIR]/init */
#define LOGIN_PROCESS 6   /* Login process */
#define USER_PROCESS  7   /* User process */
#define DEAD_PROCESS  8   /* Contains exit status */
#define ACCOUNTING    9   /* Other accounting */
```

Files

/etc/utmpx
Reflects the current state of the accounting entries; for example, who is logged in, when the date was last set, and so on.

/etc/wtmp
Contains a history of changes to any of the accounting entries.

Related Information

The **who** command

---

uucp — Format of UUCP working files

Description

UUCP uses three kinds of working files when handling UUCP requests, command, data, and execute.

All three files are stored in a subdirectory for each specific site, named after the site's name. For example, because the UUCP spool directory is **/usr/spool/uucp**, then the directory **/usr/spool/uucp/south** is used for all the command, data, and execute files associated with the remote site `south`.

Command Files

Command files are created by the mail routing agents **uucp** and **uux**. On UUCP sites, command files have names such as `C.targetA28B9`, where `target` is the name of the destination site, `A` is the job grade (as set by the **–g** option to **uucp**, and `28B9` is the sequence number or job identification number. (You can use the **–j** option on **uucp** and **uux**, as well as **uustat** to find the job identification number.)

In a command file, each line records one file transfer request. The fields are defined as follows:

`type` The `type` field can be one of the following:

- `R` Receive a file from remote to local site.
- `S` Send a file from local to remote site.

`source` The name of the source file.

`destination` The name of the file after the transfer completes, whether to the remote site (`S` request) or the local site (`R` request).

Special characters such as the tilde (~) are still present, because they are expanded on the destination site.

`sender` The login name of the user who issued the command. This is normally your login name, though some programs (such as mail programs) use a different login name for their requests.

`options` The command options, which correspond to options of the **uucp** and **uux** commands.

- `C` Use the data file name as the source for the copy; this can only be used with the `S` request.
- `c` Use the source file name as the source for the copy.
- `d` Create intermediate target directories as required. This is the default.
- `m` Send mail to the user when the transfer is complete.
- `n` Send mail to the user specified by the notification name when the transfer is complete.

`datafile` The temporary file to be used if the source file was copied into the spool directory; it is only used with the `S` request. If **C** is one of the options, the data file is the name of the copy in the destination site's data spool directory. Otherwise, the placeholder name `D.0` is used.

`file mode` The UNIX-style permission mode of the source file. It is only used with the `S` request. All files sent have mode `0666`, plus whatever execute permissions the original file had. (For an explanation of the modes, see **chmod**.)

`notification` The login name of the person to be notified after the job request completes. It is used only with the `S` request if `n` is one of the options.

Examples

1. The command

   ```
   uucp –m /memos.001 /memos.002 south!~/
   ```

   copies the files **/memos.001** and **/memos.002** root directory to the public UUCP directory on `south`. Assuming your user name is `eve`, a command file containing these lines is created in the UUCP spool directory **/usr/spool/uucp/south**:

   ```
   S /memos.001 ~/memos.001 eve –mcd D.0 0777
   S /memos.002 ~/memos.002 eve –mcd D.0 0777
   ```

2. The command

   ```
   uucp south!~/index ~/
   ```

   generates a command file on your site in the UUCP spool directory **/usr/spool/uucp/south** containing this line:

   ```
   R ~ /index ~/index eve –cd
   ```

Data Files

Data files contain data to be transferred to the remote site. They are created by **uucp** if the **–C** option is used, and by **uux** and **mail** programs.

On UUCP sites, data files have names like `D.source9B73001`, where `source` is the name of the site that the data file originated from (the local site for an `S` request, or the remote site for an `R` request), `98B3` is the sequence number, and `001` is the subsequence number, used when a request generates more than one data file.

Data files created by **uucp** contain files to be copied. Data files created by **uux** which contain commands for the remote site become execute files at their destination.

Mail sites typically create two data files, one containing the message and the other containing the command to run the mail routing agent on the remote site.

Examples

UUCP data files contain data to be copied. The contents of **uux** data files and commands that generate remote commands are execute files intended for other sites. For example, a mail message to `north` generates two data files in the UUCP spool directory **/usr/spool/uucp/north**

```
D.north000A001
X.northX000A002
```

These working files are created:

`D.north000A001`	Text of mail message
`X.northX000A002`	Execute file

The execute file contains the **uux** request for the mail routing program to be run on `north`.

Execute Files

Execute files are data files containing commands that are created on other sites and copied to your site. The files are treated as execute files when they arrive at your site, where the commands are run by **uuxqt**.

On UUCP sites, execute files are named as:

```
X.remotX28A3003
```

where *remot* is the first five characters of the destination site's name, `X` is the job grade (execute files always have the grade `X`), and `28A3` is the sequence number.

Each execute file contains one command, and the necessary information to run the command. The type of information on each line is identified by the first character in the line. Not all lines are used in all files, and not all UUCP implementations support all of these lines. The first line in an execute file must be a `U` line, and the last line must be a `C` line.

`#` Indicates a comment. Comments and unrecognized commands are ignored.

`C` *command*
Requests that *command* be run. *command* is a string that includes the program and arguments. This line must be present and must be the last line in the execute file.

`E` Processes the command with **execve()**. If the `E` line is present, **uuxqt** runs a **fork()/ecec()** sequence, unless the command contains a shell metacharacter. In that case, **uuxqt** invokes a shell to run the command.

`e` Processes the command by the POSIX shell. It is intended to handle commands that require special processing. If the `e` line is present, **uuxqt** invokes the defined shell to run the command.

`F` *filename* **[** *xqtname* **]**
Names *filename*, a file required for the command to be run. This is usually a file that is transferred from the site that **uux** was executed from, but it can also be a file from the local site or some other site. If *filename* is not from the local site, then it is usually a file in the spool directory. Multiple `F` lines are allowed. Any file other than the standard input file requires the *xqtname* argument and is copied to the execution directory as *xqtname*. If the standard input file is not from the local site, it appears in both an **F** command and an **I** command.

`I` *stdin* Names the file that supplies standard input to the command. If the standard input file is not from the site running the command, the file is also in an **F** command. If there is no standard input file, behavior depends on the site implementation. **uuxqt** rejects the command; some UNIX implementations use **/dev/null** as the standard input. Only one **I** line can be present in an execute file; the corresponding **F** line must precede the file.

`N` No mail message should be sent, even if the command failed.

`n` Requests a mail message be sent if the command succeeded. Normally a message is sent only if the command failed.

O *stdout* **[** *site* **]**
: Names the standard output file. The optional second argument names the site to which the file should be sent. If there is no second argument, the file should be created on the executing site. Only one **O** line can be present in an execute file; the corresponding **F** line must precede the **O** line.

U *user site*
: Names the user who requested the command and the site that the request came from. This line must be present and must be the first line in the execute file.

Z
: Specifies that a mail message should be sent if the command failed. This is the default for **uuxqt**.

  Not all these commands may be implemented at your site. For a list of the commands not supported by **uuxqt**, see **uuxqt**.

  Although most execute files are generated on other sites, complex **uux** commands that retrieve files from multiple sites can generate execute commands in the local spool directory, where *local* is the name of your site.

Examples

The following is an example of an execute file to run **rmail** on the site `south`. The data file containing the mail message is `D.south49Z3`. This is an execute file that might be created by the **mailx** command:

```
U eve north
F D.south49Z3
I D.south49Z3
C rmail bob
```

This command originated with user `eve` on `north`. It requests that **rmail** be run with the argument `bob` on the target site. The file **D.south49Z3** is required to run the command and is used as standard input for the command.

Portability

X/Open Portability Guide

Related Information

uucico, **uucp**, **uux**, **uuxqt**

Appendix I. Setting the Local Time Zone with the TZ Environment Variable

This appendix discusses how to use the **TZ** environment variable when setting the local time zone.

TZ Environment Variable

Format

TZ= *standardHH[:MM[:SS]] [daylight[HH[:MM[:SS:]]] [,startdate[/starttime],enddate[/endtime]]]*

Description

All commands assume that times stored in the file system and returned by the operating system are stored using Greenwich Mean Time (GMT) or Universal Time Coordinated (UTC), hereafter referred to as the universal reference time. The mapping from the universal reference time to local time is specified by the **TZ** (time zone) environment variable.

The value of the **TZ** environment variable has the following five fields (two required and three optional):

standard An alphabetic abbreviation for the local standard time zone (for example, (GMT, EST, MSEZ).

HH[:MM[:SS]]
The time offset westwards from the universal reference time. A leading minus sign (–) means that the local time zone is east of the universal reference time. An offset of this form must follow *standard* and can also optionally follow *daylight*. An optional colon (:) separates hours from optional minutes and seconds.

If *daylight* is specified without a *daylight* offset, daylight savings time is assumed to be one hour ahead of the standard time.

[daylight] The abbreviation for your local daylight savings time zone. If the first and third fields are identical or if the third field is missing, Daylight Saving Time conversion is disabled. The number of hours, minutes, and seconds your local Daylight Savings Time is offset from UTC when Daylight Savings Time is in effect. If the Daylight Savings Time abbreviation is specified, and the offset omitted, the offset of one hour is assumed.

[,startdate[/starttime],enddate[/endtime]]
A rule that identifies the start and end of Daylight Savings Time, specifying when Daylight Savings Time should be in effect. Both the *startdate* and *enddate* must be present, and must either take the form `J`*n*, *n*, or `M`*m.n.d.*.

- `J`*n* is the Julian day *n* (1 <= *n* <=365) and does not account for leap days.

© Copyright IBM Corp. 1996, 2000

- *n* is the zero-based Julian day (0 <= *n* <= 365). Leap days are counted; therefore, you can refer to February 29th.
- M*m.w.d* defines the day (0 <= *d* <= 6) of week *w* (1 <= *w* <= 5) of month *m* (1 <= 12) of the year. Week 5 has the last day (*d*) in month *m*, which may occur in either the fourth or fifth week). Week 1 is the first week in which the *d*th day occurs. Day zero is Sunday.

Neither *starttime* nor *endtime* are required. If they are omitted, their values default to 02:00:00. If this Daylight Savings Time rule is omitted altogether, the values in the rule default to the standard American Daylight Savings Time rules starting at 02:00:00 the first Sunday in April and ending at 02:00:00 the last Sunday in October.

When the **TZ** variable is not set, time conversions behave as if **TZ** were set to `TZ=GMT0`.

Portability

This interpretation of the **TZ** environment variable is a superset of that supported by UNIX System V.

Related Information

The **date** and **touch** commands.

Appendix J. Environment Variables

This appendix contains a partial list of environment variables.

- For the **c89/cc/c++** environment variables, refer to the **c89/cc/c++** command for descriptions of the environment variables used.
- For the **mailx** environment variables, refer to the **mailx** command for descriptions of the environment variables used.
- For the **tcsh** environment variables, refer to the **tcsh** command for descriptions of the environment variables used.
- For the **tso** environment variables, refer to the **tso** command for descriptions of the environment variables used.
- For the **vi** environment variables, refer to the **vi** command for descriptions of the environment variables used.
- For a list of built-in environment variables, refer to Figure 13 on page 507.

Figure 24 lists frequently-used environment variables and their purposes.

Figure 24 (Page 1 of 3). Miscellaneous Environment Variables

Environment Variable	Purpose
_BPX_ACCT_DATA	Allows users to change the mount data when creating a new process.
_BPX_BATCH_SPAWN	Causes BPXBATCH to use spawn instead of fork/exec and allows data definitions to be carried over into the spawned process.
_BPX_BATCH_UMASK	Allows the user the flexibility of modifying the permission bits on newly created files instead of using the default mask.
_BPX_JOBNAME	Specifies the job name of the process. You can use **_BPX_JOBNAME** to change the job name.
_BPX_PTRACE_ATTACH	If it is set to YES, programs invoked via the spawn, exec and attach_exec callable services or via the C language spawn() and exec() family of functions are loaded into user modifiable storage. Then these target programs can be debugged. The programs that are loaded into storage during the execution of the target program, except for modules loaded from LPA, are also loaded.
_BPX_SHAREAS	Specifies whether the spawned child process is to be run in a separate address space from the login shell's space or in the same address space.
_BPX_SPAWN_SCRIPT	Specifies whether the specified file is to be treated as a shell script, if it is not in the correct format to be a process image file or not a REXX exec.
_BPX_TERMPATH	Enables shell scripts to determine if the user logged on from TSO, rather than from rlogin or telnet.
_BPX_USERID	**_BPX_USERID** is used by daemons to set the userid for a child process where the child process will either not create additional processes or will clear the environment variable after it gets control.

© Copyright IBM Corp. 1996, 2000

Figure 24 (Page 2 of 3). Miscellaneous Environment Variables

Environment Variable	Purpose
_BPXK_MDUMP	**_BPX_MDUMP** enables a user to specify where a SYSMDUMP will be written to. Allowable values for **_BPX_MDUMP** are: **OFF** Request the dump to be written to the current working directory. This is the default. **MVS data set name** Request the dump to be written to an MVS data set. The data set name must be a fully qualified data set name and can be up to 44 characters. The name can be specified in upper and/or lower case and will be folded to uppercase. **HFS file name** Request the dump to be written to an HFS file. The file name can be up to 1024 characters. The HFS file name must begin with a slash. The slash refers to the root directory, and the file will be created in that directory.
_CEE_ENVFILE	Enables a list of environment variables to be set from a specified file. This environment variable only takes effect when it is set through the runtime option `ENVAR` on initialization of a parent program. **Attention: _CEE_ENVFILE** is intended for use in batch programs. Since it has the ability to supersede built-in shell environment variables such as **LIBPATH**, it is strongly recommended that you do not use **_CEE_ENVFILE** in the shell environment. For more information on the **_CEE_ENVFILE** environment variable, see *OS/390 C/C++ Programming Guide*.
ENV	Contains the pathname of a login script containing commands and aliases. When you invoke **sh**, it executes this file before doing anything else. **sh** performs parameter substitution on this value and uses the result as the name of an initialization file, or login script. This file is run with the **.** (**dot**) command; see the **dot** command. This variable is usually set in your **.profile**.
EXINIT	Contains a list of **vi** commands to be run when the editor is started.
LC_ALL	Specifies the locale to be used to override any values for locale categories specified by **LANG** or certain **LC_** variables, such as **LC_COLLATE**, and **LC_MESSAGES**, which a user can set and interrogate. However, **LC_ALL** does not override **LC_CTYPE**.
LPDEST	Specifies the name of the printer that **lp** commands is to submit print jobs to. This variable takes precedence over **PRINTER**.
LIBPATH	Specifies the directory to search for a DLL (dynamic link library) filename. If it is not set, the working directory is searched.
MAIL	Specifies the pathname of your system mailbox.
MAKEFLAGS	Contains a series of **make** options that are used as default options.
MAKESTARTUP	Contains the pathname of the **make** stamp file.
MANPAGER	Contains an output filtering command for use when displaying man pages on a terminal.
MORE	Contains a list of options as they would appear on the command line. This variable takes preference over the **TERM** and **LINES** variables.
OPTARG	Contains the value of the option argument found by **getopts**.
OPTIND	Contains the index of the next argument to be processed by **getopts**.
PAGER	Contains an output filtering command for use when displaying man pages on a terminal.

Figure 24 (Page 3 of 3). Miscellaneous Environment Variables

Environment Variable	Purpose
PRINTER	If **LPDEST** is not specified, specifies the default name of the printer that **lp** commands is to submit print jobs.
READONLY	Specifies that files cannot be written to.
REPLY	Contains the input (including separators) if you did not specify any variables. That is, you can omit the variable from the command and use **READ** instead.
RESOLVER_CONFIG	Used by TCP/IP to include the name of an MVS data set or HFS file in the search order for **TCPIP.DATA**.
X_ADDR	Used by some TCP/IP functions such as **getnetbyaddr** to include the name of an MVS data set or HFS file in the search order for **HOSTS.ADDRINFO**.
X_SITE	Used by some TCP/IP functions such as **gethostbyname** to include the name of an MVS data set or HFS file in the search order for **HOSTS.SITEINFO**.
X_XLATE	Used by TCP/IP to include the name of an MVS data set or HFS file in the search order for **STANDARD.TCPXLBIN**.

Appendix K. Specifying MVS Dataset Names in the Shell Environment

Several utilities allow the user to specify an MVS dataset name in place of an HFS filename. See “Utilities supporting MVS dataset names” on page 844 for the current list. This section describes the syntax for specifying an MVS dataset name. Because MVS dataset names generally contain single quotes and parenthesis which can be misinterpreted by the shell, care needs to be taken to correctly escape these characters.

What follows are general rules for specifying MVS dataset names. Consult the description of each utility for more specific instructions or exceptions.

- MVS datasets are distinguished from HFS files by preceding them with two slashes (//). For example, to specify the MVS dataset name PROGRAM.OUTPUT, enter:

  ```
  //PROGRAM.OUTPUT
  ```

  If the double slashes were not used, the name would be interpreted as the HFS pathname PROGRAM.OUTPUT in the current working directory.

- Unless a utility specifically provides an option to disable uppercasing, the default approach is to uppercase all MVS dataset names before processing. For example, the following are all equivalent methods for specifying the MVS dataset PROGRAM.OUTPUT:

  ```
  //program.output
  //ProGram.OutPut
  //PROGRAM.OUTPUT
  ```

- The single quote (') and parenthesis (**()**) metacharacters are typically used to specify fully-qualified MVS names and PDS/PDSEs, respectively. These characters, however, are metacharacters that will be incorrectly interpreted by the shell. To prevent this, they must be escaped. The simplest approach is to place the entire name within double-quotes (**"**). Alternatively, these characters can be escaped by preceding each with a backslash (****). Some examples are:

  To specify the fully qualified MVS dataset 'SMITH.PROGRAM.OUTPUT':

  ```
  "//'smith.program.output'"
   //\'smith.program.output\'
  ```

  To specify the fully qualified partitioned dataset 'SMITH.PROGRAM.SOURCE(FILE1)':

  ```
  "//'smith.program.source(file1)'"
   //\'smith.program.source\(file1\)\'
  ```

  To specify the non-qualified partitioned dataset PROGRAM.SOURCE(FILE1):

  ```
  "//program.source(file1)"
   //smith.program.source\(file1\)
  ```

© Copyright IBM Corp. 1996, 2000

Utilities supporting MVS dataset names

The following utilities currently support the use of MVS filenames. Consult the description for each utility for limitations and exceptions:

- **cp**
- **mv**
- **pax**
- **tar**
- **c89**

Appendix L. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
USA

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

© Copyright IBM Corp. 1996, 2000

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P300
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

ACF/VTAM	Open Class
AIX	OS/2
BookManager	OS/390
C/MVS	RACF
C++/MVS	RISC Systems/6000
C/370	SecureWay
DFSMS	SOM
DFSMS/MVS	SOMobjects
eNetwork	SP
ESCON	SP2
IBM	S/390
IBMLink	System/370
Language Environment	System Object Model
Library Reader	Talklink
MVS/ESA	VTAM

UNIX is a registered trademark in the United States and other countries licensed exclusively through The Open Group.

Windows is a trademark of Microsoft Corporation.

The following terms may be trademarks or service marks of others:

ANSI	American National Standards Institute
BSD	University of California
DFS	Transarc Corporation
IEEE	Institute of Electrical and Electronics Engineers
InterOpen	Mortice Kern Systems Inc.
ISO	International Organization for Standardization

Lotus	Lotus Development Corporation
MKS	Mortice Kern Systems Inc.
Notes	Lotus Development Corporation
POSIX	Institute of Electrical and Electronics Engineers
Tivoli	Tivoli Systems, Inc.

Acknowledgments

InterOpen Shell and Utilities is a source code product providing POSIX.2 (Shell and Utilities) functions to the OS/390 UNIX services offered with MVS. InterOpen/POSIX Shell and Utilities is developed and licensed by Mortice Kern Systems (MKS**) Inc. of Waterloo, Ontario, Canada.

The information contained in the glossary section and tagged by the word [POSIX] is copyrighted information of the Institute of Electrical and Electronics Engineers, Inc., extracted from IEEE** Std 1003.1-1990, IEEE P1003.0, and IEEE P1003.2. This information was written within the context of these documents in their entirety. The IEEE takes no responsibility or liability for and will assume no liability for any damages resulting from the reader's misinterpretation of said information resulting from the placement and context in this publication. Information is reproduced with the permission of the IEEE.

Glossary

This glossary defines terms and abbreviations used in OS/390 UNIX System Services. If you do not find the term you are looking for, refer to the index of the appropriate OS/390 UNIX System Services manual or view the *IBM Dictionary of Computing* located at:

http://www.ibm.com/networking/nsg.nsgmain.htm

This glossary includes terms and definitions from:

- *IBM Dictionary of Computing* (New York: McGraw-Hill, 1994).
- *Information Technology—Portable Operating System Interface (POSIX),* from the POSIX** series of standards for applications and user interfaces to open systems, copyrighted by the Institute of Electrical and Electronics Engineers (IEEE**). Copies of all POSIX drafts and standards may be purchased from IEEE at 1-800-678-IEEE.
  - Definitions identified by *[POSIX.0]* are from *Part 0: Standards Project, Draft Guide to the POSIX Open System Environment,* P1003.0 Draft 15 (June 1992), an unapproved draft subject to change.
  - Definitions identified by *[POSIX.1]* are from *Part 1: System Application Program Interface (API) [C Language],* approved September 28, 1990, as IEEE Std 1003.1-1990 by the IEEE Standards Board, and adopted in 1990 as an International Standard (ISO/IEC 9945-1: 1990) by the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC).
  - Definitions identified by *[POSIX.2]* are from *Part 2: Shell and Utilities,* P1003.2.
- *American National Standard Dictionary for Information Systems,* ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies may be purchased from the American National Standards Institute, 1430 Broadway, New York, New York 10018. Definitions are identified by the symbol *(A)* after the definition.
- *Information Technology Vocabulary,* developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1). Definitions from published sections of these vocabularies are identified by the symbol *[I]* after the definition. Definitions taken from draft international standards, committee drafts, and working papers being developed by ISO/IEC JTC1/SC1 are identified by the symbol *[T]* after the definition, indicating that final agreement has not yet been reached among the participating national bodies of SC1.
- *CCITT Sixth Plenary Assembly Orange Book, Terms and Definitions* and working documents published by the International Telecommunication Union, Geneva, 1978. These are identified by the symbol *[CCITT/ITU]* after the definition.
- Open Software Foundation (OSF**). These are identified by the symbol *[OSF]* after the definition. Copies of OSF documents may be obtained from Open Software Foundation, Inc., 11 Cambridge Center, Cambridge, MA 02142.

Sequence of Entries: For clarity and consistency of style, this glossary arranges the entries alphabetically on a letter-by-letter basis. In other words, only the letters of the alphabet are used to determine sequence; special characters and spaces between words are ignored.

Organization of Entries: Each entry consists of a single-word or multiple-word term or the abbreviation or acronym for a term, followed by a commentary. A commentary includes one or more items (definitions or references) and is organized as follows:

1. An item number, if the commentary contains two or more items.
2. A usage label, indicating the area of application of the term, for example, “In programming,” or “In TCP/IP.” Absence of a usage label implies that the term is generally applicable to the OS/390 UNIX System Services interface, to IBM, or to data processing.
3. A descriptive phrase, stating the basic meaning of the term. The descriptive phrase is assumed to be preceded by “the term is defined as ....” The part of speech being defined is indicated by the opening words of the descriptive phrase: “To ...” indicates a verb, and “Pertaining to ...” indicates a modifier. Any other wording indicates a noun or noun phrase.
4. Annotative sentences, providing additional or explanatory information.
5. References, directing the reader to other entries or items in the dictionary.
6. A source label—for example, *[A], [I], [T], [CCITT/ITU], [OSF], [POSIX.0], [POSIX.1],* or *[POSIX.2]*—that follows the definition and identifies the originator of the definition. Definitions without source labels are IBM definitions.

References: The following cross-references are used in this glossary:

Contrast with. This refers to a term that has an opposed or substantively different meaning.

© Copyright IBM Corp. 1996, 2000

Synonym for. This indicates that the term has the same meaning as a preferred term, which is defined in its proper place in the glossary.
Synonymous with. This is a backward reference from a defined term to all other terms that have the same meaning.
See. This refers you to multiple-word terms that have the same last word.
See also. This refers the reader to related terms that have a related, but not synonymous, meaning.
Deprecated term for or **Deprecated abbreviation for.** This indicates that the term or abbreviation should not be used. It refers to a preferred term, which is defined in its proper place in the glossary.

Selection of Terms: A term is a word or group of words to be defined. In this glossary, the singular form of the noun and the infinitive form of the verb are the terms most often selected to be defined. If the term may be abbreviated, the abbreviation is given in parentheses immediately following the term. The abbreviation is also defined in its proper place in the glossary.

A

absolute pathname. (1) A pathname beginning with a slash. The predecessor of the first filename in the pathname is taken to be the root directory of the process. [POSIX.1] (2) The name of any directory or file expressed as a string of directories and files beginning with the root directory. (3) The name of a shared library key that begins with a slash. The system does not append any directories to the name of the shared library key, and it attempts to open a file with that name. [OSF] (4) See also *pathname, relative pathname*.

access permission. A group of designations that determine who can access a particular file and how the user can access the file.

action. In **awk**, **lex**, and **yacc**, a C language program fragment that defines what the program does when it recognizes input. [OSF]

alert. (1) To cause the user's terminal to give some audible or visual indication that an error or some other event has occurred. When the standard output is directed to the terminal device, the method of alerting the terminal user is unspecified. When the standard output is not directed to the terminal device, the alert shall be accomplished by writing the <alert> character to standard output (unless the utility description indicates that the use of standard output produces undefined results in this case). [POSIX.2] (2) A character that causes a terminal to attempt to alert its user via a visual or audible notification. The <alert> is the character designated by "\a" in the C language binding. It is unspecified whether this character is the exact sequence transmitted to an output device by the system to accomplish the alert function. [POSIX.2]

alias. (1) An alternate name for a shell command. An alias for a command can be defined with options different from those for the command itself. An alias can be a convenient shorthand for a complicated command line, such as a pipeline. (2) An alternate name for a user, system, or file that can be used in place of the real name of the object. [OSF] An alias may be defined to change the default options given to a particular command (for example, **pg**), or may be a shorthand for a more complicated command line (for example, a pipeline). (3) An alternate label. For example, a label and one or more aliases may be used to refer to the same data element or point in a computer program. [A] (4) An alternate name for a member of a partitioned data set. (5) Synonymous with nickname.

alphabetic. Pertaining to the set of letters and symbols, excluding digits, used in a language. This set usually consists of the uppercase and lowercase letters plus special symbols (such as $ and _) allowed by a particular language. [OSF] See also *alphabetic character*.

alphabetic character. A letter or other symbol, excluding digits, used in a language: usually the uppercase and lowercase letters plus other special symbols (such as $ and _) allowed by a particular language. See also *alphanumeric character, digit*.

alphanumeric. (1) Pertaining to data that consists of letters, digits, and usually other characters, such as punctuation marks. [T] (2) Pertaining to a character set that contains letters, digits, and usually other characters, such as punctuation marks. [A] (3) Synonymous with alphameric.

alphanumeric character. A letter, number, or other symbol, such as a punctuation mark or mathematical symbol. See also *alphabetic character, digit*.

American National Standard Code for Information Interchange (ASCII). The code developed by ANSI for information interchange among data processing systems, data communication systems, and associated equipment. The ASCII character set consists of 7-bit control characters and symbolic characters. [OSF] See also *extended binary-coded decimal interchange code (EBCDIC)*.

append. The action that causes data to be added to the end of existing data. [OSF]

appropriate privilege. Having sufficient authority to perform a requested function. If applications have been assigned an UID of 0, which indicates superuser authority, they can call any service. Security product

profiles can also grant or restrict a user's access to certain functions.

archive. (1) To store programs and data for safekeeping. [OSF] (2) A copy of one or more files or a copy of a database that is saved in case the original data is damaged or lost. (3) Synonymous with backup, backup copy.

argument. (1) An independent variable. [I][A] (2) Any value of an independent variable, for example, a search key; a number identifying the location of an item in a table. [I][A] (3) A parameter passed between a calling and a called program. (4) A parameter passed to a utility as the equivalent of a single string in the *argv* array created by one of the POSIX.1 **exec** functions. An argument is one of the options, option-arguments, or operands following the command name. [POSIX.2] (5) Numbers, letters, or words that expand or change the way a command works. [OSF] (6) See *command-line argument*.

argument list. A string of arguments.

arithmetic expression. (1) An expression that contains arithmetic operations and operands that can be reduced to a single numeric value. [T] Synonymous with arithmetic statement. (2) A sequence of operations yielding a value. For example, an arithmetic operation such as $a + b * c$ is an arithmetic expression because the result of the operation is a single value. (3) One or more arithmetic operators and arithmetic primaries, the evaluation of which produces a numeric value. An arithmetic expression can be an unsigned arithmetic constant, the name of an arithmetic constant, a reference to an arithmetic variable, array element, or function, or a combination of such primaries formed by using arithmetic operators and parentheses. [OSF] See also *arithmetic operator*. (4) In assembler, a conditional assembly expression that is a combination of arithmetic terms, arithmetic operators, and paired parentheses.

arithmetic operator. A symbol that directs a compiler to perform an arithmetic operation. [OSF]

array. (1) A number of items stored together, which a user can quickly retrieve by supplying the correct index. Both **awk** and the OS/390 shells support arrays. (2) A variable that contains an ordered group of data objects. All objects in an array have the same data type. [OSF] (3) An arrangement of data in one or more dimensions, such as a list, table, or multidimensional arrangement of items. (4) In programming languages, an aggregate that consists of data objects, with identical attributes, each of which may be uniquely referenced by subscripting. [I]

ASCII. American National Standard Code for Information Interchange.

asterisk. The character "*". [POSIX.2]

automount rule. A generic or specific entry in an automount map file.

B

background. (1) In multiprogramming, the conditions under which low-priority programs are executed. (2) A mode of program execution in which the shell does not wait for program completion before prompting the user for another command. [OSF] Synonym for *background job*. (3) Contrast with *foreground.*

background job. An MVS job that the MVS initiator selects from a queue and starts running. Synonymous with background, batch job.

background process. (1) A process that is a member of a background process group. [POSIX.1] (2) A process that does not require operator intervention but can be run by the computer while the workstation is used to do other work. (3) An process that an interactive user starts running and that cannot interact with the user. An interactive user can move a background process to the foreground. (4) See also *background process group.*

background process group. Any process group, other than a foreground process group, that is a member of a session that has established a connection with a controlling terminal. [POSIX.1]

backslash. The character "\"—also known as a *reverse solidus.* [POSIX.2] The backslash enables a user to escape the special meaning of a character. That is, typing a backslash before a character tells the system to ignore any special meaning the character might have.

backspace. A character that normally causes printing (or displaying) to occur one column position previous to the position about to be printed. The <backspace> is the character designated by "\b" in the C language binding. It is unspecified whether this character is the exact sequence transmitted to an output device by the system to accomplish the backspace function. The <backspace> character defined here is not necessarily the ERASE special character defined in POSIX.1. [POSIX.2]

basename. (1) The final, or only, filename in a pathname. [POSIX.2] The basename is the part of a pathname that remains after all directory names are removed. For example, for the pathname **dir1/dir2/file.c**, the basename is **file.c**. (2) The last element to the right of a full pathname. (3) A filename specified without its parent directories.

basic mode. A central processor mode that does not use logical partitioning. Contrast with *logically partitioned (LPAR) mode*.

basic regular expression. A pattern (sequence of characters or symbols) constructed according to the rules defined in POSIX.2 2.8.3. [POSIX.2]

binary. (1) Pertaining to a selection, choice, or condition that has two possible different values or states. [I][A] (2) Pertaining to a fixed radix numeration system having a radix of two. [I][A] (3) Pertaining to a system of numbers to the base 2; the binary digits are `0` and `1`. [I][A] (4) Pertaining to a choice of two conditions, such as on-off or yes-no. [OSF]

binary data. (1) Any data not intended for direct human reading. Binary data may contain "unprintable" characters, outside the range of text characters. (2) A type of data consisting of numeric values stored in bit patterns of `0`s and `1`s. Binary data can cause a large number to be placed in a smaller space of storage.

binary file. A file that contains codes that are not part of the character set. Binary files utilize all 256 possible values for each byte in the file.

binder. Synonym for *linkage editor*.

blank. One of the characters that belong to the *blank* character class as defined via the LC_CTYPE category in the current locale. In the POSIX locale, a <blank> is either a <tab> or a <space>. [POSIX.2]

Boolean. (1) Pertaining to the processes used in the algebra formulated by George Boole. [A] (2) A value of `0` or `1` represented internally in binary notation.

Bourne Shell. The shell based on UNIX V. See also *shell*.

braces. The characters "{" (*left brace*) and "}" (*right brace*). When used in the phrase "enclosed in (curly) braces," the symbol "{" immediately precedes the object to be enclosed, and "}" immediately follows it. Synonymous with <left-brace> and <right-brace>. [POSIX.2] Synonymous with curly brackets.

brackets. The characters "[" (*left bracket*) and "]" (*right bracket*), also known as *square brackets*. When used in the phrase "enclosed in (square) brackets," the symbol "[" immediately precedes the object to be enclosed, and "]" immediately follows it. Synonymous with <left-square-bracket> and <right-square-bracket>. [POSIX.2]

byte. (1) Eight bits of data storage. [OSF] (2) The unit of data storage large enough to hold any member of the basic character set of the execution environment. It is possible to express the address of each individual byte of an object uniquely. A byte is composed of a contiguous sequence of bits, the number of which is implementation defined. The least significant bit is called the *low-order* bit; the most significant is called the *high-order* bit. [POSIX.1] (3) A string consisting of a certain number of bits, usually eight, treated as a unit, and representing a character. (4) A group of eight adjacent binary digits representing one EBCDIC character. (5) A sequence of eight adjacent binary digits that are used as a unit.

C

carriage return (CR). (1) A keystroke generally indicating the end of a command line. (2) In text data, the action that indicates to continue printing at the left margin of the next line. [OSF] (3) A character that shall cause printing to start at the beginning of the same physical line in which the <carriage return> occurred. The <carriage return> is the character designated by the "\r" in the C language binding. It is unspecified whether this character is the exact sequence transmitted to an output device by the system to accomplish the movement to the beginning of the line. [POSIX.2]

Cartesian product. The product of two sets A and B (expressed as "A×B"), in which *x* is an element of A and *y* is an element of B, and *x* and *y* are themselves part of a set of ordered pairs (*x, y*). For example, if A={1,2} and B={3,4}, then A×B={(1,3), (1,4), (2,3), (2,4)}.

case-sensitive. Pertaining to the ability to distinguish between uppercase and lowercase letters.

catalog. (1) A directory of files and libraries, with reference to their locations. [I][A] (2) To enter information about a file or a library into a catalog. [I][A] (3) The collection of all data set indexes that are used by the control program to locate a volume containing a specific data set.

CECP. Country-extended code page.

central processor (CP). The part of the computer that contains the sequencing and processing facilities for instruction execution, initial program load, and other machine operations.

character. (1) A letter, digit, or other symbol. (2) A letter, digit, or other symbol that is used as part of the organization, control, or representation of data. A character is often in the form of a spatial arrangement of adjacent or connected strokes. [A] (3) A sequence of one or more bytes representing a single graphic symbol. The term corresponds in the C Standard to the term *multibyte character,* noting that a single-byte character is a special case of a multibyte character. In POSIX, however, a character has no necessary relationship with storage space, and *byte* is used when

storage space is discussed. [POSIX.1] (4) A member of a set of elements that is used for the representation, organization, or control of data. Characters may be letters, digits, punctuation marks, or other symbols. [T]

character class. (1) A named set of characters sharing an attribute associated with the name of the class. The classes and the characters that they contain are dependent on the value of the LC_CTYPE category in the current locale. [POSIX.2] (2) Ranges of characters that match a single character in the input stream. [OSF] (3) A set of characters enclosed in sequence, or square, brackets. [OSF]

character constant. (1) A constant with a character value. (2) In C language, a character or an escape sequence enclosed in single quotation marks. Some compilers allow more than one character or escape sequence in a character constant. [OSF]

character set. (1) A defined collection of characters. (2) All the valid characters for a programming language or for a computer system. (3) A group of characters used for a specific reason—for example, the set of characters a printer can print or a keyboard can support.

character special file. (1) A special file that provides access to an input or output device. The character interface is used for devices that do not use block I/O. (2) A file that refers to a device. One specific type of character special file is a terminal device file. Other character special files have no structure defined by POSIX.1, and their use is unspecified by POSIX.1. [POSIX.1] The only character special file supported by the OS/390 UNIX implementation is the pseudo-TTY, or pseudo-terminal.

character string. A sequence of consecutive characters.

character type. A data type that consists of alphanumeric characters.

checksum. The sum of a group of data associated with the group and used for checking purposes. [T]

circumflex. The character "^" [POSIX.2]

C language. A language used to develop software applications in compact, efficient code that can be run on different types of computers with minimal change.

class. (1) Any category to which things are arranged or defined. (2) A regular expression that matches any one of a set of characters.

client. A functional unit that receives shared services from a server.

CLIST. Command list.

code page. (1) A table showing codes assigned to character sets. (2) An assignment of graphic characters and control function meanings to all code points. (3) Arrays of code points representing characters that establish ordinal sequence (numeric order) of characters. [OSF] (4) A particular assignment of hexadecimal identifiers to graphic elements. (5) See also *code point, extended character*.

code point. A 1-byte code representing one of 256 potential characters.

collating element. The smallest entity used to determine the logical ordering of strings. A collating element consists of either a single character, or two or more characters collating as a single entity. The value of the LC_COLLATE category in the current locale determines the current set of collating elements. See also *collation sequence.* [POSIX.2]

collating sequence. Synonym for *collation sequence*.

collation sequence. (1) A specified arrangement used in sequencing. [I][A] (2) The sequence in which characters are ordered within the computer for sorting, combining or comparing. (3) The relative order of collating elements as determined by the setting of the LC_COLLATE category in the current locale. The character collation sequence defines the relative order of all collating elements such that each element occupies a unique position in the sequence. This is also the default sorting sequence. Multilevel sorting is accomplished by assigning elements one or more collation weights, up to the limit {COLL_WEIGHTS_MAX}. On each level, elements may be given the same weight or can be omitted from the sequence. Strings that collate equal using the first assigned weight (primary ordering) are then compared using the next assigned weight (secondary ordering), and so on. See also *equivalence class*. [POSIX.2] (4) Synonymous with collating sequence.

column. The vertical arrangement of characters or other expressions. [A] Contrast with *row.*

command. (1) A directive to the shell to perform a particular task. [POSIX.2] (2) A unit of input to an interactive program. The syntax of a command depends on the program processing it. A frequently used program is the shell, the input of which consists mostly of command lines. (3) A request to perform an operation or run a program. When parameters, arguments, flags, or other operands are associated with a command, the resulting character string is a single command. [OSF] (4) A request from a terminal for the performance of an operation or execution of a program. (5) A character string from a source external to a system that represents a request for system action. (6) One or more words that tell the system to perform

an operation or run a program. It consists of a name and, optionally, one or more operands. (7) The typed name and parameters associated with an action that can be performed by an application. A command is one form of action request. Users type in the command and enter it. The computer performs the action requested by the command name. (8) An instruction a user can type on the command line or in a CLIST. (9) A common action users request to interact with the command area.

command alias. An abbreviation of a long command line or a new name for a command. [OSF]

command history. A feature that stores commands and allows you to edit and reuse them. [OSF]

command interpreter. A program that reads the commands that you type in and then executes them. When you are typing commands into the computer, you are actually typing input to the command interpreter. The interpreter then decides how to perform the commands that you have typed. The shell is an example of a command interpreter. Synonymous with command language interpreter.

command line. (1) The area of the screen where commands are displayed as they are typed. [OSF] (2) The position after the arrow (===>) at the top or bottom of the panel where the user can type a command. (3) A unit of input to the shell, consisting of a program name and possibly command-line arguments, separated by white space and terminated by a newline.

command-line argument. A part of a command line, delimited by white space. Arguments are used to specify detailed behavior to a program. They are usually either command line options selecting variations in program operation, or pathnames of files to be processed.

command list (CLIST). (1) A data set in which commands and possibly subcommands and data are stored for subsequent execution. (2) A sequential list of commands, control statements, or both that is assigned a name and executed when that name is invoked.

command mode. (1) A state of a system or device in which the user can enter commands. (2) In an editing session, the mode wherein the editor is waiting for the user to enter a command.

command name. (1) The first term in a command. It requests a specific action, and it is usually followed by operands. (2) The first or principal term in a command. A command name does not include parameters, arguments, flags, or other operands. [OSF]

command processor. A program executed to perform an operation specified by a command.

command substitution. The ability to capture the output of any command as an argument to another command by placing that command line within grave accents ( ). The shell first runs the command or commands enclosed within the grave accents and then replaces the whole expression, including grave accents, with their output. This feature is often used in assignment statements.

Communications Server. A RISC System/6000 connected to one or more OS/390 systems, allowing terminals on asynchronous ports of the RISC System/6000 to operate as if they were connected directly to the OS/390 system. Synonymous with OCS.

compile. (1) To convert a computer program expressed in a problem-oriented language into a computer-oriented language. (2) To prepare a machine language program from a computer program written in another programming language by making use of the overall logic structure of the program, or generating more than one computer instruction for each symbolic statement, or both, as well as performing the function of an assembler. [A] (3) To convert a program written in a high-level programming language into an intermediate language, an assembler language, or a machine language. (4) The computer actions required to transform a source file into an executable object file. [OSF]

compiler. (1) A program that translates a source program into an executable program (an object program). (2) A program that translates a computer program expressed in a problem-oriented language into a machine language program to be executed at a later time. Synonymous with compiling program.

concatenate. (1) To link together. (2) To join two character strings.

controlling process. The session leader that established the connection to the controlling terminal. If the terminal subsequently ceases to be a controlling terminal for this session, the session leader shall cease to be the controlling process. [POSIX.1]

controlling terminal. (1) An active terminal at which a user is authorized to enter commands that affect system operation. The controlling terminal for any process normally is the active terminal from which the process group for that process was started. A terminal can have no more than one controlling process group and a process group can have no more than one controlling terminal. The controlling process group receives certain interrupt signals from the controlling terminal. [OSF] (2) A terminal that is associated with a session. Each session may have at most one controlling terminal associated with it, and a controlling terminal is associated with exactly one session. Certain input sequences from the controlling terminal cause signals to be sent to

all processes in the process group associated with the controlling terminal. [POSIX.1]

conversion. (1) The process of changing from one form of representation to another—for example, to change from decimal representation to binary representation; or to change from ASCII to EBCDIC. (2) A change in the type of value. For example, in some programming languages when you add values having different data types, the compiler converts both values to the same form before adding them. (3) In programming languages, the transformation between values that represent the same data item but belong to different data types. Information may be lost due to conversion since accuracy of data representation varies among different data types. [I] (4) The process of changing from one method of data processing to another or from one data processing system to another. (5) Sometimes referred to as translation.

conversion table. (1) A table used to replace one or more characters with alternative characters—for example, to convert characters representing an event to those representing a procedure call, characters of a code set to those of another code set, or characters representing a relocated address to those representing an absolute address. (2) A table that maps virtual addresses with real addresses. (3) A table that specifies the mapping of events or event sequences to procedure names. (4) Synonymous with translate table, translation table.

copy. (1) To read data from a source, leaving the source data unchanged, and to write the same data elsewhere. [I][A] (2) The action by which the user makes a whole or partial duplicate of an already existing data object. [OSF]

country-extended code page (CECP). An 8-bit code page that has a 93-character set on its nationally standardized code points but is extended to the multilingual character set for the national languages of some European countries.

couple data set. A data set that is created through the XCF couple data set format utility and, depending on its designated type, is shared by some or all of the MVS systems in a sysplex. See also *sysplex couple data set*.

coupling facility. A special logical partition that provides high-speed caching, list processing, and locking functions in a sysplex.

coupling services. In a sysplex, the functions of XCF that transfer data and status between members of a group residing on one or more MVS systems in the sysplex.

cross-system coupling facility (XCF). XCF is a component of the MVS that provides functions to support cooperation between authorized programs running within a sysplex.

current directory. The directory that is active and can be displayed with the **pwd** command. Synonymous with current working directory. [OSF]

current working directory. (1) The directory a user is working with. (2) Synonym for *current directory*. (3) Synonym for *working directory.*

cursor. (1) A movable spot of light on the screen of a display device, usually indicating where the next character is to be entered, replaced, or deleted. (2) A movable symbol (such as an underline) on a display that indicates to the user where the next typed character will be placed or where the next action will be directed. [OSF] (3) A marker that indicates the current data access location within a file. [OSF] (4) A symbol displayed in a panel and associated with an input device. The cursor indicates where input from the device will be placed.

D

data definition. A program statement that describes the features of, specifies relationships of, or establishes the context of, data. [A] A data definition can also provide an initial value. Definitions appear outside a function or at the beginning of a block statement.

data definition name (ddname). (1) The name of a data definition (DD) statement that corresponds to a data control block that contains the same name. (2) The symbolic representation for a name placed in the name field of a data definition (DD) statement.

data set. (1) The major unit of data storage and retrieval in the operating system, consisting of a collection of data in one of several prescribed arrangements and described by control information to which the system has access. (2) The major unit of data storage and retrieval in an MVS system. An HFS data set stores a file system.

data structure. The syntactic structure of symbolic expressions and their storage characteristics. [T]

ddname. Data definition name.

DD statement. Data definition statement.

debug. To detect, diagnose, and eliminate errors in programs. [T]

debugger. A device used to detect, trace, and eliminate errors in computer programs or software. [OSF]

decimal. Pertaining to a system of numbers to the base 10.

default. (1) A value, attribute or option that is assumed when no alternative is specified by the user. [OSF] (2) An alternative value, attribute, or option that the system supplies when the user does not specify a value.

default directory. The directory name supplied by the operating system if none is specified.

descriptor. A small, unsigned integer that a UNIX system uses to identify an object supported by the kernel. Descriptors can represent files, pipes, sockets, and other I/O streams. They are created, acted on, and deallocated by system calls specific to the object. [OSF]

DFSMS. Data Facility Storage Management Subsystem.

digit. (1) A graphic character that represents an integer—for example, one of the characters 0 to 9. (2) A symbol that represents one of the nonnegative integers smaller than the radix. For example, in decimal notation, a digit is one of the characters from 0 through 9. In hexadecimal notation, a digit is one of the characters from 0 through F. (3) Synonymous with numeric character.

directory. (1) A type of file containing the names and controlling information for other files or other directories. (2) A construct for organizing computer files. As files are analogous to folders that hold information, a directory is analogous to a drawer that can hold a number of folders. Directories can also contain subdirectories, which can contain subdirectories of their own. (3) A file that contains directory entries. No two directory entries in the same directory can have the same name. [POSIX.1] (4) A file that points to files and to other directories. (5) An index used by a control program to locate blocks of data that are stored in separate areas of a data set in direct access storage.

directory entry. An object that associates a filename with a file. Several directory entries can associate names with the same file. Synonymous with link. [POSIX.1]

dollar sign. The character “$”. POSIX permits the substitution of the “currency symbol” graphic defined in ISO/IEC 646 {1} for this symbol when the character set being used has substituted that graphic for the graphic $. The graphic symbol $ is always used in the POSIX standard, but not in any monetary sense. [POSIX.2]

dot. (1) A symbol (**.**) that indicates the current directory in a relative pathname. [OSF] (2) The filename consisting of a single dot character (**.**). This filename refers to the directory specified by its predecessor. See also *period.* [POSIX.1]

dot-dot. (1) A symbol (**..**) in a relative pathname that indicates the parent directory. [OSF] (2) The filename consisting solely of two dot characters (**..**). This filename refers to the parent directory of its predecessor directory. As a special case, in the root directory, dot-dot may refer to the root directory itself. [POSIX.1]

double-quote. The character " —also known as *quotation-mark*. [POSIX.2]

dump. (1) To write at a particular instant the contents of storage, or part of storage, onto another data medium for the purpose of safeguarding or debugging the data. [T] (2) To copy data in a readable format from central or auxiliary storage onto an external medium such as tape or printer. (3) To copy the contents of all or part of virtual storage for the purpose of collecting error information. (4) To copy all or part of the contents of an auxiliary storage device for later restoration of the data to an auxiliary storage device of the same type or another type.

dynamic. Occurring at the time of execution.

E

EBCDIC. Extended binary-coded decimal interchange code.

echo. (1) In data communication, a reflected signal on a communication channel. (2) In word processing, to print or display each character or line as it is keyed in.

edit. (1) To enter, modify, or delete data. (2) To add, change, delete, rearrange, or modify the form or format of data. [OSF] (3) To check the accuracy of information and to indicate if an error is found. [OSF]

edit mode. An entry mode in which a user can issue subcommands to enter, modify, delete, or rearrange data.

editor. See *editor program.* >

editor program. (1) A program used to enter and modify programs, text, and other types of documents and data. [OSF] (2) A computer program designed to perform such functions as rearrangement, modification, and deletion of data in accordance with prescribed rules. [A] (3) Contrast with *linkage editor.*

effective user ID. (1) The current user ID, but not necessarily the user's login ID. For example, a user logged in under a login ID may change to another user's ID. The ID to which the user changes becomes the effective user ID until the user switches back to the original login ID. [OSF] (2) An attribute of a process that is used in determining various permissions, including file access permissions. This value is subject

to change during the process lifetime. See also *user ID.* [POSIX.1]

element. (1) In a set, an object, entity, or concept having the properties that define a set. [I][A] Synonymous with member. (2) The smallest unit of data in a table or array. (3) The component of an array, subrange, enumeration, or set. (4) Any of the bits of a bit string, the octets of an octet string, or the octets by means of which the characters of a character string are represented. [OSF]

empty directory. A directory that contains, at most, directory entries for dot and dot-dot. [POSIX.1]

empty string. A character array whose first element is a null character. Synonymous with null string. [POSIX.1]

enter. (1) To send information to the computer by pressing the <Enter> key. (2) To type in information on a keyboard and press the <Enter> key to send the information to the computer. (3) To place on the line a message to be transmitted from a terminal to the computer. (4) A common action that submits panel information to the computer for processing; enter tells the computer to perform selected actions on specified objects.

Note: *Select* means to mark a choice on a panel; *enter* sends all designated choices to the computer for processing.

environment. (1) The settings for shell variables and paths set when the user logs in. These variables can be modified later by the user. [OSF] (2) A block of information passed ("exported") to a command when the command is invoked. This block contains a number of environment variables. The environment provides information that the program may use in its operation, in a form that relieves you of the need to specify it with every command. (3) The set of all factors that may affect how a program behaves. (4) A named collection of logical and physical resources used to support the operation of a function. (5) See also *environment variable*.

environment variable. (1) A name associated with a string of characters, made available to the programs that you run. Some environment variables used by the shell are **PATH**, **TMPDIR**, **COLUMNS**, and **LINES**. For example, the **TMPDIR** environment variable holds the name of a directory where shell commands are free to create temporary working files. (2) A variable that describes the operating environment of the process and typically includes information about the home directory, command search path, the terminal in use, and the current time zone (the **HOME**, **PATH**, **TERM**, and **TZ** variables, respectively). (3) An environment variable the value of which is the name of a file that contains shell commands to customize a shell environment. When a user first runs the shell, the shell executes his or her profile. Then the shell runs the commands in the **ENV** file. (4) A variable included in the current software environment that is available to any called program that requests it.

equivalence class. (1) A grouping of characters or character strings that are considered equal for purposes of collation. For example, many languages place an uppercase character in the same equivalence class as its lowercase form, but some languages distinguish between accented and unaccented character forms for the purpose of collation. (2) A set of collating elements that collate equal on a particular weight level (primary, secondary, or following assigned collation weights). Elements in an equivalence class are typically elements that naturally group together, such as all accented letters based on the same base letter. The order of elements within an equivalence class is determined by the weights assigned on a subsequent level, if any. In regular expressions and pattern matching, only primary equivalence classes are recognized. [POSIX.2]

ERE. Extended regular expression.

escape sequence. (1) A character that is preceded by a \ (backslash) and is interpreted to have a special meaning to the shell. (2) A sequence sent to a terminal to perform actions such as moving the cursor, changing from normal to reverse video, and clearing the screen.

exec. To overlay the current process with another executable program. See also *fork*.

executable. See *executable file, executable program, executable statement*.

executable file. (1) A file suitable for execution. An executable file may be a program that has been compiled and link-edited, or it may be a shell script. (2) A file that contains programs or commands that perform operations on actions to be taken. (3) A regular file acceptable as a new process image file by the equivalent of the POSIX.1 **exec** family of functions, and thus usable as one form of a utility. The standard utilities described in POSIX.1 as compilers can produce executable files, but other unspecified methods of producing executable files may also be provided. The internal format of an executable file is unspecified, but a conforming application shall not assume an executable file is a text file. [POSIX.2] (4) A REXX exec. (5) See also *executable program*.

executable program. (1) A program in a form suitable for execution by a computer. The program can be an application or a shell script. An executable program is equivalent to an MVS load module. (2) A program that has been link-edited and can therefore be run in a processor. (3) A program that can be executed as a

self-contained procedure. It consists of a main program and, optionally, one or more subprograms.

executable statement. A statement that causes an action to be taken by the program. For example, to calculate, to test conditions, or to alter normal sequential execution. [OSF]

execute. (1) To perform the actions specified by a program or a portion of a program. [T] (2) To perform the actions described in POSIX.2 3.9.1.1. See also *invoke.* [POSIX.2] (3) Synonym for *run*.

existing file. A file that has been defined and, conceptually, resides on the storage medium. [OSF]

exponent. A number indicating the power to which another number (the base) is to be raised.

exponentiation. The operation in which a value is raised to a power. >

expression. (1) A representation of a value—for example, variables and constants appearing alone or in combination with operators. (2) In programming languages, a language construct for computing a value from one or more operands; for example, literals, identifiers, array references, and function calls. [I] (3) A configuration of signs. [A] (4) A group of constants or variables separated by operators that yields a single value. An expression can be arithmetic, relational, logical, or a character string. (5) An operand or a collection of operators and operands that yields a single result. (6) See *arithmetic expression.*

expression statement. An expression that ends with a **;** (semicolon). You can use an expression statement to assign the value of an expression to a variable or to call a function.

extended binary-coded decimal interchange code (EBCDIC). A coded character set consisting of 8-bit coded characters. [A] See also *American National Standard Code for Information Interchange (ASCII).*

extended character. A character other than a 7-bit ASCII character. An extended character can be a 1-byte code point with the eighth bit set (ordinal 128–255).

extended regular expression (ERE). A pattern (sequence of characters or symbols) constructed according to the rules defined in POSIX.2 2.8.4. [POSIX.2]

F

field. (1) An area in a record or panel used to contain a particular category of data. [OSF] (2) The smallest identifiable component of a record. (3) In a record, a specified area used for a particular category of data.

FIFO special file. (1) A type of file with the property that data written to such a file is read on a first-in-first-out basis. [POSIX.1] (2) A named permanent pipe that allows two unrelated processes to exchange information through a pipe connection. Synonymous with named pipe.

file. (1) A set of related records treated as a unit. (2) A sequence of records. If the file is located in internal storage, it is an internal file; if it is on an input/output device, it is an external file. [OSF] (3) A collection of related data that is stored and retrieved by an assigned name. [OSF] (4) Linear data that can be opened, written, read, and closed. A file can also contain information about the file, such as authorization information. The name used to obtain a file includes the directories in the path to the file. (5) Strings of characters with no additional structure. Structure is assumed only by the processing programs. Files can be located relative to the current directory or by an absolute pathname. (6) An object that can be written to, or read from, or both. A file has certain attributes, including access permissions and type. File types include regular file, character special file, block special file, FIFO special file, and directory. Other types of files may be defined by the implementation. [POSIX.1] In the OS/390 implementation, the file system does not support block special files, but it does support symbolic link files. (7) A collection of information or data that is organized by some method (relative, indexed, or serial, for example) and stored on a device such as a disk.

file descriptor. (1) A small unsigned integer that a UNIX system uses to identify a file. A file descriptor is created by a process through issuing an **open** system call for the filename. A file descriptor ceases to exist when it is no longer held by any process. [OSF] (2) A per-process unique, nonnegative integer used to identify an open file for the purpose of file access. [POSIX.1] (3) A small positive number used to identify an open file in I/O operations. By convention, certain file descriptors are used for the same purpose by all programs. (4) See also *standard error (stderr), standard input (stdin), standard output (stdout)*.

file mode. An object containing the file permission bits and other characteristics of a file. [POSIX.1]

filename. (1) A name assigned or declared for a file. (2) The name used by a program to identify a file. (3) A name consisting of 1 to {NAME_MAX} bytes used to name a file. The characters composing the name

may be selected from the set of all character values excluding the slash character and the null character. The filenames dot and dot-dot have special meaning. Synonymous with pathname component. See also *dot, dot-dot.* [POSIX.1]

file owner. The user who has the highest level of access authority to a file, as defined by the file.

file system. (1) A collection of files and directories. (2) The collection of files and file management structures on a physical or logical mass storage device, such as a disk or disk partition. A single device can contain several file systems. (3) A mountable subtree of the directory hierarchy. [OSF] (4) A collection of files and certain of their attributes. A file system provides a name space for file serial numbers referring to those files. [POSIX.1]

file system owner. The system which coordinates sysplex activity for a particular file system. In many cases, the file system owner is the one system in the sysplex which will have a physical connection to the file system, and as such will coordinate I/O for that file system for all users in the sysplex. In addition, the file system owner will coordinate changes to the status of a file system, such as when it is moved or unmounted.

file type. One of the five possible types of files: ordinary file, directory, block device, character device, and first-in-first-out (FIFO or named pipe). See also *file.*

first-in-first-out (FIFO). A queuing technique in which the next item to be retrieved is the item that has been in the queue for the longest time. [A]

fixed-length record. A record having the same length as all other records with which it is logically or physically associated. Contrast with *variable-length record.*

flag. (1) A modifier that appears on a command line with the command name that defines the action of the command. [OSF] (2) An indicator or parameter that shows the setting of a switch. [OSF] (3) A variable indicating that a certain condition holds. [T] (4) A character that signals the occurrence of some condition, such as the end of a word. [A] (5) An internal indicator that describes a condition to the CPU. [OSF] Synonymous with condition code.

floating point. Pertaining to a method of encoding real numbers within the limits of finite precision available on computers. Floating point represents real numbers (that is, values with fractions or decimals) in 32 bits or 64 bits. Floating-point representation is useful to describe very small or very large numbers.

fold. To translate the lowercase characters of a character string into uppercase.

foreground. (1) A process that an interactive user starts running and that can interact with the user. An interactive user can move a foreground process to the background. (2) A mode of program execution in which the shell waits for the program specified on the command line to complete before responding to user input. (3) In multiprogramming, the environment in which high-priority programs are executed. (4) The interactive execution of programs and services. (5) The environment in which interactive programs are executed. Interactive processors reside in the foreground. (6) Contrast with *background.*

foreground process. (1) A process that is a member of a foreground process group. [POSIX.1] (2) A process that must run to completion before another command is issued to the shell. The foreground process is in the foreground process group.

foreground process group. A process group whose member processes have certain privileges, denied to processes in background process groups, when accessing their controlling terminal. Each session that has established a connection with a controlling terminal has exactly one process group of the session as the foreground process group of that controlling terminal. [POSIX.1] The foreground process group receives the signals generated by the terminal.

format. (1) A defined arrangement of such things as characters, fields, and lines, usually used for displays, printouts, or files. [OSF] (2) The pattern that determines how data is recorded. [OSF] (3) To arrange such things as characters, fields, and lines. (4) In programming languages, a language construct that specifies the representation, in character form, of data objects in a file. [I]

formatted file. A file displayed and arranged with particular characteristics, such as line spacing, headings, and number of characters and lines per page. [OSF] Contrast with *unformatted file*.

form-feed. A character that shall cause printing to start on the next page of an output device. The <form-feed> is the character designated by the "\f" in the C language binding. If <form-feed> is not the first character on an output line, the result is unspecified. It is unspecified whether this character is the exact sequence transmitted to an output device by the system to accomplish the movement to the next page. [POSIX.2] >

forward. A scrolling action that displays information below that currently visible in a panel body area.

full pathname. Synonym for *absolute pathname*.

function. (1) A specific purpose of an entity, or its characteristic action. [A] (2) A set of commonly used

computations that always returns to its invoker. (3) A data type that contains executable code and returns a single value to the calling function. [OSF] (4) A programming language construct modeled after the mathematical concept of function. A function encapsulates some behavior. It is given some arguments as input, performs some processing, and returns some results. [OSF] (5) A subroutine that returns the value of a single variable and that usually has a single exit, such as subroutines that compute mathematical functions. [T]

function call. An expression that moves the path of execution from the current function to a specified function and evaluates to the return value provided by the called function. A function call contains the name of the function to which control moves and a parenthesized list of arguments.

function key. (1) A key that causes a specified sequence of operations to be performed when it is pressed. Generally used to refer to keys labeled <F*n*>, for example,<F1>. (2) A key that requests actions but does not display or print characters. This definition includes a key that normally produces a printed character, but produces a function instead when used with the code key. [OSF]

G

GID. Group ID.

global. (1) In programming languages, pertaining to the relationship between a language object and a block in which the language object has a scope extending beyond that block but contained within an encompassing block. [I] (2) Pertaining to information defined in one subdivision of a computer program and used in at least one other subdivision of the program. [OSF] (3) Pertaining to information available to more than one program or subroutine. >

graphic character. A character that can be displayed or printed.

group. (1) A collection of users who can share access authorities for protected resources. [OSF] (2) A list of names that are known together by a single name. (3) A set of related records that have the same value for a particular field in all records. (4) A series of records logically joined together.

group ID (GID). (1) A unique number assigned to a group of related users. The GID can often be substituted in commands that take a group name as an argument. (2) A nonnegative integer, which can be contained in an object of type *gid_t,* that is used to identify a group of system users. Each system user is a member of at least one group. When the identity of a group is associated with a process, a group ID value is referred to as a real group ID, an effective group ID, one of the (optional) supplementary group IDs, or an (optional) saved set-group-ID. [POSIX.1] (3) Synonymous with group number.

group name. A name that uniquely identifies a group of users to the system.

H

hard link. (1) A mechanism that allows the **ln** command to assign more than one name to a file. Both the new name and the file being linked must be in the same file system. [OSF] (2) The relationship between two directory entries that represent the same file; the result of an execution of the **ln** utility or the POSIX.1 **link()** function. [POSIX.2]

hexadecimal. (1) Pertaining to a selection, choice, or condition that has sixteen possible different values or states. [I] (2) Pertaining to a fixed-radix numeration system, with radix of sixteen. [I] (3) Pertaining to a numbering system with base of sixteen; valid numbers use the digits 0 through 9 and characters A through F, where A represents 10 and F represents 15.

HFS data set. A hierarchical file system data set, which is used to store, and is essentially identified with, a file system. >

hierarchical file. See *file*.

high-order. Most significant; leftmost. [OSF]

history file. A file in which a record is kept of shell commands that are executed. The default history file is **.sh_history**. >

home directory. (1) The current directory associated with the user at the time of login. [POSIX.2] (2) A directory associated with an individual user. (3) The user's current directory on login or after issuing the **cd** command with no argument.

I

IF expression. An expression in a procedure that tests for a condition. The action performed by the procedure depends on the result of the test.

if statement. (1) A C language conditional statement that contains the word **if** followed by an expression in parentheses (the condition), a statement (the action), and an optional **else** clause (the alternative action). (2) A conditional statement that specifies a condition to be tested and the action to be taken if the condition is satisfied. [T]

index. (1) A list of the contents of a file or of a document, together with keys or references for locating the contents. [I][A] (2) A table used to locate records in an indexed sequential data set or an indexed file. (3) A table containing the key value and location of each record in an indexed file. [OSF] (4) A computer storage position or register whose contents identify a particular element in a set of elements. [OSF]

inode. The internal structure that describes the individual files in the operating system; there is one inode for each file. An inode contains the node, type, owner, access times, number of links, and location of a file. A table of inodes is stored near the beginning of a file system. See also *vnode*.

inode number. A number specifying a particular inode file in the file system. Synonymous with inumber. See also *inode*.

input redirection. The specification of an input source other than standard input/output.

integer. A positive or negative whole number or zero. [OSF]

interactive. (1) Pertaining to an application in which each entry calls forth a response from a system or program. The shell commands can be used interactively. (2) Pertaining to an activity that involves requests and replies, such as between a system user and a program or between two programs. [OSF] (3) Pertaining to a program or system that alternately accepts input and then responds. An interactive system is conversational; that is, a continuous dialog exists between user and system. (4) Pertaining to the exchange of information between a user and a computer.

internationalization. (1) The process of generalizing programs or systems so that they can handle a variety of languages, character sets, and national customs. [OSF] (2) The process of designing and developing a product with a set of features, functions, and options intended to facilitate the adaptation of the product to satisfy a variety of cultural environments. [POSIX.0]

interrupt. (1) A suspension of a process, such as execution of a computer program caused by an external event, and performed in such a way that the process can be resumed. [A] (2) A signal sent by an I/O device to the processor when an error has occurred or when assistance is needed to complete I/O. An interrupt usually suspends execution of the currently executing program. [OSF] (3) In data communication, to take an action at a receiving station that causes the sending station to end a transmission. (4) To temporarily stop a process. (5) See also *signal*.

invoke. (1) To start a command, procedure, or program. (2) To perform the actions described in POSIX.2 3.9.1.1, except that searching for shell functions and special built-ins is suppressed. See also *execute.* [POSIX.2]

J

job control. (1) Facilities for monitoring and accessing background processes. [OSF] (2) A facility that allows users to selectively stop (suspend) the execution of processes and continue (resume) their execution at a later point. The user typically employs this facility via the interactive interface jointly supplied by the terminal I/O driver and a command interpreter. Conforming implementations may optionally support job control facilities; the presence of this option is indicated to the application at compile time or run time by the definition of the [_POSIX_JOB_CONTROL] symbol. [POSIX.1]

justified. Adjacent to the indicated margin. In nonprogrammable terminals, if attribute bytes are required, left-justified information appears one character position to the right of the left margin with an unseen attribute byte in the first position. See also *justify, left-justify, right-justify.*

justify. (1) To control the printing positions of characters on a page so that both the left-hand and right-hand margins of the printing are regular. [I][A] (2) To shift the contents of a register or field so that the significant character at the specified end of the data is at the specified position. [T] (3) To align characters horizontally or vertically to fit the positioning constraints of a required format. [A] (4) To print a document with even right margins, even left margins, or both. (5) See *justified, left-justify, right-justify.*

K

kill. An operating system command that stops a process. [OSF]

KornShell. A command interpreter developed on UNIX, which forms the basis for the OS/390 shell.

L

left-justify. (1) To shift the contents of a register or field so that the character at the left-hand end of the data is at a specific position. [T] (2) To control the printing positions of characters on a page so that the left-hand margin of the printing is regular. [I][A]

library. (1) A collection of functions, calls, subroutines, or other data. (2) A named area on disk that can contain programs and related information (not files). A library consists of different sections, called library

members. (3) A data file that contains copies of a number of individual files and control information that allows them to be accessed individually. (4) A collection of related files. For example, one line of an invoice may form an item, a complete invoice may form a file, the collection of inventory control files may form a library, and the libraries used by an organization are known as its data bank. (5) A repository for demountable recorded media, such as magnetic disk packs and magnetic tapes. [A] (6) The set of publications for a product.

line. (1) A string of characters accepted by a system as a single block of input from a terminal, such as all characters entered before a carriage return. (2) A sequence of text consisting of zero or more non-<newline> characters plus a terminating <newline> character. [POSIX.2] (3) In terminal-oriented programs, a stream of bytes terminated by <newline>. (4) A horizontal display on a screen. [OSF]

line editor. An editor that displays data one line at a time and that allows data to be accessed and modified only by entering commands. [OSF]

link. (1) In the file system, a connection between an inode and one or more file names associated with it. (2) Synonym for *directory entry.* (3) In data communication, a transmission medium and data link control component that together transmit data between adjacent nodes. [OSF] (4) In programming, the part of a program that passes control and parameters between separate portions of the computer program. (5) To interconnect items of data or portions of one or more computer programs: for example, the linking of object programs by a linkage editor or linking data items by pointers. [T] (6) In SNA, the combination of the link connection and the link stations joining network nodes—for example, a System/370 channel and its associated protocols in a serial-by-bit connection under the control of Synchronous Data Link Control (SDLC).

linkage editor. (1) A computer program for creating load modules from one or more object modules or load modules by resolving cross-references among the modules and, if necessary, adjusting addresses. [T] (2) A program that resolves cross-references between separately compiled object modules and then assigns final addresses to create a single relocatable load module. If a single object module is linked, the linkage editor simply makes it relocatable. [OSF] (3) Synonymous with binder.

link-edit. To create a loadable computer program by means of a linkage editor.

list. (1) A sequence of one or more pipelines. (2) A data object consisting of a collection of related records.

literal. (1) A symbol or a quantity in a source program that is itself data, rather than a reference to data. (2) In programming languages, a unit that directly represents a value. For example, 14 represents the integer 14.

local. (1) Pertaining to a device, file, or system that is accessed directly from your system, without the use of a communication line. (2) Pertaining to that which is defined and used only in one subdivision of a computer program. [T]

locale. (1) A description of a cultural environment. [POSIX.0] (2) The definition of the subset of a user's environment that depends on language and cultural conventions. [POSIX.2] (3) A tuple generally consisting of a language, territory, and codeset specification and used in internalization configuration. [OSF]

logarithm. A mathematical operation related to the base of a numbering system. [OSF]

logger. (1) A functional unit that records events and physical conditions, usually with respect to time. [I][A] (2) A program that enables a user entity to log in (for example, identify itself, its purpose, and time of entry) and log out with the corresponding data. This enables the appropriate accounting procedures to be carried out in accordance with the operating system.

logical operator. A symbol that represents an operation, such as AND, OR, or NOT, on logical expressions. [OSF]

logically partitioned (LPAR) mode. A central processor complex (CPC) power-on reset mode that enables use of the PR/SM feature and allows an operator to allocate CPC hardware resources (including central processors, central storage, expanded storage, and channel paths) among logical partitions.

log in. (1) In UNIX systems, to gain access to a computer system by entering identification information at the workstation. (2) To begin a session at a workstation. (3) See also *log on,* the MVS term.

login. (1) In UNIX systems, the act of gaining access to a computer system by entering identification and authentication information at the workstation. (2) The unspecified activity by which a user gains access to the system. Each login shall be associated with exactly one login name. [POSIX.1] In the OS/390 implementation, a user gains interactive access to the shell by first logging on to a TSO/E user ID through existing documented MVS externals. Once logged on to TSO/E, a user can execute the shell by entering the OMVS command.

login name. (1) A string of characters that uniquely identifies a user to the system. (2) A user name that is associated with a login. [POSIX.1]

log on. (1) In MVS TSO/E, to begin a session. (2) See also *log in,* the UNIX term.

low-order. Least significant; rightmost. For example, in a 32-bit register (0–31), bit 31 is the low-order bit. [OSF]

M

mask. A pattern of characters that controls the keeping, deleting, or testing of portions of another pattern of characters. [I][A]

member. (1) A data object in a structure, a union, or a library. (2) A partition of a partitioned data set. (3) A part of a partitioned data set that can be used independently of other members of the data set. (4) Synonym for *element*.

metacharacter. (1) A character used to specify another character or series of characters. [OSF] (2) A character that may have a special meaning in a regular expression. You can usually use a backslash to remove the special meaning.

mode. (1) A method of operation. (2) A method of operation, frequently used in UNIX to refer to read, write, run, or search permissions of a file or directory. [OSF] (3) A collection of attributes that specifies a file's type and its access permissions. [POSIX.1]

mount. (1) To make a file system accessible. [OSF] (2) To logically mount a file system in another file system with the TSO/E command MOUNT. The mount point is in a directory. (3) The action taken by an NFS client to use file systems that an NFS server has made available. [OSF]

mount point. (1) The pathname of the directory on which the file system is mounted. (2) Either the root directory or a directory for which the *st_rdev* field of the POSIX.1 *struct stat* differs from that of its parent directory. [POSIX.2]

N

named pipe. Synonym for *FIFO special file*.

newline. (1) A cursor-movement function that moves the cursor to the first entry field on the next line that contains an entry field. (2) The line terminator in text files and keyboard input. On the keyboard, this is generated by the <Enter> key. (3) See also *newline character*.

newline character (NL or <newline>). (1) A control character that causes the print or display position to move to the first position on the next line. This character is often represented by "\n". [OSF] (2) A character that shall cause printing to start at the beginning of the next line. The <newline> shall be the character designated by the "\n" in the C language binding. It is unspecified whether this character is the exact sequence transmitted to an output device by the system to accomplish the movement to the next line. [POSIX.2]

null character. A character with all bits set to zero [POSIX.2].

null string. (1) A string containing no element. [T] (2) The notion of a string depleted of its entities, or the notion of a string prior to establishing its entities. [A] (3) Synonymous with empty string, null character string.

null value. (1) A parameter position for which no value is specified. (2) When used with a relational data base, an indication that no data value has been assigned to the intersection of a row and a column in a relational table.

numeric. Pertaining to any of the digits 0 through 9.

O

object. (1) An autonomous data item that is an abstraction of some idea or process. Objects hide their implementation to the manipulating program or function by providing methods. (2) One of a set of items that actions work on.

object code. (1) Output from a compiler or assembler which is itself executable machine code or is suitable for processing to produce executable machine code. [A] Contrast with *source code*. (2) Machine-executable instructions, usually generated by a compiler from source code written in a higher level language (such as C language). For programs that must be linked, object code consists of relocatable machine code. [OSF]

object module. (1) A set of instructions in machine language produced by a compiler from a source program. (2) A portion of an object program suitable as input to a linkage editor. [T]

octal. (1) A base-eight numbering system. [OSF] (2) Pertaining to a fixed-radix numeration having a radix of eight. [I][A]

OCS. Outboard Communications Server. See *Communications Server*.

OS/390 UNIX System Services. OS/390 services that support an environment within which operating systems, servers, distributed systems, and workstations share common interfaces. OS/390 UNIX System Services supports standard application development across multi-vendor systems. It is required if you want to create and

use applications that conform to the POSIX standard. OS/390 UNIX System Services combines the personal power of the workstation, the flexibility of open systems, and the strength of MVS. It supports and fosters a superenvironment of larger operating systems or servers and of distributed systems and workstations that share common interfaces. Users can switch back and forth between the traditional TSO/E interface and the shell interface. UNIX-skilled users can interact with the system, using a familiar set of standard commands and utilities. MVS-skilled users can interact with the system, using familiar TSO/E commands and interactive menus to create and manage hierarchical file system files and to copy data back and forth between MVS data sets and files. Application programmers and users have both sets of interfaces to choose from and, by making appropriate tradeoffs, can choose to mix these interfaces.

open file. A file that is currently associated with a file descriptor. [POSIX.1]

open system. (1) A system whose characteristics comply with standards made available throughout the industry and that therefore can be connected to other systems complying with the same standards. [T] (2) A comprehensive and consistent set of international information technology standards and functional standards profiles that specify interfaces, services, and supporting formats to accomplish interoperability and portability of applications, data and people. [T] (3) In computer security, a system in which resources that are not defined to the system are not protected. (4) A system that implements sufficient open specifications for interfaces, services, and supporting formats to enable properly engineered applications software to be ported with minimal changes across a wide range of systems, to interoperate with other applications on local and remote systems, and to interact with users in a style that facilitates user portability. [POSIX.0] (5) A system that uses standard-based communication to work with other such systems to accomplish business tasks. (6) A system that is part of other such systems in an enterprise in which the following criteria apply:

- All data within each system is accessible by any user or application (within the limits of some security feature).
- Any application within the enterprise can be run from a user's terminal without the user or the program that calls the application needing to specify where the application is or how it works.
- Software or hardware system configurations can be acquired from more than a single vendor. Products from one vendor can communicate with products with products from other vendors.
- Applications are portable from system to system, independently of the differences in hardware.
- Data is portable from system to system.
- New applications and systems integrate well with the current applications and systems.
- All systems conform to standards formalized by international or national standards committees or consortia, rather than to standards developed and promulgated by hardware or software vendors.
- All systems use standards-based UNIX.

operand. (1) An argument to a command that is generally used as an object supplying information to a utility necessary to complete its processing. Operands generally follow the options in a command line. [POSIX.2] (2) An instruction field that represents data (or the location of data) to be manipulated or operated upon. Not all instructions require an operand field. [OSF] (3) An identifier, constant, or expression that is grouped with an operator. [OSF] (4) An entity on which an operation is performed. [T][A] (5) Information entered with a command name that defines the data on which a command processor operates and that controls the execution of the command processor. (6) Information entered with a macro instruction.

operation. (1) A specific action (such as add, multiply, or shift) that the computer performs when requested. [OSF] (2) A set of step-by-step actions specified by a procedure, function, or routine. [OSF] (3) The task performed by a given routine or procedure. [OSF]

operator. (1) A person who operates a device. (2) The person responsible for monitoring the system console and controlling the system. (3) A symbol (such as +, –, or *) that represents an operation (in this case, addition, subtraction, multiplication). [OSF] (4) A symbol that represents an operation to be done. (5) See *logical operator.*

option. (1) A specification in a statement that can influence the execution of the statement. (2) An argument to a command that is generally used to specify changes in the *utility*'s default behavior. [POSIX.2] (3) Synonym for *process option.*

Outboard Communication Server (OCS). See *Communications Server*.

output. (1) The result of processing data. [OSF] (2) Pertaining to a functional unit or channel involved in an output process, or to the data or involved in such a process. (3) Data transferred from storage to an output device.

output file. (1) A file that a program opens so that it can write to that file. (2) A file that contains the results of processing.

output redirection. The specification of an output destination other than the standard one. [OSF]

output stream. (1) Diagnostic messages and other output data issued by an operating system or a processing program on output devices especially activated

for this purpose by the operator. [OSF] (2) Messages and other output data that an operating system or a processing program displays on output devices. <

override. (1) A parameter or value that replaces a previous parameter or value. (2) To replace a parameter or value.

overwrite. To write into an area of storage, thereby destroying the data previously stored in the same area. [T]

owner. The user who has the highest level of access authority to a data object or action, as defined by the object or action. [OSF]

P

P1003.2. POSIX.2

parallel processing. The simultaneous processing of units of work by many servers. The units of work can be either transactions or subdivisions of large units of work (batch).

Parallel Sysplex. A sysplex that uses one or more coupling facilities.

parameter. (1) A variable that is given a constant value for a specified application and that may denote the application. [I][A] (2) A name in a procedure that is used to refer to an argument passed to that procedure. (3) An argument the user supplies to a command or function. [OSF] (4) Data passed between programs or procedures. (5) See also *operand*.

parent directory. (1) The directory one level above the current directory. (2) When discussing a given directory, the directory that both contains a directory entry for the given directory and is represented by the pathname dot-dot in the given directory. [POSIX.1] (3) When discussing other types of files, a directory containing a directory entry for the file under discussion. [POSIX.1]

parent process. A process created to carry out a program. The parent process in turn creates child processes to execute requests. See also *parent process ID, process.*

parent process ID. An attribute of a new process after it is created by a currently active process. The parent process ID of a process is the process ID of its creator, for the lifetime of the creator. After the creator's lifetime has ended, the parent process ID is the process ID of an implementation-defined system process. [POSIX.1] In the OS/390 implementation, the parent process ID of the children of an ended process is set to the process ID of the INIT process, or 1.

parse. To analyze the operands entered with a command and build a parameter list for the command processor from the information.

partitioned data set (PDS). A data set in direct access storage that is divided into partitions, called members, each of which can contain a program, part of a program, or data. Synonymous with program library.

pathname. (1) A filename specifying all directories leading to the file. (2) See also *absolute pathname, relative pathname*. (3) A filename specifying all directories leading to a file plus the filename itself. (4) A string that is used to identify a file. A pathname consists of, at most, [PATH_MAX] bytes, including the terminating null character. It has an optional beginning slash, followed by zero or more filenames separated by slashes. If the pathname refers to a directory, it may also have one or more trailing slashes. Multiple successive slashes are considered to be the same as one slash. A pathname that begins with two successive slashes may be interpreted in an implementation-defined manner, although more than two leading slashes shall be treated as a single slash. [POSIX.1] In the OS/390 implementation, the C/370 functions **fopen()**, **freopen()**, **remove()**, and **rename()** interpret names with exactly two leading slashes, no leading blanks or other characters, and the third character not a slash to mean that the rest of the name refers to a traditional MVS data set.

pattern. (1) A regular expression or series of regular expressions that define the search pattern. (2) A sequence of characters used either with regular expression notation or for pathname expansion, as a means of selecting various character strings or pathnames, respectively. The syntaxes of the two patterns are similar, but not identical; the standard always indicates the type of pattern being referred to in the immediate context of the use of the term. [POSIX.2] (3) A sequence of characters used by commands that search for strings. Some characters have special meanings in patterns; for example, $ stands for the end of a line and `abc$` refers to the sequence *abc* appearing at the end of a line. Some patterns can be matched by many different strings.

pattern matching. (1) The identifying of one of a predetermined set of items which has the closest resemblance to a given object, by comparing its coded representation against the representation of all the items. [T] (2) Specifying a pattern of characters that the system should find. [OSF] (3) The process of searching for strings of characters that conform to the pattern of characters in a regular expression.

PDS. Partitioned data set.

period. The character "**.**" The term *period* is contrasted against *dot,* which is used to describe a specific directory entry. See also *dot.* [POSIX.2]

permission. (1) A code that determine how the file can be used by any users who work on the system. [OSF] (2) The modes of access to a protected object. [OSF]

PID. Process ID.

pipe. (1) An interprocess communication mechanism that connects an output file descriptor to an input file descriptor. Usually the standard output of one process is connected to the standard input of another, forming a pipeline. (2) A sequence of one or more commands in FIFO order. The output of one command becomes the input to the next command. A pipe usually contains several filters. Pipes allow related or unrelated processes to read and write to each other as if they were files; they allow unidirectional communication from one process to another. OS/390 UNIX System Services treats pipes as though they were files. A named pipe has a directory name and is accessed by a pathname. An unnamed pipes must be used between a parent process and a child process. (3) An object accessed by one of the pair of file descriptors created by the **pipe()** function. Once created, the file descriptors can be used to manipulate it, and it behaves identically to a FIFO special file when accessed in this way. It has no name in the file hierarchy. [POSIX.1] (4) To direct data so that the output from one process becomes the input to another process. (5) An I/O stream that has a descriptor and can be used in unidirectional communications between related processes. [OSF]

pipeline. (1) A chain of two or more processes connected by pipes. Each process in the chain acts as a filter, reading data from the standard input, performing some transformation, and writing the results to the standard output. (2) A direct, one-way connection between two or more processes. (3) A serial arrangement of processors or a serial arrangement of registers within a processor. Each processor or register performs part of a task and passes results to the next processor. Several parts of different tasks can be performed at the same time. (4) To perform processes in a series. (5) For increased processing speed, to start execution of an instruction sequence before the previous instruction sequence is completed.

pointer. (1) In the C language, a variable that holds the address of a data object or a function. (2) A physical or symbolic identifier of a unique target. (3) An identifier that indicates the location of an item of data. [A] (4) A data element that indicates the location of another data element. [T] (5) In computer graphics, a manually operated functional unit used to specify an addressable point. A pointer may be used to conduct interactive graphic operations, such as selection of one member of a predetermined set of display elements, or indication of a position on a display space while generating coordinate data. [T]

portability. (1) The ability to run a program on more than one computer without modifying it. (2) The ability to use applications, data sets, or files with different operating systems. (3) The ease with which software can be transferred from one information system to another. [POSIX.0]

portable character set. The set of characters described in POSIX.2 2.4 that is supported on all conforming systems. See also *portable filename character set.* [POSIX.2]

portable filename character set. The set of characters from which portable filenames are constructed. For a filename to be portable across conforming implementations of POSIX.1, it shall consist only of the uppercase and lowercase characters of the alphabet (*A* through *Z* and *a* through *z*), the digits *0* through *9,* the period (**.**), the underscore (_), and the hyphen (**-**). The hyphen shall not be used as the first character of a portable filename. Uppercase and lowercase letters shall retain their unique identities between conforming implementations. In the case of a portable pathname, the slash character may also be used. [POSIX.1]

position. (1) Any location in a string that may be occupied by an element and that is identified by a serial number. [T] (2) The location of a character in a series, as in a record, a displayed message, or a computer printout.

positional parameter. (1) A shell facility that assigns values from the command line to variables in a program. [OSF] (2) A parameter that must appear in a specified location relative to other positional parameters. (3) One of the command-line arguments to a shell file. Positional parameters are referenced by `$1`, `$2`, and so on.

POSIX. Portable Operating System Interface for Computer Environments, an interface standard governed by the IEEE and based on UNIX. POSIX is not a product. Rather, it is an evolving family of standards describing a wide spectrum of operating system components ranging from C language and shell interfaces to system administration.

precedence. (1) The priority system for grouping different types of operators with their operands. [OSF] (2) In programming languages, an order relation defining the sequence of the application of operators within an expression. [OSF]

primary. An irreducible unit of data. For example, a single constant, variable, or array element.

print file. A file that is created for the purpose of printing data.

process. (1) A function being performed or waiting to be performed. (2) An executing function, or one waiting to execute. (3) A function, created by a **fork()** request, with three logical sections:

- Text, which is the function's instructions.
- Data, which the instructions use but do not change.
- Stack, which is a push-down, pop-up save area of the dynamic data that the function operates upon.

The three types of processes are:

- User processes, which are associated with a user at a workstation
- Daemon processes, which do systemwide functions in user mode, such as printer spooling
- Kernel processes, which do systemwide functions in kernel mode, such as paging

A process can run in a user address space, a forked address space, or a kernel address space. In an MVS system, a process is handled like a task. See also *task*. (4) An address space and one or more threads of control that execute within that address space, and their required system resources. [POSIX.0] (5) An address space and single thread of control that executes within that address space, and its required system resources. A process is created by another process issuing the **fork()** function. The process that issues **fork()** is known as the parent process, and the new process created by the **fork()** is known as the child process. [POSIX.1] (6) A sequence of actions required to produce a desired result. [OSF] (7) An entity receiving a portion of the processor's time for executing a program. [OSF] (8) An activity within the system that is started by a command, a shell program, or another process. Any running program is a process. (9) A unique, finite course of events defined by its purpose or by its effect, achieved under given conditions. (10) Any operation or combination of operations on data. (11) The current state of a program that is running—including a memory image, the program data, the variables used, the general register values, the status of opened files used, and the current directory. Programs running in a process must be either operating system programs or user programs. [OSF] (12) A running program, including the memory occupied, the open files, the environment, and other attributes specific to a running program.

process group. A collection of processes that permits the signaling of related processes. Each process in the system is a member of a process group that is identified by a process group ID. A newly created process joins the process group of its creator. [POSIX.1]

process group ID. The unique identifier representing a process group during its lifetime. A process group ID is a positive integer that can be contained in a *pid_t*. It shall not be reused by the system until the process group lifetime ends. [POSIX.1]

process ID (PID). (1) A unique number assigned to a process that is running. [OSF] (2) The unique identifier representing a process. A process ID is a positive integer that can be contained in a *pid_t*. A process ID shall not be reused by the system until the process lifetime ends. In addition, if there exists a process group whose process group ID is equal to that process ID, the process ID shall not be reused by the system until the process group lifetime ends. A process that is not a system process shall not have a process ID of `1`. [POSIX.1]

Q

qualifier. (1) A unique name used to identify another name. [OSF] (2) A modifier that makes a name unique. (3) All names in a qualified name other than the rightmost, which is called the simple name.

quote. To mask the special meaning of certain characters, causing them to be taken literally. [OSF]

R

RACF. Resource Access Control Facility.

record. (1) In programming languages, an aggregate that consists of data objects, possibly with different attributes, that usually have identifiers attached to them. [I] (2) A set of data treated as a unit. [T] (3) A collection of fields treated as a unit. [OSF] (4) A self-contained collection of information about a single object. A record is made up of a number of distinct items, called fields. A number of shell programs (for example, **awk**, **join**, and **sort**) are designed to process data consisting of records separated by newlines, where each record contains a number of fields separated by spaces or some other character. **awk** can also handle records separated by characters other then newlines. (5) Using a function to define itself. [OSF] (6) See *fixed-length record, variable-length record*.

redirect. To divert data from a process to a file or device to which it would not normally go. [OSF]

redirection. (1) A system profile construction method of starting at a base platform and adding new services by allowing a service component to ask the base platform to redirect all requests for that type of service to the service component. [POSIX.0] (2) Changing the association between files and file descriptors for a program. A process inherits file descriptors from the process that created the program (usually the shell). A file descriptor's standard input and standard output are usually associated with the keyboard and display screen, respectively. The shell can arrange for these

descriptors (or any others) to be associated with other files before creating the new process. In particular, `<infile` redirects the standard input from **infile**, while `>outfile` redirects the standard output to **outfile**.

regular expression. (1) A mechanism to select specific strings from a set of character strings. [POSIX.2] (2) A set of characters, metacharacters, and operators that define a string or group of strings in a search pattern. [OSF] (3) A string containing wildcard characters and operations that define a set of one or more possible strings. [OSF] (4) A more technical term for *pattern*. (5) See also *wildcard character*.

regular file. A file that is a randomly accessible sequence of bytes, with no further structure imposed by the system. [POSIX.1]

relative pathname. (1) The name of a directory or file expressed as a sequence of directories followed by a filename, beginning from the current directory. Relative pathnames do not begin with a / (slash) but are relative to the current directory. (2) A pathname that does not begin with a slash. The predecessor of the first filename in the pathname is taken to be the current working directory of the process. [POSIX.1] See also *absolute pathname, pathname.*

Resource Access Control Facility (RACF). An IBM-licensed program that provides for access control by identifying and by verifying the users to the system, authorizing access to protected resources, and logging the detected unauthorized access to protected resources.

right-justify. (1) To control the positions of characters on a page so that the right-hand margin of the printing is regular. [I][A] (2) To shift the contents of a register or field so that the character at the right-hand end of the data is at a specific position. [T] (3) To align characters horizontally so that the rightmost character of a string is in a specified position. [A]

root. (1) The starting point of the file system. (2) The first directory in the system.

root directory. (1) The directory that contains all other directories in the system. (2) The lowest directory in the file system hierarchy. It is referred to as "/". (3) A directory, associated with a process, that is used in pathname resolution for pathnames that begin with a slash. [POSIX.1] (4) The top directory in the file system tree. UNIX and POSIX-conforming systems have a single root directory, with mounted devices. (5) See also *effective root directory*.

root file system. The basic file system, onto which all other file systems can be mounted. The root file system contains the operating system files that get the rest of the system running.

row. A horizontal arrangement of characters or other expressions. [A] Contrast with *column.*

run. (1) To cause a program, utility, or other machine function to be performed. (2) A performance of one or more jobs. [I][A] (3) A single, continuous performance of a computer program or routine.

S

S_ISGID bit. Synonym for *set-group-ID mode bit*.

S_ISUID bit. Synonym for *set-user-ID mode bit*.

scroll. (1) To move a display image vertically or horizontally to view data that otherwise cannot be observed within the boundaries of the display screen. (2) To move the representation of data vertically or horizontally relative to the terminal screen. There are two types of scrolling: (1) the cursor moves with the data; or (2) the cursor remains stationary while the data moves. [POSIX.2]

search path. The sequence of directories that a command interpreter should search to find the program that the user wants to run.

security. (1) The protection of data, system operations, and devices from accidental or intentional ruin, damage, or exposure. [OSF] (2) The protection of computer hardware, software, and data from accidental or malicious access, use, modification, destruction, or disclosure. [POSIX.0]

security administrator. A programmer who manages, protects, and controls access to sensitive information.

select. (1) To examine records to find those that satisfy a specified condition. (2) To mark or choose an item in a panel. (3) Contrast with *enter.*

separator. (1) A punctuation character that separates parts of a command or file. (2) A punctuation character used to delimit character strings.

sequential data set. (1) A data set whose records are organized on the basis of their successive physical positions, such as on magnetic tape. Contrast with *direct data set*. (2) A data set in which the contents are arranged in successive physical order and are stored as an entity. The data set can contain data, text, a program, or part of a program. Contrast with *partitioned data set (PDS).*

session. (1) The period of time during which a user of a terminal can communicate with an interactive system—usually, the elapsed time between logon and logoff. (2) The period of time during which programs or devices can communicate with each other. [OSF] (3) A collection of process groups established for job

control purposes. Each process group is a member of a session. Each process is considered to be a member of the session of which its process group is a member. A newly created process joins the session of its creator. A process can alter its session membership. Implementations that support the **setpgid()** function can have multiple process groups in the same session. [POSIX.1] Every process group, and associated process, belongs to a session. Any new process also belongs to the session of the process that created it. (4) In network architecture, an association of facilities that establish, maintain, and release connections for communication between stations. [OSF] (5) In SNA, a logical connection established between two network addressable units (NAUs) that allows them to communicate. For routing purposes each session is identified by the local or network addresses of the session partners.

set flags. Flags that can be put into effect with the shell **set** command. [OSF]

setgid bit. Deprecated term for *set-group-ID mode bit*.

set-group-ID mode bit. In setting file access permissions, the bit that sets the effective group ID of the process to the file's group on execution. Synonymous with S_ISGID bit.

setuid bit. Deprecated term for *set-user-ID mode bit*.

set-user-ID mode bit. In setting file access permissions, the bit that sets the effective user ID of the process to the file's owner on execution. Synonymous with S_ISUID bit.

shared HFS support. Enables systems in a sysplex to share file system data throughout the sysplex. Thus, file systems are available to be accessed (read/write) by all systems participating in shared HFS support.

sh_history. See *history file*.

shell. (1) A program that interprets and processes interactive commands from a pseudoterminal or from lines in a shell script. (2) A program that interprets sequences of text input as commands. It may operate on an input stream, or it may interactively prompt and read commands from a terminal. Synonymous with command language interpreter. [POSIX.2] (3) A software interface between a user and the operating system of a computer. Shell programs interpret commands and user interactions on devices such as keyboards, pointing devices and touch-sensitive screens and communicate them to the operating system. (4) The command interpreter that provides a user interface to the operating system and its commands. (5) The program that reads a user's commands and executes them. (6) The shell command language interpreter, a specific instance of a shell. [POSIX.2] (7) A layer, above the kernel, that provides a flexible interface between users and the rest of the system. (8) Software that allows a kernel program to run under different operating system environments. (9) See also *shell program, KornShell*.

shell procedure. See *shell script*.

shell program. A program that accepts and interprets commands for the operating system. See also *shell*.

shell prompt. The character string on the command line indicating that the system can accept a command.

shell script. (1) A file of shell commands. If the file is executable, a user can run it by specifying the file's name as a shell command or as an operand on **sh** or on the TSO/E OMVS command. A shell script is like a TSO/E REXX exec. (2) A file containing shell commands. If the file is made executable, it can be executed by specifying its name as a simple command: Execution of a shell script causes a shell to execute the commands within the script. Alternately, a shell can be requested to execute the commands in a shell script by specifying the name of the shell script as the operand to the **sh** utility. [POSIX.2]

shell variables. Facilities of the shell program for assigning variable values to constant names.

signal. (1) A means of informing processes of asynchronous events. (2) A mechanism by which a process may be notified of, or affected by, an event occurring in the system. Examples of such events include hardware exceptions and specific actions by processes. The term *signal* is also used to refer to the event itself. [POSIX.1] (3) An indication that an asynchronous event completed. A signal is sent to a process. Signals are simulations of *interrupts*. (4) A simple method of communication between two processes. One process can inform the other process when an event occurs. [OSF] (5) A method of interprocess communication that simulates software interrupts. Contrast with *interrupt*.

single-precision. (1) The use of one computer word to represent a number, in accordance with the required precision. (2) The specification that causes a floating-point value to be stored in the short format.

single-quote. The character "'", also known as *apostrophe*. [POSIX.2]

slash. (1) The literal character "/". This character is also known as *solidus* in ISO 8859-1 [B34] [POSIX.1] (2) The character /. UNIX and POSIX-conforming systems use the slash to separate the parts of a filename.

sort. (1) To rearrange some or all of a group of items, based upon the contents or characteristics of those items. [OSF] (2) To segregate items into groups

according to specified criteria. [I][A] Sorting involves ordering, but need not involve sequencing, for the groups may be arranged in an arbitrary sequence. (3) To arrange a set of items according to keys that are used as a basis for determining the sequence of the items, for example, to arrange the records of a personnel file into alphabetical sequence by using the employee names as sort keys. [A]

source code. (1) The input to a compiler or assembler, written in a source language. Synonymous with source. Contrast with *object code*. (2) In the shell command language, input to the command language interpreter; synonym for *shell script*. [POSIX.2] (3) In the C Language Bindings Option, input to a C compiler conforming to the C Standard {7}. [POSIX.2] (4) In another ISO/IEC conforming language, input to a compiler conforming to that ISO/IEC standard. [POSIX.2] (5) The input statements prepared for the following standard utilities: **awk**, **bc**, **ed**, **lex**, **localedef**, **make**, **sed**, and **yacc**. [POSIX.2] (6) A collection of sources meeting any or all of the foregoing meanings. [POSIX.2]

source file. A file that contains source statements for such items as high-level language problems and data description specifications.

special character. A character other than a letter or number. For example, *, +, and % are special characters.

stack. (1) An area in storage that stores temporary register information and returns addresses of subroutines. (2) A list constructed and maintained so that the last data element stored is the first data element retrieved. [OSF] (3) In kernel mode, an area that is paged with the user process. The kernel maintains a stack for each process. It saves the process information such as the call chain and local variables used by the kernel for the user process.

standard error (stderr). (1) The place where many programs place error messages: the display screen unless another place is specified with redirection. [OSF] (2) An output stream usually intended to be used for diagnostic messages. [POSIX.2] (3) The conventional name for file descriptor 2. By convention, programs write diagnostics and error messages to this descriptor. Usually, the descriptor refers to the display screen, but it may be changed by redirection. This descriptor is separate from standard output so that error diagnostics are still visible when the output is redirected.

standard input (stdin). (1) The primary source of data going into a command. Standard input comes from the keyboard unless redirection or piping is used, in which case standard input can be from a file or the output from another command. (2) An input stream usually intended to be used for primary data input. [POSIX.2] (3) The conventional name for file descriptor 0. By convention, programs read input from this descriptor. Usually, the descriptor refers to the keyboard, but it may be changed by redirection.

standard output (stdout). (1) The primary destination of data coming from a command. Standard output goes to the display unless redirection or piping is used, in which case standard output can be to a file or another command. (2) An output stream usually intended to be used for primary data output. [POSIX.2] (3) The conventional name for file descriptor 1. By convention, programs write output to this descriptor. Usually, the descriptor refers to the display screen, but may be changed by redirection.

standards. Documents, established by consensus and approved by a recognized body, that provide, for common and repeated use, rules, guidelines, or characteristics for activities or their results, aimed at the achievement of the optimum degree of order in a given context. [POSIX.0]

statement. (1) In a programming language, a meaningful expression that may describe or specify operations and is usually complete in the context of that programming language. (2) An instruction in a program or procedure. (3) In programming languages, a language construct that represents a step in a sequence of actions or a set of declarations. [I] (4) In computer programming, a symbol string or other arrangement of symbols. [A]

status. (1) The current condition or state of a program or device—for example, the status of a printer. [OSF] (2) The condition of the hardware or software, usually represented in a status code.

stderr. Standard error.

stdin. Standard input.

stdout. Standard output.

sticky bit. An access permission bit that allows multiple users to show a single copy of an executable file. Only someone with root authority can set the sticky bit.

stop. To end, in a controlled manner, the current processing activity in a computer system because it is impossible or undesirable for the activity to proceed. [OSF] Synonymous with abort.

storage administrator. A programmer who controls use of external storage, including the hierarchical file system.

stream. (1) An ordered sequence of characters, as described by the C Standard {7}. [POSIX.2] (2) Sequential input or output from an open file

descriptor. [OSF] (3) All data transmitted through a data channel in a single read or write operation. [OSF] Synonymous with data stream. (4) A full-duplex connection between a user process and device or pseudodevice. A stream consists of several linearly connected modules, and is analogous to a shell pipeline, except that data flows in both directions. The modules communicate almost exclusively by passing messages to their neighbors. [OSF]

stream editor. An editor invoked by the **sed** command, which modifies lines from a specified file, according to an edit script, and writes them to a standard output. [OSF]

string. (1) A linear sequence of entities such as characters or physical elements. Examples of strings are alphabetic string, binary element string, bit string, character string, search string, and symbol string. [OSF] (2) An ordered sequence of bits, octets, or characters, accompanied by the string's length. [OSF] (3) A sequence of characters or numbers. Any sequence of characters, as in `abc`. For the shell, strings should be enclosed by quotes (") to hide any blanks or tabs in the string from the shell.

structure. (1) A variable that contains an ordered group of data objects. Unlike an array, the data objects within a structure can have varied data types. (2) Synonym for *data structure*.

subcommand. A request for an operation that is within the scope of work requested by a previously issued command.

subdirectory. In the file system hierarchy, a directory contained within another directory. Directories may be nested to arbitrary depth.

subshell. An instance of the shell program started from an existing shell program.

substring. A part of a character string.

suffix. A character string attached to the end of a filename that helps identify its file type. [OSF] For example, in **/dir/file.c**, the suffix is **c**.

superuser. A system user who operates without restrictions. A superuser has the special rights and privileges needed to perform administrative tasks. >

switch. To change processing from the OS/390 shell to a TSO/E command mode or to subcommand mode, or to change back to the shell from one of those modes. In TSO/E command mode, you can enter TSO/E commands; in subcommand mode, you can enter OMVS subcommands.

symbolic link. A type of file system entry that contains the pathname of and acts as a pointer to another file or directory. [OSF]

syntax. (1) The rules for the construction of a command, statement, or program. [OSF] (2) The rules governing the structure of a language. [A] (3) The relationship and structure of characters or groups of characters, independent of their meanings or the manner of their interpretation and use. [I][A]

SYSOUT. A system output stream, or an indicator used in data definition statements, to signify that a data set is to be written on a system output device.]>

sysplex. A set of MVS systems communicating and cooperating with each other through certain multisystem hardware components and software services to process customer workloads. See also *Parallel Sysplex*.

sysplex couple data set. A couple data set that contains sysplex-wide data about systems, groups, and members that use XCF services. All MVS systems in a sysplex must have connectivity to the sysplex couple data set. See also *couple data set*.

Sysplex Timer. An IBM unit that synchronizes the time-of-day (TOD) clocks in multiple processors or processor slides. External Time Reference (ETR) is the MVS generic name for the IBM Sysplex Timer (9037).

system services. Firmware and software that provide an aggregation of network element functions into a higher level function, and provide an interface to the data contained in the system. [POSIX.0]

T

table. (1) An array of data each item of which can be unambiguously identified by means of one or more arguments. [I][A] (2) A two-dimensional array in which each item and its position with respect to other items is identified.

text. (1) Human-readable data, containing only printable characters. (2) A type of data consisting of a set of linguistic characters (letters, numbers, and symbols) and formatting controls. (3) In word processing, information for human comprehension that is intended for presentation in a two-dimensional form—for example, information printed on paper or displayed on a screen. Text consists of symbols, phrases, or sentences in natural or artificial languages. [OSF] (4) A graphic representation of information on an output medium. Text can consist of alphanumeric characters and symbols arranged in paragraphs, tables, columns, or other shapes. (5) In ASCII and data communication, a sequence of characters treated as an entity when preceded by one start-of-text and terminated by one end-of-text communication control character. (6) The part of

a message that is not the header or control information. (7) The control sections of a load module.

text editor. A program used to create, modify, and print or display text files.

text file. A file that contains characters organized into one or more lines. The lines shall not contain NUL characters and none shall exceed {LINE_MAX} bytes in length, including the <newline>. Although POSIX.1 does not distinguish between text files and binary files, many utilities only produce predictable or meaningful output when operating on text files. The standard utilities that have such restrictions always specify *text files* in their Standard Input or Input Files subclauses. [POSIX.2]

tilde. The character "~". [POSIX.2]]>

Time Sharing Option (TSO). An operating system option; for the System/370* system, the option provides interactive time sharing from remote terminals.

Time Sharing Option Extensions (TSO/E). (1) The base for all TSO enhancements. It provides MVS users with additional functions, improved usability, and better performance. (2) In the OS/390 environment, TSO/E also provides virtual storage constraint relief.

TSO/E. Time Sharing Option Extensions.

U

underscore character. A character used in each position of an entry field to indicate its length. This means of indicating entry field length is used on display devices that do not have the underscore attribute. The underscores are replaced by characters typed in the field.

unformatted file. A file displayed with the data that is not arranged with particular characters. Contrast with *formatted file*.

UNIX. A highly portable operating system originally developed by Bell Laboratories that features multiprogramming in a multiuser environment. UNIX is implemented in the C language. UNIX was originally developed for use on minicomputers but has been adapted on mainframes and microcomputers. It is especially suitable for multiprocessor, graphics, and vector-processing systems. Many of the commands in the OS/390 shell are based on similar commands available with UNIX System V. >

unmount. To logically disassociate a mountable file system from another file system. The TSO/E command to perform this action is UNMOUNT or UMOUNT.

update. (1) To modify a master data set with current information according to a specified procedure. (2) An improvement for some part of a system. [OSF]

user address space. An address space that has at least one MVS task known to the kernel address space. This address space can contain a shell or an application program that uses OS/390 UNIX System Services.

user ID. (1) A unique string of characters that identifies an operator to the system. This string of characters limits the functions and information the operator can use. (2) A nonnegative integer, which can be contained in an object of type *uid_t*, that is used to identify a system user. When the identity of the user is associated with a process, a user ID value is referred to as a real user ID, an effective user ID, or an (optional) saved set-user-ID. [POSIX.1] (3) The identification associated with a user or job. The two types of user IDs are:

- **TSO/E user ID:** A string of characters that uniquely identifies a TSO/E user or a batch job owner to the security program for the system. The batch job owner is specified on the USER parameter on the JOB statement or inherited from the submitter of the job. This user ID identifies a RACF user profile.
- **OS/390 UNIX System Services user ID:** A fullword integer that the security administrator assigns to each MVS user ID. This integer, referred to as the UID, is the sole basis for authority checking against such POSIX-defined resources as hierarchical files.

A user ID is equivalent to an account on a UNIX-type system. (4) A symbol identifying a system user. (5) Synonymous with user identification.

user mode. In a multiprocessor environment, the slave in a master-slave relationship. One processor operates in kernel mode, the other (slave) in user mode. Kernel mode handles the interrupts and callable service. User mode informs the master when making a callable service. In user mode, a process is carried out in the user's program rather than in the kernel.

user profile. (1) A description of a user, including user ID, user name, default group name, password, owner, access authority, and other attributes obtained at logon. (2) A file in the user's home directory named **.profile** that contains shell commands that set initial user-defined characteristics and defaults for the session.

V

value. (1) In programming, the contents of a variable or a storage location. [OSF] (2) A quantity assigned to a constant, variable, parameter, or symbol. See also *argument.*

variable-length record. (1) A record having a length

independent of the length of other records with which it is logically or physically associated. (2) Pertaining to a file in which the records need not be uniform in length. [A] (3) Contrast with *fixed-length record.*

vertical tab. The vertical tab character (<vertical tab>). [POSIX.2]

virtual storage access method (VSAM). An access method for direct or sequential processing of fixed- and variable-length records on direct access devices. The records in a VSAM data set or file can be organized in logical sequence by a key field (key sequence), in the physical sequence in which they are written on the data set or file (entry sequence), or by relative record number.

Virtual Telecommunications Access Method (VTAM). Deprecated term for *Advanced Communications Function for the Virtual Telecommunications Access Method (ACF/VTAM).*

vnode. Virtual inode. An object in a file system that represents a file. Unlike an inode, there is no one-to-one correspondence between a vnode and the file system; multiple vnodes can refer to a single file. Vnodes are used to communicate between the upper half of the file system and the file system representations. See also *inode*.

VSAM. Virtual storage access method.

W

wait. (1) A state allowing a parent process to synchronize with the execution of an exit issued by a child process. (2) A state in which no instructions are fetched or executed. Synonymous with wait state.

white space. (1) Space characters, tab characters, newline characters, and comments. [OSF] (2) A sequence of one or more spaces or horizontal tab characters. White space is used to separate commands on the command line. (3) A sequence of one or more characters that belong to the **space** character class as defined via the LC_CTYPE category in the current locale. In the POSIX Locale, white space consists of one or more <blank>s (<space>s and <tab>s), <newline>s, <carriage return>s, <form-feed>s, and <vertical tab>s. [POSIX.2]

wildcard character. Either the * and ? special character that can be used in a file specification to match one or more characters. For example, placing ? in a file specification means any character can be in that position. For another example, the file selector ?a matches the filenames **aa**, **Ba**, **ca**, and so on. The * wildcard character matches any sequence of zero or more characters; thus the file selector a* matches any name beginning with **a**. Synonymous with global character (in an MVS context), pattern-making character. See also *regular expression*.

working directory. (1) The active directory used to resolve pathnames that do not begin with a slash. In similar systems, a working directory may be called the *current directory* or the *current working directory*. (2) A directory, associated with a process, that is used in pathname resolution for pathnames that do not begin with a slash. [POSIX.1] (3) Synonym for *current directory*. (4) See also *current directory*.

write access. In computer security, permission to write to an object. Synonymous with write permission.

write permission. Synonym for *write access*.

XCF. Cross-system coupling facility.

Index

Special Characters

-
  explanation of 1
_ environment variable
  description of 507
_BPX_ACCT_DATA environment variable
  description of 839
_BPX_BATCH_SPAWN environment variable
  description of 803, 839
_BPX_BATCH_UMASK environment variable
  description of 803, 839
_BPX_JOBNAME environment variable
  description of 839
_BPX_PTRACE_ATTACH environment variable
  description of 839
_BPX_SHAREAS environment variable
  description of 839
_BPX_SPAWN_SCRIPT environment variable
  description of 803, 839
  magic number
    #! 803
_BPX_TERMPATH environment variable
  description of 839
  used by chcp 105, 106
_CEE_ENVFILE environment variable
  description of 840
: (colon) shell command 121
? subcommand for dbx 160
/ subcommand for dbx 161
/bin directory
  setting up special files in the 739
/bin/mail file
  used by calendar 95
/def/ocslock file
  used by ocsconfig 418
/dev/mt/0m file
  used by tar 553
/dev/ocsadmin file
  used by ocsconfig 418
/etc/auto.master file
  used by automount 23
/etc/banner file
  used by lm 311
/etc/csh.cshrc
  used by tcsh login 557
/etc/csh.login
  used by tcsh login 557
/etc/inetd.conf file
  used by the inetd daemon 283
/etc/inetd.pid file
  used by the inetd daemon 286
/etc/lm.rc file
  used by lm 311
/etc/magic file
  explanation of 822
  used by file 255
/etc/mailx.rc file
  used by mailx 337, 348, 354
/etc/profile file 510
  used by the login shell 486
/etc/rc file
  used by automount 23
  used by lm 311
/etc/recover file
  used by exrecover 246
/etc/recover/$LOGNAME/VIn* file
  used by exrecover 246
/etc/recover/$LOGNAME/VIt* file
  used by exrecover 246
/etc/services file
  used by lm 311
  used by ocsconfig 418
/etc/startup.mk file
  used by make 374
/etc/suid_us.profile
  used by sh 510
/etc/system file
  used by ocsconfig 418
/etc/utmpx file
  used by who 724
/etc/wtmp file
  used by who 724
/etc/yylex file
  used by lex 308
/etc/yylex.c file
  used by lex 306
/etc/yyparse.c file
  used by yacc 733
/tmp file
  used by ar 16
  used by fc, history, r 251
/tmp/e* file
  used by ed 232
/tmp/sh* file
  description of 510
/tmp/stm* file
  used by sort 525
/tmp/VIl* file
  used by exrecover 246
/tmp/VIn* file
  used by exrecover 246
/tmp/VIt* file
  used by exrecover 246

© Copyright IBM Corp. 1996, 2000

/usr/lib/config file
used by uucc 664
/usr/lib/cron/at.allow file
used by cron 144
/usr/lib/cron/at.deny file
used by cron 144
/usr/lib/cron/cron.allow
used by cron 145
/usr/lib/cron/cron.deny file
used by cron 145
/usr/lib/cron/queuedefs file
used by cron 145
/usr/lib/hash file
used by spell 528, 529
/usr/lib/hashb file
used by spell 528, 529
/usr/lib/lib.b file
used by bc 59
/usr/lib/libl.a file
used by lex 308
/usr/lib/lwords file
used by spell 528, 529
/usr/lib/uucp file
used by uucc 664
/usr/lib/uucp/config
used by uucc 664
used by uuto 682
/usr/lib/uucp/config file
used by uucico 666
used by uucp 665, 670
used by uulog 675
used by uuname 676
used by uupick 678
used by uustat 681
used by uux 686
used by uuxqt 687
/usr/lib/uucp/devices file
used by uucp 664
/usr/lib/uucp/dialcodes file
used by uucp 665
/usr/lib/uucp/dialers file
used by uucp 664
/usr/lib/uucp/permissions file
used by uucp 665
/usr/lib/uucp/systems file
used by uucp 664
/usr/man/%L/man¬0–9l/*.book file
used by man 379
/usr/man/%L/whatis file
used by man 380
/usr/spool/cron file
used by cron 144
/usr/spool/cron/atjobs file
used by cron 144
/usr/spool/cron/crontabs file
used by cron 144
/usr/spool/cron/log file
used by cron 144
/usr/spool/cron/pid
used by cron 144
/usr/spool/locks
used by uucico 666
/usr/spool/uucp
used by uustat 681
/usr/spool/uucp file
used by uulog 675
/usr/spool/uucp spool directory 831
/usr/spool/uucp/.Xqtdir directory 684
/usr/spool/uucp/.Sequence file
used by uucp 670
used by uux 686
used by uuxqt 687
/usr/spool/uucp/.STATUS file
used by uucico 666
used by uustat 681
/usr/spool/uucp/.Xqtdir
used by uuxqt 687
/usr/spool/uucp/LOGFILE
used by uulog 675
/usr/spool/uucp/LOGFILE file
used by uucico 666
used by uucp 670
used by uux 686
/usr/spool/uucp/site
used by uuxqt 688
/usr/spool/uucp/site file
used by uux 686
/usr/spool/uucp/south directory 831
/usr/spool/uucppublic file
used by uucp 670
used by uuto 683
/usr/spool/uucpublic file
used by uupick 678
/usr/spool/uucpublic/receive file
used by uupick 678
/var/man/%L/entry.¬0–9l/*.bookname file
used by man 379
. (dot) script
returning from 471
. (dot) shell command 221
... (ellipsis)
explanation of 2
.dbxinit file
used by dbx 158
.dbxsetup file
used by dbx 158
.exrc file
used by vi 715
.profile file 510
[(left bracket) shell command 304
[]
explanation of 1

$HOME / .sh_history file
used by fc, history, r 250
$HOME/.exrc file
used by vi 715
$HOME/mbox file
used by mail 336
used by mailx 354
$MAILDIR file
used by mailx 354
$MAILRC file
used by mailx 354
$TMPDIR/pg* file
used by pg 449
& shell operator 489
#!
magic number 807
|& shell operator 489

Numerics

3270 alarm
controlling the 764, 769, 774
3270 passthrough mode
used to invoke the TSO/E OBROWSE command 411
3270 terminals
specifying the use of DBCS 766

A

a.out file
used by dbx 160
abnormal condition
trapping 637
abnormal interrupt
trapping 637
absolute movement command (for vi) 690
absolute movement commands
list of 692
access permission
changing 109
access time
resetting 140
setting, for destination files 132
action
explanation of 35
address
removing breakpoints from 165
AF_UENT stack
configuring the 127
ALARM function key for OMVS command 769
ALARM option of OMVS command 764
ALARM subcommand of OMVS command 774
alias
creating 11
tracked aliases 276
alias *(continued)*
detecting 652
removing
definitions 652
those specified by the name argument 202
alias shell command 11, 722
alias subcommand for dbx command 161
allnet environment variable
used by mailx 350, 355
alloc tcsh shell command 601
allocate
MVS standard files as HFS files
using the BPXBATCH program 803
OS/390 standard files as HFS files
using the BPXBATCH command 738
standard files
using the BPXBATCH command 738
using the BPXBATCH program 803
stdenv as HFS files
using the BPXBATCH command 738
using the BPXBATCH program 803
stderr as HFS files
using the BPXBATCH command 738
using the BPXBATCH program 803
stdin as HFS files
using the BPXBATCH command 738
using the BPXBATCH program 803
stdout as HFS files
using the BPXBATCH command 738
using the BPXBATCH program 803
ALLOCATE TSO/E command 748
allow
messages 380
American Daylight Savings time
used in the TZ environment variable 838
ampm shell variable
description of 587
append
user's commands to a file
of given identifiers 186
application program
displaying
list of mutex objects 199
appointment
displaying 95
ar shell command 14
arbitrary-precision arithmetic calculation language
using the 44
archive
copying files from directory 140
creating 140
extracting
components from the 434
contents 140
tapes 552

archive file
cpio format 821
manipulating 552
reading 433, 552
tar format 828
writing 433, 552
archive library
creating 14
displaying symbol table 407
maintaining 14
ARGC (built-in variable for awk) 30
args subcommand for the dbx command 162
argument
changing dates for 631
concatenating in the current shell environment 236
evaluating
as an expression 243
in the current shell environment 236
obtaining from a list of parameters 271
printing 454
removing 473
returning from the shell 454
writing to standard output 225
argv shell variable 587
description of 587
arithmetic calculation
calculating to arbitrary precision 44
arithmetic expression
evaluating 305
arrange
items on command line 3
options 1
array element (awk variable) 27
asa shell command 17
ASA/FORTRAN carriage control
interpreting 17
ASCII code pages for the terminal
setting, resetting, or querying 105
ASCII to EBCDIC conversion 208
ask environment variable
used by mailx 350
askbcc environment variable
used by mailx 350
asksub environment variable
used by mailx 350
assemble
OS/390 C and OS/390 C++ source files 64
assign
aliases for dbx subcommands 161
attributes to variables 647
values to variables 162, 647
assign subcommand for dbx command 162
at shell command 19
submitting jobs to cron 146
attribute of files
listing 327
audit attribute
changing 103
audit flag
changing 103
autocorrect shell variable
description of 587
autoexpand shell variable
description of 587
autolist shell variable
description of 587
autoloaded functions
description of 494
autologout shell variable
description of 587
automatic scrolling
controlling 769, 774, 775
turning off 775
Automatic, Periodic, and Timed Events 582
automount facility
configuring the 23
automount shell command 23
autoprint environment variable
used by mailx 338, 345, 350
AUTOSCROLL function key for OMVS command 769
AUTOSCROLL function of OMVS command
setting the 764
AUTOSCROLL option of OMVS command 764, 769
AUTOSCROLL subcommand of OMVS command 774
awk shell command 25
action 35
ARGC built-in variable 30
arrays 27
BEGIN 35
built-in arithmetic functions 32
built-in string functions 33
built-in variables
FILENAME 31
FNR 31
NF 31
NR 31
comments 27
conditions 35
END 35
ENVIRON environment 29
examples 37
FILENAME file being read 31
FNR number of records read from file 31
FS field separator string 31
functions 35
getline 31
NF field in current record 31
NR number of records read 31
OFMT output number format 36
OFS output field separator 36

awk shell command *(continued)*
operators 29
ORS output record separator 36
patterns 35
processing programs 25
RLENGTH built-in variable 33
RS record separator character 31
RSTART built-in variable 33
statements 35
SUBSEP 27
SYMTAB symbol table 29
system functions 34
variables 27

B

background job
scheduling 145
backquoting 500
BACKSCR function key for OMVS command 769
BACKSCR subcommand of OMVS command 774
backslash shell variable
description of 587
backup files 552
backward retrieve function of OMVS command 771
bang environment variable
used by mailx 347, 350
basename shell command 42
basic regular expression
explanation of 797
list of commands using 800
meaning of metacharacters used 797
batch environment
running shell commands, shell scripts, and C/MVS 738
running shell scripts and OS/390 C applications under MVS 803
batch job
submitting
using the BPXBATCH command 738
using the BPXBATCH program 803
batch shell command 43
submitting jobs to cron 146
bc shell command 44
built-in functions 56
built-in variables 47
dynamic scoping 55
specifying numbers in different bases 47
between-rule circular dependency 362
bg shell command 61
/bin directory
setting up special files in the 739
binary file
decoding 672
encoding for transmission 673
bindkey tcsh sell command 601
blind carbon copy 338
bold typeface
explanation of 1
BOTTOM function key for OMVS command 769
BOTTOM subcommand of OMVS command 774
Bourne shell 486
BPXACOPY program
automatic setting of permission bits during installation 813
BPXBATCH program 803, 809
invoked in the OSHELL REXX exec 809
invoking
with OSHELL 783
BPXBATCH TSO/E command 738
BPXBATSL
run program using local spawn 738, 803
BPXCOPY program 811, 816
invoking 811
BPXK_MDUMP environment variable
description of 840
bpxmtext shell command 62
bracket expression 797
brackets
explanation of 1
break
lines 261
break shell command 63
break up
files 530
text file 148
breakpoint
removing from addresses 165
broadcast message 721
browse
files, with BPXBATCH 783
files, with the ISPF shell 739
HFS files
with the obrowse shell command 411
with the OBROWSE TSO/E command 748
build
argument lists before running a command 727
list of files 784
built-in functions
for the bc shell command 56
built-in shell commands 505
: (colon) 121
[304
alias 11
bg 61
break 63
cd 100
colon (:) 121
description of 494
echo 225
exit 240

built-in shell commands *(continued)*
false 249
fc 250
getopts 271
hash 276
jobs 298
kill 301
let 305
print 454
pwd 465
read 466
test 624
time 629
times 630
true 639
type 646
typeset 647
ulimit 650
umask 654
unalias 652
wait 720
whence 722
built-in variable
for the bc command 47
builtins tcsh shell command 603
byte count
calculating
with the cksum command 117
with the sum command 543
displaying
with the cksum command 117
with the sum command 543
bytes
counting 721
swapping 141

C

C escape sequences 427
C functions
displaying online 762
C/MVS executable file
running
using the BPXBATCH command 738
using the BPXBATCH program 803
c++ shell command 64
c89/cc/c++ environment variable
_ACCEPTABLE_RC 75
_ASUFFIX 75
_ASUFFIX_HOST 75
_CCMODE 75
_CLIB_PREFIX 75
_CMEMORY 76
_CMSGS 76
_CNAME 76
_CSUFFIX 76
c89/cc/c++ environment variable *(continued)*
_CSUFFIX_HOST 76
_CSYSLIB 77
_CXXSUFFIX 77
_CXXSUFFIX_HOST 77
_DAMPLEVEL 77
_DAMPNAME 77
_DCB121M 78
_DCB133M 78
_DCB137 78
_DCB137A 78
_DCB3200 78
_DCB80 78
_DCBF2008 78
_DCBU 78
_ELINES 78
_EXTRA_ARGS 79
_ILCTL 79
_ILMSGS 79
_ILNAME 79
_ILSUFFIX 79
_ILSUFFIX_HOST 80
_INCDIRS 80
_INCLIBS 80
_ISUFFIX 80
_ISUFFIX_HOST 80
_IXXSUFFIX 80
_IXXSUFFIX_HOST 80
_LIBDIRS 80
_LSYSLIB 80
_MEMORY 81
_NEW_DATACLAS 81
_NEW_DSNTYPE 81
_NEW_MGMTCLAS 81
_NEW_SPACE 81
_NEW_STORCLAS 81
_NEW_UNIT 81
_OPERANDS 81
_OPTIONS 81
_OSUFFIX 81
_OSUFFIX_HOST 82
_OSUFFIX_HOSTQUAL 82
_OSUFFIX_HOSTRULE 82
_PMEMORY 83
_PMSGS 83
_PNAME 83
_PSUFFIX 83
_PSUFFIX_HOST 83
_PSYSIX 83
_PSYSLIB 84
_SLIB_PREFIX 84
_SNAME 84
_SSUFFIX 84
_SSUFFIX_HOST 84
_SSYSLIB 84
_STEPS 84

c89/cc/c++ environment variable *(continued)*
_SUSRLIB 85
_TMPS 85
_WORK_DATACLAS 85
_WORK_DSNTYPE 86
_WORK_MGMTCLAS 86
_WORK_SPACE 86
_WORK_STORCLAS 86
_WORK_UNIT 86
_XSOMDIRS 86
_XSOMLIBS 86
_XSUFFIX 86
_XSUFFIX_HOST 87
c89/cc/c++ shell command
environment variables 74
options 65
specifying
system and operational information to c89/cc/c++/cxx 74
–W option
compiler, prelinker, IPA linker and link editor options 69
DLL, SOM and IPA extensions 69
cal shell command 94
calculate
checksum for each input file
with the cksum command 117
with the sum command 543
number of bytes in each input file
with the cksum command 117
with the sum command 543
calendar
displaying 94
calendar file 96
calendar shell command 95
call up
other systems 152
callable services
displaying online 762
cancel
print queue requests 96
cancel shell command 96
captoinfo shell command 97
carriage control
interpreting 17
case of letters 1
case shell command 492
case subcommand for dbx command 163
catch subcommand for dbx command 163
Caution section
explanation of 6
cc shell command 100
cd shell command 100
CDPATH environment variable
description of 507
used by cd 101
CDPATH environment variable *(continued)*
used by vi 712, 716
cdpath shell variable
description of 587
change
audit attributes 103
audit flags 103
crontab entries 145
dates for arguments 631
dbx command prompts 184
file access times 631
file modification times 631
files
using diff output 429
functions 172
group owners 108
groups 402
groups of directories 113
groups of files 113
mount mode 785
next line to be displayed 179
owners of directories 113
owners of files 113
priorities of running processes 469
program counter address 173
root directory 114
source files 172
terminal characteristics 633
user ID
connected with sessions 540
working directories 100
character
counting 721
escaping 495
translating 635
character class expression 798
character conversion table
specifying the 764
character special files
creating 742
characters
converting from one code set to another 280
chaudit shell command 103
chcp shell command 105
check
conditions 624
for spelling errors 527
pathnames 432
checksum
calculating 117, 543
displaying 117
with the sum command 543
chgrp shell command 108
child process
waiting for it to end 720

child process time
displaying time accumulated 630
child shell environment 504
chmod shell command 109
chmount
change file system mount attributes 112
chmount shell command 112
chown shell command 113
chroot shell command 114
circular dependencies 362
cksum shell command 117
clear breakpoints at addresses 165
clear subcommand for dbx command 164
cleari subcommand for dbx command 165
clock daemon (cron) 143
clocks 629
clone output streams 623
close 503
file descriptors 238
shell sessions 770, 774
standard output (stdout) 503
CLOSE function of OMVS command 770
CLOSE subcommand of OMVS command 774
cmd environment variable
used by mailx 343, 350
cmp shell command 118
code set
converting characters to another code set 280
col shell command 119
collation sequence 797
collect
debugging information 767
colon (:) shell command 121
COLUMNS environment variable 507
description of 507
used by ed 232
used by ls 331
used by more 390
used by pg 446, 449
used by ps 464
used by sed 480
used by shedit 513
used by vi 716
COLUMNS tcsh environment variable
description of 598
comand shell variable
description of 587
comm shell command 122
command
aliases
creating or displaying 11
built-in 505
changing prompts, for dbx 184
constructing
in the current shell environment 236
with templates 727

command *(continued)*
conventions 1
creating aliases 11
descriptions
reading 1
displaying 250
aliases 11
elapsed time 629
online 762
process IDs 263
suppressing command numbers 251
editing 250, 513
executing 114
interpreting names 722
names
interpreting 722
numbers
suppressing 251
options
setting 482
unsetting 482
passing to shell for execution 194
processing history lists 250
prompts, changing for dbx 184
reading descriptions 1
reentering 250
remote execution
displaying information about 674
running
after constructing an argument list 727
at a different priority 404
at a specified time 19
from the shell 640
simple 123
using the OMVS interface 640
using the TSO/E service routine 640
when system is not busy 43
setting options 482
specifying command lines for another command 238
substituting 500
suppressing numbers 251
sysvar 545
template 727
TSO/E
ALLOCATE 748
BPXBATCH 738
ISHELL 739
MKDIR 740
MKNOD 741
MOUNT 743
OBROWSE 747
OCOPY 748
OEDIT 753
OGET 754
OGETX 758
OHELP 762

command *(continued)*
TSO/E *(continued)*
OMVS 764
OPUT 776
OPUTX 780
OSHELL 783
OSTEPLIB 784
UNMOUNT 785
unsetting options 482
command aliases
displaying 11
command interpreter 356
command line
specifying for another command 238
command mode 338, 689
command shell command 123
command substitution 500
command summary
vi mode 690
commands
nonfunctional
cancel 96
cu 152
lpstat 326
running
on remote sites 683
communicate
with other users 726
compare
directories
with the dircmp command 219
with the ISHELL command 739
with the OSHELL command 783
files
with the cmp command 118
with the diff command 214
with the ISHELL command 739
with the OSHELL command 783
terminfo database entries 286
compile
link-edit object file 64
OS/390 C and OS/390 C++ source file 64
terminfo database entries 627
UUCP configuration files 664
compiler
yacc 731
complete shell variable
description of 587
component directory 433, 551
component file 433, 551
compress
files
using Huffman coding 424
using Lempel-Ziv compression 124
spaces into tabs 656
compress shell command 124
concatenate
arguments in the current shell environment 236
corresponding or subsequent lines of files 427
files 98
lines 427
lines of input files 427
regular expressions 799
condition
explanation of 35
testing for 624
trapping abnormal 637
condition subcommand for dbx command 165
condition variable
displaying list of 165
confighfs shell command 126
configstk shell command 127
configuration files
/usr/lib/config
used by uucc 664
Devices 664
Dialcodes 664
Dialers 664
Permissions 664
reading and compiling contents of UUCP 664
Systems 664
configuration variable
writing values to standard output 267
configure
AF_UEINT stacks 127
automount facility, the 23
OCS objects 412
conflicting pathname 313
connect to
other systems 152
connecting to
remote systems, with the uucico daemon 665
console log
saving messages in 322
construct
argument lists before running a command 727
commands in the current shell environment 236
cont subcommand of for command 166
context diff file 429
context-dependent movement commands (for vi) 691
continuation prompt 509
continue shell command 129
control
3270 alarms 764, 769, 774
automatic scrolling 769, 774, 775
AUTOSCROLL function of OMVS command 764
display of function key settings 771
control character
processing 119

CONTROL function of OMVS command 770
control operator 493
conv environment variable
used by mailx 337, 355
conventions for command descriptions 1
conversion buffer 210
convert
characters from one code set to another 280
files 207
from ASCII to EBCDIC 208
from EBCDIC to ASCII 208
from lowercase to uppercase 209
from uppercase to lowercase 209
from variable to fixed records 208
source definitions for locale categories 320
CONVERT option of OMVS command 764
copy
archive files, with the tar command 552
data read from standard input to standard output 635
data sets into files, with BPXCOPY 811
data with format conversion 207
file descriptors 238
files
between UUCP systems 667
from one directory to another 140
selectively 152
to MVS partitioned data set 758
to target named by the last argument on command line 130
to users on remote systems 682
with BPXBATCH 783
with data conversion 207
with the ISPF shell 739
HFS
directories to MVS partitioned data set 758
files into MVS data sets 754
files to another member or file 748
in/out file archive 140
lines 309
MVS data sets
members into HFS directories 780
members into HFS files 777
to another member or file 748
one line of standard input 309
standard input to each output file 623
copy mode 434
correct shell variable
description of 587
count
bytes 721
characters 721
lines 721
newlines 721
words 721
cp shell command 130
cpio archive
reading and writing 140
cpio archive format 821
cpio file format 821
cpio shell command 140
CPU time 630
create
aliases for dbx subcommands 161
archives 140
character special files 742
command aliases 11
crontab entries 145
directories
for each named directory argument 383
with the MKDIR command 740
FIFO special files 385
hard link 310
hierarchical file systems (HFS) 739
libraries 373
library archives 14
link to files 313
message catalogs 264
tag files 150
tracked aliases 276
create executable files 64
cron
submitting jobs to 146
cron daemon 146
crontab
changing entries 145
creating entries 145
editing entries 146
obtaining output of entries 146
crontab shell command 145
submitting jobs to cron 146
crt environment variable
used by mailx 343, 351
csplit shell command 148
ctags shell command 150
cu shell command 152
current appointment
displaying 95
current mail message 339
current operating system
displaying name of the 654
current position pointer 690
current users
displaying information about 723
current working directory
changing to previous working directory 100
displaying pathname of the 465
setting to value of the HOME environment variable 100
customize
settings for function keys 769

cwd shell variable
description of 587

D

daemons
cron 143
exrecover 245
inetd 283
login monitor 311
OCS 414
uucico 665
uucpd 671
uuxqt 687
dash
explanation of 1
data
displaying after uncompressing 735
manipulating 25
reading 207
refreshing 771
removing from executable files 533
transferring to remote sites 833
writing 207
data file 833
data set
copying
between two files 748
BPXCOPY program, with the 811
database
joining two 299
dataset names
specifying in the shell 843
date
displaying the 154
date shell command 154
DBCS mode
specifying 766
DBCS option of OMVS command 766
dbx debug program
defining values for variables 190
searching for source files 204
dbx shell command 157
creating aliases for subcommands 161
displaying synopsis of 174
reading subcommands from file 195
subcommands
? 160
/ 161
alias 161
args 162
assign 162
case 163
catch 163
clear 164
cleari 165
condition 165
dbx shell command *(continued)*
subcommands *(continued)*
cont 166
delete 167
detach 168
display memory 168
down 170
dump 170
edit 171
file 172
func 172
goto 173
gotoi 173
help 174
history 174
ignore 175
list 175
listfiles 176
listi 178
map 179
move 179
multproc 180
next 181
nexti 182
object 183
onload 183
print 184
prompt 184
quit 185
record 186
registers 187
rerun 188
return 188
run 189
set 190
sh 194
skip 194
source 195
status 195
step 196
stepi 196
stop 197
stopi 198
trace 200
tracei 201
unalias 202
unset 203
up 203
use 204
whatis 205
where 205
whereis 206
which 207
dbx shell command prompt
changing 184

dd shell command 207
deactivate
function key 771
dead environment variable
used by mailx 351
dead.letter file
used by mail 334, 335
used by mailx 346
DEBUG option of OMVS command 767
debug programs
changing interpretation of symbols 163
with the dbx command 157
debug session
enabling or disabling multiprocess 180
ending 185
debugger
using the 157
debugging information
collecting 767
writing 767
decode
files packed by using Hoffman coding 661
Huffman-packed files 445
transmitted binary files 672
default
function key 772
define
local environments 320
values for dbx variables 190
delay program execution 521
delete
alias definitions 652
aliases 202
arguments 473
attributes of variables and functions 663
breakpoints at addresses 165
directories 474
directory entries 473, 659
information from executable files 533
stops
from programs 167
from source lines 164
traces from program 167
trailing part of filenames 220
values of variables and functions 663
variables 203
delete subcommand of the for command 167
Description section
explanation of 3
destination file
setting
destination time 132
modification time 132
detach subcommand for dbx command 168
detect
aliases 652
detect *(continued)*
spelling errors 527
determine
file types 254
/dev/tty 452
dextract shell variable
description of 587
df
in a sysplex 213
df shell command 212
diff output
used when changing files 429
diff shell command 214
dircmp shell command 219
directory
/usr/spool/uucp 831
/usr/spool/uucp/.Xqtdir 684
/usr/spool/uucp/south 831
changing
access permission of 109
audit attributes 103
audit flags 103
group owners 108
modes 109
owners and groups 113
comparing 219
with the ISHELL command 739
with the OSHELL command 783
copying files 140
creating
for each named directory argument 383
with BPXBATCH 783
with the ISPF shell 739
with the MKDIR command 740
deleting, with BPXBATCH 783
deleting, with the ISPF shell 739
listing files in a
with ISHELL 739
with OSHELL 783
moving files to a different 393
naming, with BPXBATCH 783
naming, with the ISPF shell 739
removing
entries 473, 659
with the rmdir command 474
searching 204
setting owners and groups 113
setting up special files in the /bin 739
directory (UUCP)
searching public 677
directory substitution 496
dirname shell command 220
dirsfile shell variable
description of 587
dirstack shell variable
description of 588

disable multiprocess debugging 180
display 177
active stop subcommands 195
active trace subcommands 195
aliases for dbx subcommands 161
amount of free space on file system 212
appointments 95
arguments
of programs 162
attributes and contents of symlink 739, 783
C functions online 762
calendars 94
callable services online 762
changing next line to be displayed 179
checksum for each input file
with the cksum command 117
with the sum command 543
command aliases 11
commands
suppressing command numbers 251
with the fc command 250
commands in history list 174
commands online 762
compressed files 445
crontab entries 146
current appointments 95
currently exported variables 242
data after uncompressing 735
dates 154
DBCS characters 9
declaration of program components 205
differences between two files 214
elapsed time for a command 629
environment variables 235, 482
extended attributes for files 249
file attributes 739, 783
files 98
page by page 387
files interactively 446
first part of files 277
information about current users 723
information about locales 317
information about the OMVS command 774
input files 446
instructions in program 178
last part of files 548
lines common to two files 122
list of active condition variables 165
list of active mutex objects 180, 199
list of active program and functions 205
list of active read/write lock objects 185
list of files
of module 176
list of functions associated with a program file 177
list of UUCP systems 676
load characteristics of program 179
display *(continued)*
log information about UUCP events 674
login information 723
memory 168
message catalogs 222
messages from message catalogs 223
names of
current operating systems 654
shell variables 482
variables in procedures 170
number of bytes in each input file
with the cksum command 117
with the sum command 543
OS/390 UNIX System Services publications online 762
pathname of working directories 465
piped files 446
process status 458
processors 629
qualifications
of given identifiers 207
of symbols 206
reason_code text 62
shell messages 762
specified number of lines in source files 175
static system symbols 545
status of pending UUCP transfers 679
status of print queues 326
strings in a binary file 532
synopsis of dbx commands 174
system time accumulated by commands 630
terminal names 645
terminal options 534
times 154
unprintable characters 99
user time accumulated by the shell 630
values of
floating-point registers 187
general-purpose registers 187
instruction registers 187
program status words (PSW) 187
shell variables 482
system control registers 187
variables in procedures 170
values of environment variables 455
display memory subcommand for dbx command 168
DISPLAY tcsh environment variable
description of 598
display-oriented text editor
vi 688
displaying
man pages 378
displays
reason_code text 62

DLL (dynamic link library)
description of 69
link-editing 69
dot (.) script
returning from 471
dot (.) shell command 221
dot environment variable
used by mailx 351
double-spacing 450
doublebyte character set 766
See also DBCS
displaying characters 9
locales
switching 9
strings 9
using the 7
doublebyte characters
converting 280
down subcommand for dbx command 170
dspcat shell command 222
dspmsg shell command 223
du shell command 224
dump file to standard output 420
dump subcommand for dbx command 170
dunique shell variable
description of 588
duplicate output stream 623
dynamic link library (DLL)
description of 69
link-editing 69
dynamic scoping
used in the bc shell command 55

E

EBCDIC to ASCII. conversion 208
ECHO function key for OVMS command 770
ECHO option of OMVS command 767
echo shell command 225
echo shell variable
description of 588
ECHO subcommand of OMVS command 774
echo_style shell variable
description of 588
ed shell command 227
ed text editor
using the 227
ed.hup file
used by ed 232
edit
commands 250
crontab entries 146
files, with BPXBATCH 783
files, with the ISPF shell 739
HFS files
with the oedit shell command 423
with the OEDIT TSO/E command 754
edit *(continued)*
message catalogs 264
edit shell variable
description of 588
edit subcommand for dbx command 171
editing subcommands
starting 476
editor
invoking 171
EDITOR environment variable
description of 507
used by crontab 147
used by mailx 341, 346, 351
used by shedit 513
editor initialization 715
EDITOR tcsh environment variable
description of 598
egrep shell command 274
electronic mail
sending and receiving 335
elif shell subcommand 492
ellipsis
explanation of 2
ellipsis shell variable
description of 589
else shell subcommand 492
emacs
enabling, with the EDITOR environment variable 507
enable multiprocess debugging 180
encode
binary files for transmission 673
files
using Huffman coding 424
end
dbx debugging sessions 185
jobs 301
processes 301
shell sessions 771, 775
shells 240
end of file 548
ENDPASSTHROUGH option of OMVS command 767
ENV environment variable
description of 840
used by sh 486, 489
used by vi 716
env shell command 235
environment
defining local 320
environment variable
_ 507
_ACCEPTABLE_RC
used by c89/cc/c++ 75
_ASUFFIX
used by c89/cc/c++ 75

environment variable *(continued)*
_ASUFFIX_HOST
used by c89/cc/c++ 75
_BPX_ACCT_DATA
description of 839
_BPX_BATCH_SPAWN
description of 803, 839
_BPX_BATCH_UMASK
description of 803, 839
_BPX_JOBNAME
description of 839
_BPX_PTRACE_ATTACH
description of 839
_BPX_SHAREAS
description of 839
_BPX_SPAWN_SCRIPT
description of 803, 839
_BPX_TERMPATH
description of 839
used by chcp 105, 106
_BPX_USERID
description of 839
_BPXK_MDUMP
description of 840
_CCMODE
used by c89/cc/c++ 75
_CEE_ENVFILE
description of 840
_CLIB_PREFIX
used by c89/cc/c++ 75
_CMEMORY
used by c89/cc/c++ 76
_CMSGS
used by c89/cc/c++ 76
_CNAME
used by c89/cc/c++ 76
_CSUFFIX
used by c89/cc/c++ 76
_CSYSLIB
used by c89/cc/c++ 77
_CXXSUFFIX
used by c89/cc/c++ 77
_CXXSUFFIX_HOST
used by c89/cc/c++ 77
_DAMPLEVEL
used by c89/cc/c++ 77
_DAMPNAME
used by c89/cc/c++ 77
_DCB121M
used by c89/cc/c++ 78
_DCB133M
used by c89/cc/c++ 78
_DCB137
used by c89/cc/c++ 78
_DCB137A
used by c89/cc/c++ 78

environment variable *(continued)*
_DCB3200
used by c89/cc/c++ 78
_DCB80
used by c89/cc/c++ 78
_DCBF2008
used by c89/cc/c++ 78
_DCBU
used by c89/cc/c++ 78
_ELINES
used by c89/cc/c++ 78
_EXTRA_ARGS
used by c89/cc/c++ 79
_ILCTL
used by c89/cc 79
_ILMSGS
used by c89/cc 79
_ILNAME
used by c89/cc/c++ 79
_ILSUFFIX
used by c89/cc 79
_ILSUFFIX_HOST
used by c89/cc 80
_INCDIRS
used by c89/cc/c++ 80
_INCLIBS
used by c89/cc/c++ 80
_ISUFFIX
used by c89/cc/c++ 80
_ISUFFIX_HOST
used by c89/cc/c++ 80
_IXXSUFFIX
used by c89/cc/c++ 80
_LIBDIRS
used by c89/cc/c++ 80
_LSYSLIB
used by c89/cc/c++ 80
_MEMORY
used by c89/cc/c++ 81
_NEW_DATACLAS
used by c89/cc/c++ 81
_NEW_DSNTYPE
used by c89/cc/c++ 81
_NEW_MGMTCLAS
used by c89/cc/c++ 81
_NEW_SPACE
used by c89/cc/c++ 81
_NEW_STORCLAS
used by c89/cc/c++ 81
_NEW_UNIT
used by c89/cc/c++ 81
_OPERANDS
used by c89/cc/c++ 81
_OPTIONS
used by c89/cc/c++ 81
_OSUFFIX
used by c89/cc/c++ 81

environment variable *(continued)*
_OSUFFIX_HOST
used by c89/cc/c++ 82
_OSUFFIX_HOSTQUAL
used by c89/cc/c++ 82
_OSUFFIX_HOSTRULE
used by c89/cc/c++ 82
_PLIB_PREFIX
used by c89/cc/c++ 82
_PMEMORY
used by c89/cc/c++ 83
_PMSGS
used by c89/cc/c++ 83
_PNAME
used by c89/cc/c++ 83
_PSUFFIX
used by c89/cc/c++ 83
_PSUFFIX_HOST
used by c89/cc/c++ 83
_PSYSIX
used by c89/cc/c++ 83
_PSYSLIB
used by c89/cc/c++ 84
_SLIB_PREFIX
used by c89/cc/c++ 84
_SNAME
used by c89/cc/c++ 84
_SSUFFIX
used by c89/cc/c++ 84
_SSUFFIX_HOST
used by c89/cc/c++ 84
_SSYSLIB
used by c89/cc/c++ 84
_STEPS
used by c89/cc/c++ 84
_SUSRLIB
used by c89/cc/c++ 85
_TMPS
used by c89/cc/c++ 85
_WORK_DATACLAS
used by c89/cc/c++ 85
_WORK_DSNTYPE
used by c89/cc/c++ 86
_WORK_MGMTCLAS
used by c89/cc/c++ 86
_WORK_SPACE
used by c89/cc/c++ 86
_WORK_STORCLAS
used by c89/cc/c++ 86
_WORK_UNIT
used by c89/cc/c++ 86
_XSOMDIRS
used by c89/cc/c++ 86
_XSOMLIBS
used by c89/cc/c++ 86
_XSUFFIX
used by c89/cc/c++ 86

environment variable *(continued)*
_XSUFFIX_HOST
used by c89/cc/c++ 87
allnet
used by mailx 350, 355
append
used by mailx 344
ask
used by mailx 350
askbcc
used by mailx 350
asksub
used by mailx 350
autoprint
used by mailx 338, 345, 350
bang
used by mailx 347, 350
CDPATH
description of 507
used by cd 101
used by vi 712, 716
cmd
used by mailx 343, 350
COLUMNS
description of 507
used by ed 232
used by ls 331
used by more 390
used by pg 446, 449
used by ps 464
used by sed 480
used by shedit 513
used by vi 716
conv
used by mailx 337, 355
crt
used by mailx 343, 351
dead
used by mailx 351
description of 507
displaying 235, 482
displaying the value of a 455
dot
used by mailx 351
EDITOR
description of 507
used by crontab 147
used by mailx 346, 351
used by shedit 513
ENV
description of 486, 840
used by sh 489
used by vi 716
ERRNO
description of 507
escape
used by mailx 346, 351

environment variable *(continued)*
EXINIT
description of 840
used by vi 715, 716
FCEDIT
description of 507
used by fc, history, r 251
flipr
used by mailx 343, 351
folder
used by mailx 340, 344, 351
FPATH
description of 507
header
used by mailx 351
HISTFILE
description of 507
used by fc, history, r 250, 251
HISTSIZE
description of 507
used by fc, history, r 250, 251
hold
used by mailx 338, 351
HOME
description of 507
used by cd 101
used by crontab 146, 147
used by mailx 349
used by vi 712, 715, 716
IFS
description of 508
used by read 466, 467
used by sh 488
ignore
used by mailx 337, 351
ignoreeof
used by mailx 351
indent
used by mailx 352
indentprefix
used by mailx 346, 352
keep
used by mailx 352
keepsave
used by mailx 338, 352
LANG
description of 508
LC_ALL
description of 840
LIBPATH
description of 508, 840
used by c89/cc/c++ 70
LINENO
description of 508
LINES
description of 508
used by more 390

environment variable *(continued)*
LINES *(continued)*
used by pg 447, 449
used by vi 715
lister
used by mailx 352
LOCPATH
description of 508
LOGNAME
description of 508
used by crontab 146, 147
used by logname 324
used by mailx 349
LPDEST
description of 840
used by lp 325, 326
MAIL
description of 840
used by mailx 349
MAILCHECK
description of 508
MAILDIR
used by mailx 349
MAILER
used by calendar 95
MAILPATH
description of 508
MAILRC
used by mailx 349, 352
mailserv
used by mailx 352, 355
MAKEFLAGS
description of 840
used by mail 334
used by make 372, 374
MAKESTARTUP
description of 840
used by make 356, 372, 374
MANPAGER
description of 840
used by man 379, 380
MANPATH
description of 508
used by man 379, 380
mbox
description of 508
used by mailx 352
metoo
used by mailx 352
MORE
description of 840
used by more 390
NLSPATH
description of 508
OLDPWD
description of 508
used by cd 101

environment variable *(continued)*
onehop
used by mailx 352, 355
OPTARG
description of 840
used by getopts 273
OPTIND
description of 840
used by getopts 273
outfolder
used by mailx 352
page
used by mailx 343, 352
pager 840
used by mailx 343, 353
used by man 379
PATH
description of 508
used by awk 40
used by crontab 146, 147
used by vi 716
PID
description of 508
PRINTER 841
used by lp 325, 326
prompt
used by mailx 353
PS1
description of 509
PS2
description of 509
used by read 467
PS3
description of 509
PS4
description of 509
PWD
description of 509
used by cd 101
quiet
used by mailx 353
RANDOM
description of 509
READONLY
descripion of 841
used by vi 689, 708, 716
record
used by mailx 336, 342, 343, 353
REPLY
description of 841
used by read 466, 467
replyall
used by mailx 353, 355
RESOLVER_CONFIG
description of 841
save
used by mailx 353

environment variable *(continued)*
screen
used by mailx 342, 353
SECONDS
description of 509
sendmail
used by mailx 353, 355
sendwait
used by mailx 353, 355
setting 235
SHELL
description of 509
used by at 21
used by awk 44
used by crontab 146
used by ed 232
used by mailx 344, 353
used by make 374
used by vi 709, 716
showto
used by mailx 353
sign
used by mailx 346, 353, 354
STEPLIB
description of 509
SYSEXEC
used by tso 641
SYSPROC
used by tso 641
term
used by mailx 354
used by more 390
used by tabs 547
used by talk 550
used by touch 634
used by vi 689, 715, 716
terminfo 290, 628
used by tabs 547
used by talk 550, 634
used by vi 716
TMOUT
description of 509
TMP
used by exrecover 246
TMP_VI
used by vi 717
TMPDIR
description of 509
used by ar 16
used by ed 232
used by man 379
used by pg 449
used by sort 525
used by vi 716
toplines
used by mailx 344, 354

environment variable *(continued)*
TSOALLOC
used by tso 641
TSOPREFIX
used by tso 642
TSOPROFILE
used by tso 642
TZ 837
description of 510
used by at 21
used by cron 143
used by crontab 147
used by date 156
used by ls 331
used by mail 334
used by pr 452
used by touch 632
used by uulog 675
used by uustat 681
used by mailx 341
used to specify system and operational information to c89/cc/c++/cxx 74
VISUAL
description of 510
used by mailx 345, 347, 354
used by shedit 513
X_ADDR
description of 841
X_SITE
description of 841
used by ocsconfig 417
X_SLATE
description of 841
Environment Variables section
explanation of 4
equivalence class 798
ERRNO environment variable
description of 507
escape character
displaying current settings 775
specifying the 768
turning off display for settings 775
escape environment variable
used by mailx 346, 351
ESCAPE option of OMVS command 768
escape sequences 495
escaping characters 495
eval shell command 236
evaluate
arguments as expression 243
arguments in the current shell environment 236
arithmetic expression 305
shell expressions 121
ex commands
special characters 710
ex mode
commands issued from 700
current position pointer 690
entering 700
starting session in 689
ex shell command 237
creating tag files for the 150
ex text editor
using the 237
Examples section
explanation of 4
exception condition
trapping 637
exec shell command 238
executable file
creating 64
displaying symbol table 407
execute
commands on remote sites 683
execute files 834
EXINIT environment variable
description of 840
used by vi 715, 716
exit code
returning a nonzero 249
exit shell command 240
exit shell subcommand 511
exit status
returning values of 0 639
Exit Values section
explanation of 6
expand
compressed data written by Lempel-Ziv compression 655
tabs to spaces 241
expand shell command 241
export
aliases 12
environment variables 242
export shell command 242
expr operators 243
expr shell command 243
expression 797
See also basic expression
See also regular extended expression
bracket 797
character class 798
evaluating 243, 305
handling, for the dbx command 159
printing tracing information 200
expression values
printing 184
exrecover shell daemon 245
extattr shell command 249
extended attributes
APF | NOAPF 813

extended attributes *(continued)*
PROGCTL|NOPROGCTL 814
SHAREAS | NOSHAREAS 814
SHARELIB | NOSHARELIB 814
extended regular expression
explanation of 797
list of commands using 800
external link
creating 313
identifying 328, 329
ln 314
extract
components from archives 434
contents of archive files 140

F

false shell command 249
fc shell command 250
FCEDIT environment variable
description of 507
used by fc, history, r 251
fg shell command 253
fgrep shell command 274
Fibonacci sequence 506
field (awk variable) 27
FIFO special files
creating 385, 386
fignore shell variable
description of 589
file
allocating
using the BPXBATCH command 738
using the BPXBATCH program 803
backing up
archive files 552
backup 552
binary
decoding transmitted 672
encoding for transmission 673
browsing, with BPXBATCH 783
browsing, with the ISPF shell 739
calculating
byte counts 117, 543
checksum 117, 543
changing
access permission of 109
access times 631
audit attributes 103
audit flags 103
group owners 108
groups 113
modes 109
modification times 631
owners 113
source 172
using diff output 429
file *(continued)*
comparing two 214
with the cmp command 118
with the diff command 214
with the ISHELL command 739
with the OSHELL command 783
with the sum command 543
compressed
displaying 445
compressing
using Lempel-Ziv compression 124
concatenating lines into standard output 427
converting 207
from ASCII to EBCDIC 208
from EBCDIC to ASCII 208
copying
archive files 552
between sites 667
between two files 748
to target named by the last argument on command line 130
to users on remote systems 682
with BPXBATCH 783
with data conversion 207
with the ISPF shell 739
creating
character special files for file systems 742
directories for 740
FIFO special 386
links to 313
with BPXBATCH 783
with the ISPF shell 739
deleting
information from 533
with BPXBATCH 783
with the ISPF shell 739
displaying 178
attributes of 739, 783
compressed 445
first part 277
interactively 446
last part of the 548
lines common to two files 122
names 645
page by page 387
specified number of lines in source 175
dumping to standard output 420
editing, with BPXBATCH 783
editing, with the ISPF shell 739
expanding compressed files 655
finding one that meets specified criteria 256
formatting in paginated form 450
instructions in a source 178
lines
numbering 405
list of
building, with the OSTEPLIB command 784

file *(continued)*
listing
attributes 327
names 327
maintaining
interdependent 355
program-generated 355
manipulating repeated lines 657
merging corresponding or subsequent lines of files 427
misspelled words
looking for 527
mounting file systems 743
moving 393
names
displaying 645
naming, with BPXBATCH 783
naming, with the ISPF shell 739
object
displaying symbol table of an 407
output tags
used by ctags 151
passing small amounts to 226
processing 25
reading dbx subcommands from 195
removing
information from 533
renaming 393
running
object files, with previous arguments 188
with the ISHELL command 739
with the OSHELL command 783
searching
backward for patterns 160
for specified patterns 274
for text strings 739, 783
forward for patterns 161
given file hierarchies 256
hierarchy 256
sending
paginated files to printer 450
to other users 726
setting
destination time 132
groups 113
modification time 132
owners 113
showing differences between two 214
sorting
in topological order 644
splitting 530
summarizing use of space 224
text
comparing two 214
concatenating 98
counting items in 721
displaying 98
file *(continued)*
text *(continued)*
finding information in 25
finding strings in 797
retrieving information from 25
splitting 148
transfers
displaying information about 674
uncompressing
Huffman-coded 445
words
looking for misspelled 527
file cache
managing 256
file-creation permission-code mask
setting or returning 651
file descriptor
closing 238
copying 238
opening 238
file formats
cpio 821
list of 821
magic 822
pax 824
queuedefs 826
tags 827
tar 828
utmpx 830
file mode creation mask
setting or returning 651
filename
deleting trailing parts 220
expanding on command line 226
file owner
group
GID(group) 814
UID(owner) 814
file recovery daemon for vi (exrecover) 245
file shell command 254
using the magic file 822
file space
summarizing use of 224
file subcommand for dbx command 172
file systems
hierarchical
mounting 743
unmounting 787
TFS
mounting 747
unmounting 787
file transfer
daemon for (uucico) 665
requests
processing, with the uucico daemon 665

file type
determining the 254
filec shell variable
description of 589
filename
generation 503
FILENAME built-in variable for awk 31
files 255
See also file
/bin/mail
used by calendar 95
/dev/mt/0m
used by tar 553
/dev/ocsadmin
used by ocsconfig 418
/dev/ocslock
used by ocsconfig 418
/dev/tty 452
/etc/auto/master
used by automount 23
/etc/banner
used by lm 311
/etc/csh.cshrc
used by tcsh login 557
/etc/csh.login
used by tcsh login 557
/etc/inetd.conf
used by the inetd daemon 283
/etc/inetd.pid
used by the inetd daemon 286
/etc/lm.rc
used by lm 311
/etc/magic
used by file 255
/etc/mailx.rc
used by mailx 337, 348, 354
/etc/profile 510
used by the login shell 486
/etc/rc
used by lm 311
/etc/recover
used by exrecover 246
/etc/recover/$LOGNAME/VIn*
used by exrecover 246
/etc/recover/$LOGNAME/VIt*
used by exrecover 246
/etc/services
used by lm 311
used by ocsconfig 418
/etc/startup.mk
used by make 374
/etc/suid_us.profile
used by sh 510
/etc/system
used by ocsconfig 418
/etc/utmpx
used by who 724
files *(continued)*
/etc/wtmp
used by who 724
/etc/yylex
used by lex 308
/etc/yylex.c
used by lex 306
/etc/yyparse.c
used by yacc 733
/tmp
used by ar 16
used by fc, history, r 251
/tmp/e*
used by ed 232
/tmp/sh*
description of 510
/tmp/stm*
used by sort 525
/tmp/VIl*
used by exrecover 246
/tmp/VIn*
used by exrecover 246
/tmp/VIt*
used by exrecover 246
/usr/lib/config
used by uucc 664
/usr/lib/cron/at.allow
used by cron 144
/usr/lib/cron/at.deny file
used by cron 144
/usr/lib/cron/cron.allow
used by cron 145
/usr/lib/cron/cron.deny
used by cron 145
/usr/lib/cron/queuedefs
used by cron 145
/usr/lib/hash
used by spell 528, 529
/usr/lib/hashb
used by spell 528, 529
/usr/lib/lib.b
used by bc 59
/usr/lib/libl.a
used by lex 308
/usr/lib/lwords
used by spell 528, 529
/usr/lib/uucp
used by uucc 664
/usr/lib/uucp/config
used by uucc 664, 665
used by uucico 666
used by uucp 670
used by uulog 675
used by uuname 676
used by uupick 678
used by uustat 681
used by uuto 682

files *(continued)*
/usr/lib/uucp/config *(continued)*
used by uux 686
used by uuxqt 687
/usr/lib/uucp/devices
used by uucc 664
/usr/lib/uucp/dialcodes
used by uucc 665
/usr/lib/uucp/dialers
used by uucc 664
/usr/lib/uucp/permissions
used by uucc 665
/usr/lib/uucp/systems
used by uucc 664
/usr/man/%L/man¬0–9l/*.book
used by man 379
/usr/man/%L/whatis
used by man 380
/usr/spool/.Sequence
used by uucp 670
/usr/spool/cron
used by cron 144
/usr/spool/cron/atjobs
used by cron 144
/usr/spool/cron/crontabs
used by cron 144
/usr/spool/cron/log
used by cron 144
/usr/spool/cron/pid
used by cron 144
/usr/spool/locks
used by uucico 666
/usr/spool/uucp
used by uulog 675
used by uustat 681
/usr/spool/uucp/.Status
used by uucico 666
/usr/spool/uucp/.Sequence
used by uux 686
used by uuxqt 687
/usr/spool/uucp/.Status
used by uustat 681
/usr/spool/uucp/.Xqtdir
used by uuxqt 687
/usr/spool/uucp/LOGFILE 666
used by uucico 666
used by uucp 670
used by uulog 675
used by uux 686
/usr/spool/uucp/site
used by uux 686
used by uuxqt 688
/usr/spool/uucppublic
used by uucp 670
used by uuto 683
/usr/spool/uucpublic
used by uupick 678

files *(continued)*
/usr/spool/uucpublic/receive
used by uupick 678
/var/man/%L/entry.¬0–9l/*.bookname
used by man 379
.dbxinit
used by dbx 158
.dbxsetup
used by dbx 158
.exrc
used by vi 715
.profile 510
$HOME / .sh_history
used by fc, history, r 250
$HOME/.exrc
used by vi 715
$HOME/mbox
used by mailx 354
used by make 336
$MAILRC
used by mailx 354
$TMPDIR/pg*
used by pg 449
a.out
used by dbx 160
calendar 96
dead.letter
used by mail 334, 335
used by mailx 346
ed.hup
used by ed 232
extended attributes
displaying 249
HOME/.profile
used by tcsh login 557
used by the login shell 486
l.output
used by lex 307, 308
lex.yy.c
used by lex 308
MAILDIR
used by mailx 354
mailrc
used by mailx 348
MapName
used by automount 24
mbox
used by mail 335
used by mailx 337
pk$*
used by unpack 662
queuedefs
used by cron 143
rsh
used by the sh command 486
sh_history 510

files *(continued)*
TCPIP.ETC.SERVICES
used by lm 311
used by ocsconfig 418
terminfo.src
used by tic 628
y.output
used by yacc 732
y.tab.c
used by yacc 732
y.tab.h
used by yacc 732
Files section
description of 6
filex
/etc/rs
used by automount 23
filter
numbering lines in a file 405
passing small amounts to 226
filter out
repeated lines in a file 657
find 797
files that match specified criteria 256
group affiliation of invoking processes 282
identical lines within files 122
patterns, using regular expressions 797
spelling errors 527
strings, using regular expressions 797
user identity of invoking processes 282
find shell command 256
fixed records
converting from variable records 208
fixed to variable-record conversion 209
flipr environment variable
used by mailx 343, 351
floating-point registers
displaying values of 187
FNR built-in variable for awk 31
fold shell command 261
folder environment variable
used by mailx 340, 344, 351
for loop
exiting from, in a shell script 63
for shell subcommand 488
format files in paginated form 450
Format section
explanation of 1
formatted output
writing 456
forward retrieve function of OMVS command 770
FPATH environment variable
description of 507
fpath search 494
free space
displaying amount of 212
fullword
definition of, for vi 693
func subcommand for dbx command 172
function
changing 172
explanation of 54
listing 205
moving down the stack 170
moving up the stack 203
printing tracing information for 200
unsetting values and attributes of 663
function key
customizing settings for 769
deactivating 771
displaying current settings 775
list of defaults 772
setting
controlling display of 771
setting up
to control display of the function key settings 771
to enter subcommand mode 771
to enter TSO/E command mode 772
to return from subcommand mode to shell 771
to scroll data backward 772
turning off display of 775
function shell subcommand 494
fuser shell command 263
FWDRETR function key for OMVS command 770
FWDRETR option of OMVS command 770

G

gencat shell command 264
preprocessing message source files for 381
general-purpose registers
displaying values of 187
generate
filenames 503
programs for lexical tasks 306
get
configuration values 267
contents of archive files 140
messages 380
getconf shell command 267
getopts shell command 271
gid shell variable
description of 589
glob characters 503
glob patterns 503
gmacs 483
enabling, with the EDITOR environment variable 507
GMT (Greenwich Mean time)
used by the TZ environment variable 837

gotoi subcommand for dbx command 173
Greenwich Mean Time (GMT)
used by the TZ environment variable 837
grep shell command 274
group
changing 402
setting 739
group affiliation
finding 282
returning 282
group owner
changing 108
setting 108
group recipe
explanation of 356
group shell variable
description of 589
GROUP tcsh environment variable
description of 598

H

HALFSCR function key for OMVS command 770
HALFSCR subcommand of OMVS command 774
hangup 511
hash shell command 276
head shell command 277
header environment variable
used by mailx 351
header line 338
HELP function key for OMVS command 770
help information
refreshing 771
scrolling
backward 769, 774
forward 769, 771, 774, 776
half a screen forward 770, 774
help subcommand for dbx command 174
HELP subcommand of OMVS command 774
here document 502
HFS
invoking
vfs_pfsctl HFS functions 126
mounting
from the shell 391
unmounting
from the shell 660
HFS (hierarchical file system) 411, 748
See also hierarchical file system (HFS)
hide
data entered on the shell command line 770, 774
OMVS command input area 770, 774
HIDE function key for OMVS command 770
HIDE subcommand of OMVS command 774
hierarchical file system (HFS)
browsing files in the 748
with the obrowse shell command 411
hierarchical file system (HFS) *(continued)*
copying
between two files 748
data sets into MVS data sets 754
directories to PDS or PDSE 758
files to PDS or PDSE 758
MVS data set members 777, 780
creating 739
editing files
with the oedit shell command 423
with the OEDIT TSO/E command 754
mounting 739
running
C/MVS executable files from TSO/E sessions 738
unmounting 739
hierarchical file systems
mounting 743
TFS file systems 743
unmounting 787
histchars shell variable
description of 589
histdup shell variable
description of 589
HISTFILE environment variable
description of 507
used by fc, history, r 250, 251
histfile shell variable
description of 589
histlit shell variable
description of 589
history
editing 513
history file
processing 250
truncating the 250
history list
displaying commands in a 174
processing for commands 250
history shell variable
description of 589
history storage file (sh_history) 510
history subcommand for dbx command 174
HISTSIZE environment variable
description of 507
used by fc, history, r 250, 251
hold buffer 476
hold environment variable
used by mailx 338, 351
home directory 507
HOME environment variable
description of 507
used by cd 101
used by crontab 146, 147
used by mailx 349
used by vi 712, 715, 716

home shell variable
description of 589
HOME tcsh environment variable
description of 598
HOME/.profile file
used by tcsh login 557
used by the login shell 486
HOST tcsh environment variable
description of 598
HOSTTYPE tcsh environment variable
description of 598
HPATH tcsh environment variable
description of 598
Huffman coding
compressing files with 424, 661
uncompressing files 445
hyphen
explanation of 1

I

iconv shell command 280
id shell command 282
identifier
displaying qualifications of 207
identifier (awk variable) 27
identify shell names 646
if shell subcommand 488
IFS environment variable
description of 508
used by read 466, 467
used by sh 488
ignore environment variable
used by mailx 337, 351
ignore subcommand for dbx command 175
ignoreeof shell variable
description of 589
ignoreof environment variable
used by mailx 351
illegal byte sequence
in DBCS strings 9
implicitcd shell variable
description of 589
in shell subcommand 492
in/out file archives
copying 140
indent environment variable
used by mailx 352
indentprefix environment variable
used by mailx 346, 352
inetd daemon 283
handling of requests by uucpd 671
infocmp shell command 286
input
passing small amounts to filter or file 226
input file
concatenating lines 427
displaying 446
printing 325
input mode 338
inputmode shell variable
description of 590
insert mode 689
instruction
displaying 178
running 196
instruction register
displaying values of 187
interactive shell 487, 558
Interactive System Productivity Facility 411, 748
See also ISPF
interactive text editor (vi) 688
intercept
abnormal conditions and interrupts 637
signals 637
interdependent file
maintaining 355
internal field separator 508
internationalization
explanation of 817
interpret
ASA/FORTRAN carriage control 17
interpret command names 722
Interprocedural Analysis (IPA) optimization
enabling 69
explanation of 71
interprocess communication facility status
reporting the 292
interrupt
trapping abnormal 637
invalid byte sequence
in DBCS strings 9
invoke
BPXBATCH
with OSHELL 783
BPXBATCH program 803
editor 171
HFS functions 126
ISPF shell
with ISHELL 739
OS/390 shell 764
shell 485, 557
TSO/E command mode 776
utilities, ignoring the SIGHUP signal 410
invokes
vfs_pfsctl HFS functions 126
IPA (Interprocedural Analysis) optimization
enabling 69
explanation of 71
ipcrm shell command 291

ipcs shell command 292
ISHELL TSO command 739
ISPF (Interactive System Productivity Facility)
browsing files
with the obrowse shell command 411
with the OBROWSE TSO/E command 748
editing files
with the oedit shell command 423
with the OEDIT TSO/E command 754
entering TSO/E commands from 737
invoking the shell
with ISHELL 739
italic typeface
explanation of 1, 2

J

JCL (job control language)
example of, using the BPXCOPY program 815
job
ending 301
moving
from background to foreground 253
to background 61
restarting a suspended 253
returning list of, in current session 298
running in background 61
scheduling background 145
waiting for it to end 720
jobs shell command 298
join shell command 299
join two databases 299

K

keep environment variable
used by mailx 352
keepsave environment variable
used by mailx 338, 352
key
sorting 523
kill shell command 301

L

l.output file
used by lex 307, 308
LALR(1) grammar
converting 731
LANG environment variable 4, 817
description of 508
LANG tcsh environment variable
description of 598
last lines of file 548
LC_ALL environment variable 4, 817
description of 840
LC_COLLATE environment variable 817
LC_CTYPE environment variable 817
LC_CTYPE tcsh environment variable
description of 598
LC_MESSAGES environment variable 4, 817
LC_MONETARY environment variable 817
LC_NUMERIC environment variable 817
LC_SYNTAX environment variable 817
LC_TIME environment variable 817
Lempel-Ziv compression 142, 440
compressing data with 124
uncompressing data 655
let shell command 305
lex shell command 306
lex.yy.c file 308
lexical analyzer 306
lexical syntax
reading description of 306
lexical tasks
generating programs for 306
LIBPATH environment variable
description of 508, 840
used by c89/cc/c++ 70
library
creating 14
maintaining 14
making 373
library of objects
displaying symbol table 407
Limits section
explanation of 6
line
breaking into shorter lines 261
changing next line to be displayed 179
numbering, in a file 405
reading from standard input 466
line editor (ex) 689
line shell command 309
LINENO environment variable
description of 508
lines
counting 721
LINES environment variable
description of 508
used by more 390
used by pg 447, 449
used by vi 715
LINES option of OMVS command 769
LINES tcsh environment variable
description of 598
link
creating, for files 313
link shell command 310, 313
link-edit
OS/390 C and OS/390 C++ object files 64

links
creating 310
list
active procedures and functions 205
file attributes 327
filenames 327
files in directories
with ISHELL 739
with OSHELL 783
instructions in program 178
process IDs 263
variables and their attributes 647
list mode 434
list subcommand for dbx command 175
lister environment variable
used by mailx 352
listfiles subcommand for the dbx command 176
listflags shell variable
description of 590
listfuncs subcommand for dbx command 177
listi subcommand for dbx command 178
listjobs shell variable
description of 590
listlinks shell variable
description of 590
listmax shell variable
description of 590
listmaxrows shell variable
description of 590
lm shell command 311
load characteristics
displaying 179
local environment
defining 320
local spawn
BPXBATSL 738, 803
locale 4, 308
converting source definitions for categories 320
displaying information about 317
giving it control over a category 817
switching 9
locale shell command 317
localedef shell command 320
localization
categories of 817
explanation of 817
Localization section
explanation of 4
locate identical lines within files 122
LOCPATH environment variable
description of 508
log information
displaying about UUCP events 674
log messages 322
logger shell command 322
logging in 486, 557
login accounting information
storing 830
login information
displaying 723
login monitor
starting 311
login monitor daemon 311
login name
returning 324
login password
changing the 426
login shell
description of 486, 557
system profile for the 510
truncating history files 250
user profile for the 510
loginsh shell variable 590
LOGNAME environment variable
description of 508
used by crontab 146, 147
used by logname 324
used by mailx 349
logname shell command 324
logout shell variable
description of 590
loop
exiting from, in a shell script 63
skipping to the next iteration of a 129
lowercase
converting to uppercase 209
lowercase letters 1
lp shell command 325
LPDEST environment variable
description of 840
used by lp 325, 326
lpstat shell command 326
ls
in a sysplex 331
ls shell command 327

M

MACHTYPE tcsh environment variable
description of 598
macro definitions 358
macro modifiers 359
magic file format 822
used by the file command 254
magic number
#! 807
mail
reading 332
sending 332
sending and receiving 335

MAIL environment variable
description of 840
used by mailx 349
mail shell command 332
mail shell variable
description of 590
MAILCHECK environment variable
description of 508
MAILDIR environment variable
used by mailx 349
MAILER environment variable
used by calendar 95
MAILPATH environment variable
description of 508
MAILRC environment variable
used by mailx 349, 352
mailrc file
used by mailx 348
mailserv environment variable
used by mailx 352, 355
mailx environment variable
used by mailx 344
mailx shell command 335
maintain
library archives 14
program-generated and interdependent files 355
make
directories
for each named directory argument 383
with the MKDIR command 740
FIFO special files 385
libraries 373
make shell command 355
makefile 358
contents of 361
MAKEFLAGS environment variable
description of 840
used by mail 334
used by make 372, 374
MAKESTARTUP environment variable
description of 840
used by make 356, 372, 374
man page
displaying 378, 840
man shell command 378
manage
file caches 256
manipulate
dates 25
repeated lines 657
tar archive files 552
manipulation command (for vi) 691
MANPAGER environment variable
description of 840
used by man 379, 380
MANPATH environment variable
description of 508
used by man 379, 380
map subcommand for dbx command 179
MapName file
used by automount 24
mark name 692
master mode 665
match
strings of text in text file 797
matchbeep shell variable
description of 590
matching strings
searching for 274
mbox environment variable
description of 508
used by mailx 352
mbox file
used by mail 335
used by mailx 337
memory
displaying 168
merge
corresponding or subsequent lines of files 427
mesg shell command 380
message
allowing 380
broadcasting a 721
header line 338
logging 322
receiving 380
refusing 380
sending to other users 726
message catalog
creating 264
displaying 222
displaying messages from 223
editing 264
modifying 264
piping from mkcatdefs to gencat 475
message queue
removing 291
message source file
preprocessing 381
metacharacter
used in regular expressions 797
metarules
used by make 363
metoo environment variable
used by mailx 352
mkcatdefs shell command 381
mkdir shell command 383
MKDIR TSO/E command 740
mkfifo shell command 385
MKNOD TSO/E command 741

mode
changing 109
command 338
input 338
modification time 327
setting for destination files 132
modify
message catalogs 264
MORE environment variable
description of 840
used by more 390
more shell command 387
creating tag files for the 150
mount
a file system 391
hierachical file systems 743
hierarchical file systems (HFS) 739
TFS file systems 743
mount attributes
changing
from the shell 112
mount mode
changing the 785
mount shell command 391
MOUNT TSO/E command 743
move
current function down the stack 170
current function up the stack 203
files 393
jobs from background to foreground 253
positional parameters 519
move subcommand for dbx command 179
movement commands (for vi) 690
MsgFile.h
mkcatdef output file 381
multihop name 667
multinode name 667
multiple volume support 142, 439
multiprocess debugging
enabling or disabling 180
multproc subcommand for dbx command 180
mutex object
display list of 199
displaying list of 180
mutex subcommand for dbx command 180
mv shell command 393
MVS (Multiple Virtual Storage)
batch environment
running shell scripts and OS/390 C applications under 803
running shell scripts, shell commands and OS/390 C applications under 738
copying
data sets into HFS directories 780
data sets into HFS files 777
data sets to another member or file 748
HFS files to MVS data sets 754
MVS (Multiple Virtual Storage) *(continued)*
copying *(continued)*
sequential data sets into HFS directories 780
sequential data sets into HFS files 777
submitting batch jobs
using the BPXBATCH program 803

N

name of files
listing 327
name, user
displaying your 725
named pipe 386
See also FIFO special file
Native Language System Report 582
newgrp shell command 402
newline
counting 721
next subcommand for dbx command 181
nexti subcommand for dbx command 182
NEXTSESS function key for OMVS command 770
NEXTSESS subcommand of OMVS command 775
NF built-in variable for awk 31
nice shell command 404
nickname
creating 11
nl shell command 405
NLSPATH environment variable 4
description of 508
nm shell command 407
NO function key for OMVS command 771
NOALARM function key for OMVS command 769
NOALARM option of OMVS command 764
NOALARM subcommand of OMVS command 775
NOAUTOSCROLL function key for OMVS command 769
NOAUTOSCROLL option of OMVS command 764
NOAUTOSCROLL subcommand of OMVS command 775
nobeep shell variable
description of 590
noclobber shell variable
description of 590
NODBCS option of OMVS command 767
NOECHO function key for OVMS command 770
NOECHO option of OMVS command 767
NOECHO subcommand of OMVS command 775
nogob shell variable
description of 591
NOHIDE function key for OMVS command 770
NOHIDE subcommand of OMVS command 775
nohup shell command 410
nokanji shell variable
description of 591

nonfunctional commands 152
cancel 96
lpstat 326
nonomatch shell variable
description of 591
nonsupported commands
cu 152
nonzero exit code
returning 249
NOPFSHOW function key for OMVS command 771
NOPFSHOW option of OMVS command 772
NOPFSHOW subcommand of OMVS command 775
NOREBIND tcsh environment variable
description of 598
NOSHAREAS option of OMVS command 773
nostat shell variable
description of 591
Notices 845
notify shell variable
description of 591
NR built-in variable for awk 31
null command 121
number
lines in a file 405

O

object file
displaying the symbol table of an 407
loading for execution 183
managing 355
running with previous arguments 188
object library
displaying symbol table 407
object manipulator commands (for vi) 695
object subcommand for dbx command 183
obrowse shell command 411
OBROWSE TSO/E command 747
obtain
crontab entries 145
obtain options and their arguments 271
OCOPY TSO/E command 748
OCS (Outboard Communications Server) 311
See also Outboard Communications Server (OCS)
OCS daemon 414
ocsconfig shell command 412
octal dump 420
od shell command 420
oedit shell command 423
OEDIT TSO/E command 753
OGET TSO/E command 754
OGETX TSO/E command 758
OHELP TSO/E command 762
OLDPWD environment variable
description of 508
used by cd 101
OMVS command
list of subcommands 774
OMVS command input area
hiding 774
hiding or unhiding 770
unhiding 775
OMVS interface
running commands from the shell using the 640
onehop environment variable
used by mailx 352, 355
online reference manual
printing entries 378
searching for entries 378
onload subcommand for dbx command 183
open file descriptors 238
OPEN function key for OMVS command 771
OPEN subcommand of OMVS command 775
operator
control 493
description of 493
redirection 493
OPTARG environment variable
used by getopts 273, 840
OPTIND environment variable
used by getopts 273, 840
options
explanation of 1
obtaining from a list of parameters 271
order of 1
Options section
explanation of 4
OPUT TSO/E command 776
OPUTX TSO/E command 780
order of items on command line 3
order of options 1
OS/390
submitting batch jobs
using the BPXBATCH command 738
OS/390 C and OS/390 C++ source files
using the c89 command to compile, assemble, and link-edit 64
OS/390 UNIX System Services
displaying publications online 762
giving TSO/E users access to 739, 783
invoking the shell 764
managing functions with the ISPF shell
with ISHELL 739
publications
on CD-ROM xxxi
softcopy xxxi
setting up functions with the ISPF shell
with ISHELL 739
OSHELL REXX exec 809
OSHELL TSO command 783
OSTYPE tcsh environment variable
description of 598

Outboard Communications Server (OCS)
configuring objects 412
querying objects 412
starting login monitor 311
unconfiguring objects 412
outfolder environment variable
used by mailx 352
output file
copying standard input to each 623
output stream
cloning 623
output tags file
used by ctags 151
output, formatted
writing 456
overlay commands 238
owd shell variable
description of 591

P

pack shell command 424
page environment variable
used by mailx 343, 352
pager environment variable
description of 840
used by mailx 343, 353
used by man 379
paginated file
formatting 450
printing 450
parameter
positional
description of 496
setting 482
shifting 519
unsetting 482
special
description of 496
parameter substitution 496, 583
parent process
returning to the 240
parse
utility options 271
partitioned data set (PDS) 758
See also PDS
partitioned data set extended (PDSE) 758
See also PDSE
pass
command to shell for execution 194
small amounts of input to filter or file 226
passwd shell command 426
password
changing the 426
paste shell command 427
patch shell command 429
PATH environment variable
description of 508
used by awk 40
used by crontab 146, 147
used by vi 716
path search 494
path shell variable
description of 591
PATH tcsh environment variable
description of 598
pathchk shell command 432
pathname
checking for validity and portability 432
displaying 465
returning
directory components of 220
nondirectory components of 42
pattern
finding, using regular expressions 797
rules for 492
searching 274
backward for a 160
forward for a 161
pattern buffer 476
pax file format 824
pax shell command 433
pcat shell command 445
PDS (partitioned data set)
copying
HFS directories or file to a 758
members from MVS to files 780
members to files 777
PDSE (partitioned data set extended)
copying
HFS directories or files to a 758
members from MVS to files 780
members to files 777
permission bits
of files, setting 813
permissions 329
permuted index xv
PF keys
showing at the bottom of the screen 772
PFSHOW function key for OMVS command 771
PFSHOW option of OMVS command 772
PFSHOW subcommand of OMVS command 775
pg shell command 446
PID environment variable
description of 508
pipe
creating 489
output from mkcatdefs to gencat 475
piped file
displaying 446

pipeline 489
pk$* file
used by unpack 662
placeholder information in commands 2
Portability section
explanation of 6
positional parameter 519
See also parameter, positional
POSIX.1 standard parameter names 267
POSIX.2 standard parameter names 269
pr shell command 450
preprocess
message source files 381
prevent changes to values of the name argument 468
PREVSESS function key for OMVS command 771
PREVSESS subcommand of OMVS command 775
print
arguments 454
expression values 184
formatted output 456
input files 325
paginated files 450
sections of online reference manuals 378
terminal entries in the terminfo database 97
terminfo database entries 286
tracing information 200
print queue
requests
canceling 96
displaying status of 326
print shell command 454
print subcommand of for command 184
printenv shell command 455
printer
sending files to 325
PRINTER environment variable
description of 841
used by lp 325, 326
printexitvalue shell variable
description of 591
printf shell command 456
priorities of running processes
changing 469
priority
running commands at a different 404
procedure
listing 205
printing tracing information for 200
process
changing priorities of running 469
displaying
status of 458
time accumulated 630
ending 301
returning
file-creation permission-code masks 651
process *(continued)*
returning *(continued)*
status of 458
sending signals to 301
setting
file-creation permission-code masks 651
resource limits 650
process IDs
displaying 263
process list
returning 298
processes IDs 263
processing
awk programs 25
command history list 250
processor
displaying 629
processor time 630
program
continuing execution 166
from stopping point 194
continuing execution without dbx control 168
debugging 157
delaying execution of 521
deleting stops and traces from 167
displaying
declarations of components 205
instructions 178
load characteristics 179
generating, for lexical tasks 306
managing 355
printing tracing information 200
running
object files 189
program instructions 196
source lines 196
to next instruction 182
to next source line 181
until return is reached 188
with previous arguments 188
stopping
at a specific location 198
when certain conditions are met 197
writing printouts created by 17
program counter address
changing 173
program file
displaying
list of functions 177
program-generated file
maintaining 355
prompt
continuation 509
string 509
prompt environment variable
used by mailx 353

prompt shell variable
description of 594
prompt subcommand of for command 184
prompt2 shell variable
description of 591
prompt3 shell variable
description of 591
promptchars shell variable
description of 591
ps shell command 458
PS1 environment variable
description of 509
PS2 environment variable
description of 509
used by read 467
PS3 environment variable
description of 509
PS4 environment variable
description of 509
public directories (UUCP)
searching 677
publications
on CD-ROM xxxi
softcopy xxxi
pushdsilent shell variable
description of 591
pushdtohome shell variable
description of 591
PWD environment variable
description of 509
used by cd 101
pwd shell command 465
PWD tcsh environment variable
description of 598

Q

query
ASCII/EBCDIC code pages for the terminal 105
OCS objects 412
queuedefs file
used by cron 143
queuedefs file format 826
quiet environment variable
used by mailx 353
quiet mode
turning on 689
quit
sessions
ending 771
shell sessions 771, 775
QUIT function key for OMVS command 771
QUIT subcommand of OMVS command 775
quit subcommand of the for command 185
QUITALL function key for OMVS command 771
QUITALL option of OMVS command 775
QUITALL subcommand of OMVS command 775
quoting 495

R

RACF (Resource Access Control Facility) 11
RANDOM environment variable
description of 509
read
archive files 433, 552
contents of UUCP configuration files 664
cpio archives 140
data 207
dbx subcommands from file 195
description of lexical syntax 306
electronic mail 335
lines from standard input 466
mail 332
read mode 434
read shell command 466
read/write lock objects
displaying list of 185
READONLY environment variable
description of 841
used by vi 689, 708, 716
readonly shell command 468
readwritelock subcommand for dbx command 185
reason_code text
displaying 62
receive
electronic mail 335
messages 380
recexact shell variable
description of 591
recipe line
explanation of 356
recipes 363
explanation of 358
recognize_only_executables shell variable
description of 591
record environment variable
used by mailx 336, 342, 343, 353
record separator character 31
record subcommand for the dbx command 186
recovery daemon
for vi 245
redirection 3, 501
redirection operator 493
reenter commands 250
reference manual
online
printing entries 378
searching for entries 378
refresh
data 771

refresh *(continued)*
help information 771
REFRESH function key for OMVS command 771
refuse
messages 380
regexp 797
See also regular expression
registers subcommand for dbx 187
regular expression 701
composition of 797
concatenating to form a larger regular expression 799
examples 800
explanation of 797
features that apply to OS/390 shell commands 800
matching 274
supported by awk 28
used in vi 701
used to find patterns in files 797
used when finding strings in files 797
regular expression summary 702
reject lines common to two files 122
Related Information section
explanation of 7
remote site
running commands on 683
transferring data to 833
remote system
copying files to users on 682
remote systems
connecting to, with the uucico daemon 665
REMOTEHOST tcsh environment variable
description of 598
remount
specified file systems 785
remove
alias definitions 652
aliases 202
arguments 473
attributes of shell variables 663
attributes of variables and functions 663
breakpoints at addresses 165
crontab entries 146
directories 474
directory entries 473
duplicate files 657
files 473, 659
information from executable files 533
message queues 291
reverse line feeds 119
semaphore sets 291
shared memory identifiers 291
stops from programs 164, 167
traces from program 167
trailing part of filenames 220
values of variables and functions 663
remove *(continued)*
variables 203
removes
directory entries 659
rename files 393
renice shell command 469
replacement pattern summary 702
REPLY environment variable
description of 841
used by read 466, 467
replyall environment variable
used by mailx 353, 355
report
interprocess communication facility status 292
repeated lines in a file 657
request (file transfer)
processing, with the uucico daemon 665
rerun subcommand for dbx command 188
reset
ASCII/EBCDIC code pages for the terminal 105
reset access time 140
RESOLVER_CONFIG environment variable
description of 841
Resource Access Control Facility 11
See also RACF
restart suspended jobs 253
restricted shell 486, 487
retrieve
saved input lines by going backward 771
saved input lines by going forward 770
RETRIEVE function key for OMVS command 771
return
arguments from the shell 454
directory components of pathnames 220
file mode creation masks 651
from . (dot) scripts 471
from shell functions 471
from subcommand mode to shell session 775
group affiliation of invoking processes 282
list of jobs in current session 298
login names 324
nonzero exit codes 249
pathname of working directories 465
process status 458
to shell mode from TSO/3270 passthrough mode 767
to the parent process 240
to TSO/E 240
user's terminal name 645
user ID of person who entered commands 324
user identity of invoking processes 282
RETURN function key for OMVS command 771
return shell command 471
return subcommand for dbx command 188
return values of 0 639

reverse line feed
removing the 119
REXX
OSHELL 809
RLENGTH (awk built-in variable) 33
rlogin requests
handling 283
rlogind program 283, 471
rm shell command 473
rmdir shell command 474
rmstar shell variable
description of 592
root directory
changing 114
root file system
setting up directories for the 739
rprompt shell variable
description of 592
rsh file
description of 486
RSTART (awk built-in variable_ 33
run
C/MVS executable files
with the BPXBATCH command 738
with the BPXBATCH program 803
commands
after building an argument list 727
at a different priority 404
at a specified time 19
at specified dates and times 143
on remote sites 683
simple 123
when system is not busy 43
with the exec command 238
debug programs 157
files, with the ISHELL command 739
files, with the OSHELL command 783
from TSO/E sessions
C/MVS executable files 738
shell commands 738
shell scripts 738
object files with previous arguments 188
program instructions 196
programs 189
shell commands
with the BPXBATCH command 738
shell scripts
with the BPXBATCH command 738
with the BPXBATCH program 803
simple commands 123
source lines 173, 196
run subcommand for dbx 189
runcat shell command 475
running processes
changing priorities of 469
runtime macros 360

S

save environment variable
used by mailx 353
save messages 322
saved input line
retrieving by going backward 771
retrieving by going forward 770
savedirs shell variable
description of 592
savehist shell variable
description of 592
SBCS mode
specifying the 767
scale value 47
sched tcsh shell variable
description of 592
schedule
background jobs 145
screen editor (vi) 689
screen environment variable
used by mailx 342, 353
scroll
automatic
controlling 769, 774, 775
data 776
data backward 769, 771, 772, 774
data forward 769, 774
data half a screen forward 770, 774
help information backward 769, 774
help information forward 769, 771
help information half a screen forward 770
SCROLL function key for OMVS command 771
SCROLL subcommand of OMVS command 776
scrolling commands (for vi) 690, 691
search
backward for patterns 160
directories 204
files for text strings 739, 783
for entries in online reference manuals 378
for strings 274
forward for patterns 161
given file hierarchies 256
public UUCP directories 677
search path 494, 722
search rules 494, 722
SECONDS environment variable
description of 509
sections
meaning of, in command descriptions 1
sed noninteractive stream editor
starting the 476
sed shell command 476

select loop
exiting from, in a shell script 63
select shell subcommand 492
semaphore set
removing 291
send
electronic mail 335
files to printer 325
mail 332
messages
to other users 726
paginated files to printer 450
signals to processes 301
sendmail environment variable
used by mailx 353, 355
sendwait environment variable
used by mailx 353, 355
sequential data set
copying to files 777, 780
session
specifying number to be started 773
starting
in ex mode 689
in vi mode 689
switching
to the next higher-numbered one 770, 775
to the previous (lower-numbered) session 771, 775
SESSION option of OMVS command 773
session, returning list of jobs in 298
set
ASCII/EBCDIC code pages for the terminal 105
command options 482
commands to be run at a specified time 19
export attributes for variables 242
file mode creation masks 651
group owners 108
positional parameters 482
priorities of running processes 469
process limits 650
terminal options 534
terminal tab stops 545
timezones with the TZ environment variable 837
set option variables 710
set shell command 482
set subcommand for dbx command 190
set up
directories for the root file system 739
existing groups 739
existing users 739
special files in the /bin directory 739
sh shell command 485
rsh file 486
sh subcommand for dbx command 194
sh_history file 510
SHAREAS option of OMVS command 773
shared HFS
changing file system mount attributes 112
displaying amount of free space 212
mounting a file system 391
using df 213
using ls 331
shared memory identifier
removing 291
shedit shell command 513
shell 738
access to, giving users 739, 783
alias command, and the 11
archive 506
arguments
evaluating 236
returning 454
arrays 504
command lines 11
command syntax 488
commands 488
running from TSO/E sessions 738, 809
running from TSO/E sessions, with OSHELL 809
using extended regular expressions 797
using regular expressions 797
comments 488
displaying variables 647
editing
interactive 513
ending 240
entering TSO/E commands from 737
evaluating
arguments 236
expressions 121
execution environment 504
removing aliases from 652
expressions
evaluating 121
functions
returning from 471
giving TSO/E users access to 739, 783
identifying names 646
interpreting command names 722
invoking 485, 557
keywords 11
messages
displaying online 762
OS/390 UNIX
giving TSO/E users access to 783
OS/390 UNIX System Services
giving TSO/E users access to 739
program
running in a separate address space 773
running in the TSO/E address space 773
removing attributes of shell variables 663
reserved word comments 489

shell *(continued)*
returning
arguments from 454
functions 471
running
programs in a separate address space 773
programs in a TSO/E address space 773
TSO/E commands from the 640
scripts
exits from loops in a 63
running from TSO/E sessions, with BPXBATCH 738
running from TSO/E sessions, with OSHELL 809
running, with the . (dot) command 221
running, with the BPXBATCH program 803
skipping to the next iteration of a loop 129
sessions 771
closing 770, 774
ending 775
returning from subcommand mode 775
starting 771, 775
variables
displaying 647
removing attributes of 663
rules for 504
shell command
chmount 112
mount 391
unmount 660
shell command line
hiding data so secure data can be entered 770, 774, 775
SHELL environment variable
description of 509
used by at 21
used by awk 44
used by crontab 146
used by ed 232
used by mailx 344, 353
used by make 374
used by vi 709, 716
shell mode
returning to, from TSO/3270 passthrough mode 767
shell pre-defined aliases
autoload 22
functions 262
history 279
integer 290
stop 531
suspend 545
shell tcsh shell variable
description of 592
shell variable
displaying
names of 482
values of 482
shift out
used in DBCS strings 9
shift positional parameters 519
shift shell command 519
SHLVL tcsh environment variable
description of 598
shlvl tcsh shell variable
description of 592
short circuit evaluation 29
show
amount of free space on file system 212
arguments
of programs 162
attributes and contents of symlink 739, 783
currently exported variables 242
declaration of program components 205
differences between two files 214
elapsed time for a command 629
environment variables 235
file attributes 739, 783
first part of files 277
information about locales 317
instructions in program 178
lines common to two files 122
list of active program and functions 205
list of files
of module 176
memory 168
names of
shell variables 482
variables in procedures 170
online OS/390 UNIX System Services publications 762
pathname of working directories 465
process IDs 263
process status 458
processors 629
qualifications
of given identifiers 207
of symbols 206
status of print queues 326
system time accumulated by commands 630
terminal names 645
user time accumulated by the shell 630
values of
shell variables 482
variables in procedures 170
showto environment variable
used by mailx 353
SIGHUP signal
ignored when utility is invoked 410
sign environment variable
used by mailx 346, 353, 354
signal
intercepting 637
sending to processes 301

signal *(continued)*
trapping
starting 163
stopping 175
signal handling 583
simple command 493
singlebyte character set (SBCS)
when you must use 8
singlebyte characters
converting 280
site
transferring data to remote 833
skip subcommand for dbx 194
skip to the next iteration of a loop in a shell script 129
slave mode 665
sleep shell command 521
socket
identifying file types 329
SOM (System Object Model), explanation 70
sort
files
in topological order 644
sort shell command 522
sort-merge utility
starting the 522
sorted files
locating 122
sorting keys 523
sound
3270 alarms 764, 769, 774
source definitions
converting for locale categories 320
source file
changing 172
displaying
instructions in a 178
specific number of lines 175
managing 355
source line
printing tracing information for 200
removing stops from 164
running 173, 196
specifying 173
source subcommand for dbx 195
SourceFile
mkcatdefs message file 381
space
compressing into tabs 656
expanding tabs to 241
special built-in commands 505
special built-in shell commands
. (dot) 221
break 63
colon. 121
continue 129
special built-in shell commands *(continued)*
dot (.) 221
eval 236
exec 238
export 242
readonly 468
return 471
set 482
shell 240
shift 519
trap 637
unset 663
special file
creating a FIFO 386
manipulating 739, 783
special parameter
description of 496
special target directives 368
specify
character conversion tables 764
command lines for another command 238
escape characters 768
number of sessions to be started 773
OS/390 UNIX character conversion table 764
source lines 173
that OMVS operate in DBCS mode 766
that OMVS operate in SBCS mode 767
that PF keys be shown at the bottom of the screen 772
spell shell command 527
spelling errors
looking for 527
split
files 530
text file 148
split output stream 623
split shell command 530
spool directory
/usr/spool/uucp 831
standard environment variables (stdenv)
allocating as files for passing input
using the BPXBATCH command 738
using the BPXBATCH program 803
standard error (stderr)
allocating as files for passing input
using the BPXBATCH command 738
using the BPXBATCH program 803
standard input (stdin)
allocating as files for passing input 738
using the BPXBATCH command 738
using the BPXBATCH program 803
closing 503
copying
data read from 635
to each output file 623
copying one line of 309

standard input (stdin) *(continued)*
explanation of 3
reading 98
reading lines from 466
standard output (stdout)
allocating as files for passing input
using the BPXBATCH command 738
using the BPXBATCH program 803
closing 503
copying standard output to each 623
dumping file to 420
explanation of 3
reading lines from 466
sending paginated files to 450
writing
arguments to 225, 456
configuration values to 267
start
login monitor 311
pending UUCP transfers 679
sessions
in ex mode 689
in vi mode 689
shell sessions 771, 775
sort-merge utility 522
statement
explanation of 35
status
displaying 458
of pending UUCP transfers 679
of print queues
displaying 326
status reporting 581
status subcommand for dbx command 195
status tcsh shell variable
description of 592
stdenv (standard environment variables)
allocating as files for passing input 738
using the BPXBATCH command 738
using the BPXBATCH program 803
stderr (standard error)
allocating as files for passing input
using the BPXBATCH command 738
using the BPXBATCH program 803
stdin (standard input)
allocating as files for passing input
using the BPXBATCH command 738
using the BPXBATCH program 803
closing 503
copying
data read from 635
standard output to each 623
copying one line of 309
explanation of 3
reading 98
reading lines from 466
stdout (standard output)
allocating as files for passing input
using the BPXBATCH command 738
using the BPXBATCH program 803
closing 503
copying standard input to each 623
dumping file to 420
explanation of 3
sending paginated files to 450
writing
arguments to 225
configuration values to 267
writing arguments to 456
step subcommand for dbx command 196
stepi subcommand for dbx command 196
STEPLIB environment variable
description of 509
sticky bit 110
stop
dbx debug session 185
pending UUCP transfers 679
program at a specific location 198
program execution 197
removing from program 167
removing from source lines 164
shell 240
stop subcommand for dbx command 197
displaying 195
stopi subcommand for dbx command 198
string
displaying in a binary file 532
finding, in text files 797
searching for 274
strings shell command 532
strip shell command 533
stty shell command 534
stub commands
cancel 96
cu 152
explanation of 819
lpstat 326
su shell command 540
SUBCOMMAND function key for OMVS command 771
subcommand mode
setting up
function key to enter 771
setting up function key to return from 771
submit
batch jobs
using the BPXBATCH command 738
using the BPXBATCH program 803
C/MVS executable files 803
using the BPXBATCH program 803
MVS batch jobs that run shell commands 803
using the BPXBATCH program 803

submit *(continued)*
OS/390 batch jobs that run shell commands
using the BPXBATCH command 738
subscript-in-array condition 28
substitute
commands 500
directories 496
suffix 760
sum shell command 543
summarize
use of file space 224
suppress command numbers 251
suspend program execution 521
swap bytes 141
switch
locales 9
to the next higher-numbered session 770, 775
to the previous (lower-numbered) session 771, 775
symbol
changing interpretation of 163
displaying qualifications of 206
symbol table
displaying the 407
symbol table used in awk 29
symbolic link
ln 314
symbolic link (symlink)
displaying attributes and contents of 739
symbolic links
SYMLINK
linkname 812
SYMPATH 812
symlink
displaying attributes and contents of 739, 783
symlinks tcsh shell variable
description of 596
SYMTAB symbol table 29
synopsis of dbx commands
displaying 174
syntax
explanation of 1
syntax, lexical
reading description of 306
SYSEXEC environment variable
used by tso 641
sysplex
moving file systems in a sysplex 746
unmounting a file system 660
SYSPROC environment variable
used by tso 641
SYSROOT 787
dummy file system 747
system
calling up 152
connection to 152
system control registers
displaying values of 187
system files 624
System Object Model (SOM) 70
sysvar shell command 545

T

tab
compressing from spaces 656
expanding to spaces 241
tab stop
setting 545
tabs shell command 545
tag files
creating 150
tags file format 827
tail shell command 548
talk
to another user 549
talk shell command 549
tape archive 552
tar archive files
manipulating 552
tar file format 828
tar shell command 552
target 357
TCPIP.ETC.SERVICES file
used by lm 311
used by ocsconfig 418
tcsh
command execution 576
command syntax 567
signal handling 583
tcsh environment variable
COLUMNS
description of 598
DISPLAY
description of 598
EDITOR
description of 598
GROUP
description of 598
HOME
description of 598
HOST
description of 598
HOSTTYPE
description of 598
HPATH
description of 598
LANG
description of 598
LC_CTYPE
description of 598
LINES
description of 598

tcsh environment variable *(continued)*
MACHTYPE
description of 598
NOREBIND
description of 598
OSTYPE
description of 598
PATH
description of 598
PWD
description of 598
REMOTEHOST
description of 598
SHLVL
description of 598
TERM
description of 598
USER
description of 598
VENDOR
description of 598
VISUAL
description of 598
tcsh files 599
tcsh shell
alias shell command 11
at shell command 19
automatic, periodic, and timed events 582
bg shell command 61
break shell command 63
cd shell command 100
colon (:) shell command 121
echo shell command 225
eval shell command 236
exec shell command 238
exit shell command 240
features 578
fg shell command 253
history shell command 279
jobs shell command 298
kill shell command 301
ls–F shell command 613
Native Language System Report 582
newgrp shell command 403
nice shell command 404
nohup shell command 410
printenv shell command 455
problems and limitations 599
set shell command 482
shift shell command 519
status reporting 581
stop shell command 531
substitutions 567
suspend shell command 545
time shell command 629
umask shell command 652
tcsh shell *(continued)*
unalias shell command 653
unset shell command 663
wait shell command 720
tcsh shell command 557
alloc 601
bindkey 601
builtins 603
tcsh shell variable
ampm
description of 587
argv
description of 587
autocorrect
description of 587
autoexpand
description of 587
autolist
description of 587
autologout
description of 587
backslash
description of 587
cdpath
description of 587
command
description of 587
complete
description of 587
correct
description of 587
cwd
description of 587
dextract
description of 587
dirsfile
description of 587
dirstack
description of 588
dunique
description of 588
echo
description of 588
echo_style
description of 588
edit
description of 588
fignore
description of 589
filec
description of 589
gid
description of 589
group
description of 589
histchars
description of 589

tcsh shell variable *(continued)*
histdup
description of 589
histfile
description of 589
histlit
description of 589
history
description of 589
home
description of 589
ignoreeof
description of 589
implicitcd
description of 589
inputmode
description of 590
listflags
description of 590
listjobs
description of 590
listlinks
description of 590
listmax
description of 590
listmaxrows
description of 590
loginsh
description of 590
logout
description of 590
mail
description of 590
matchbeep
description of 590
nobeep
description of 590
noclobber
description of 590
noglob
description of 591
nokanji
description of 591
nonomatch
description of 591
nostat
description of 591
notify
description of 591
owd
description of 591
path
description of 591
printexitvalue
description of 591
prompt
description of 594

tcsh shell variable *(continued)*
prompt2
description of 591
prompt3
description of 591
promptchars
description of 591
pushdsilent
description of 591
pushdtohome
description of 591
recexact
description of 591
recognize_only_executables
description of 591
rmstar
description of 592
rprompt
description of 592
savedirs
description of 592
savehist
description of 592
sched
description of 592
shell
description of 592
shlvl
description of 592
status
description of 592
symlinks
description of 596
tcsh
description of 589, 592
term
description of 592
time
description of 597
tperiod
description of 592
tty
description of 592
uid
description of 592
user
description of 592
verbose
description of 592
version
description of 593
visiblebell
description of 593
watch
description of 594
who
description of 594

tcsh shell variable *(continued)*
wordchars
description of 594
tcsh tcsh shell variable
description of 592
tee shell command 623
template for commands 727
temporary files 234
/tmp/sh*
description of 510
term environment variable
used by mailx 354
used by more 390
used by TABS 547
used by talk 550
used by touch 634
used by vi 689, 715, 716
TERM tcsh environment variable
description of 598
term tcsh shell variable
description of 592
terminal
changing characteristics of 633
sending messages to a 726
setting, resetting, or querying ASCII/EBCDIC code pages 105
terminal entry
printing 97
terminal name
displaying 645
terminal options
displaying 534
setting 534
terminal tab stop
setting 545
terminfo database
printing terminal entries in the 97
terminfo database entries
comparing 286
compiling 627
printing 286
TERMINFO environment variable 628
used by tabs 547
used by talk 550, 634
used by vi 716
terminfo.src file
used by tic 628
test condition 624
test shell command 624
text editor
ex 237
using the 227
vi 688
text file
comparing two 214
concatenating 98
text file *(continued)*
counting items in 721
displaying 98
finding information in 25
finding strings in 797
retrieving information from 25
showing differences between two 214
splitting 148
text insertion commands (for vi) 691
text insertion commands (vi command) 697
TFS file systems
unmounting 787
then shell subcommand 492
then statement
using null shell statement 121
thread
displaying information about 199
thread subcommand for dbx command 199
tic shell command 627
time
displaying 154
time program 629
time sharing option extensions 737
See also TSO/E
time shell command 629
time tcsh shell variable
description of 597
time zone
setting, with the TZ environment variable 837
times shell command 630
TMOUT environment variable
description of 509
TMP environment variable
used by exrecover 246
TMP_VI environment variable
used by vi 717
TMPDIR environment variable
description of 509
used by ar 16
used by ed 232
used by exrecover 246
used by man 379
used by pg 449
used by sort 525
used by vi 716
token
description of 493
TOP function key for OMVS command 772
TOP subcommand of OMVS command 776
toplines environment variable
used by mailx 344, 354
topological sort 644
touch shell command 631
tperiod tcsh shell variable
description of 592

tput shell command 633
tr shell command 635
trace
removing from program 167
trace subcommand for dbx command 200
displaying 195
tracei subcommand for dbx command 201
tracing
turning on 201
tracing information
printing 200
tracked alias 494
creating a 276
transfers, UUCP
displaying status of pending 679
translate characters 635
trap
abnormal conditions and interrupts 637
signals
starting 163
stopping 175
trap shell command 637
true shell command 639
TSO function key for OMVS command 772
tso shell command 640
TSO subcommand of OMVS command 776
TSO/3270 passthrough mode
returning to shell mode 767
TSO/E (Time Sharing Option Extension)
environment variables
RESOLVER_CONFIG 841
X_ADDR 841
X_SITE 841
X_SLATE 841
TSO/E (Time Sharing Option Extensions)
command mode
invoking the 776
commands
ISHELL 739
MKDIR 740
MKNOD 741
MOUNT 743
OBROWSE 747
OCOPY 748
OEDIT 753
OGET 754
OGETX 758
OHELP 762
OMVS 764
OPUT 776
OPUTX 780
OSHELL 783
OSTEPLIB 784
UNMOUNT 785
entering commands from 737
giving users access to OS/390 UNIX System Services and shell 739, 783
TSO/E (Time Sharing Option Extensions) *(continued)*
invoking BPXBATCH from the 783
returning to the 240
running
C/MVS executable files 738
commands from the shell using the 640
shell commands 738
shell scripts 738
setting up function key to enter mode 772
TSOALLOC environment variable
used by tso 641
TSOPREFIX environment variable
used by tso 642
TSOPROFILE environment variable
used by tso 642
tsort shell command 644
tty shell command 645
tty tcsh shell variable
description of 592
turn off
automatic scrolling 775
type shell command 646
typeset shell command 647
TZ environment variable
description of 510
setting timezones with the 837
used by at 21
used by cron 143
used by crontab 147
used by date 156
used by ls 331
used by mail 334
used by pr 452
used by touch 632
used by uulog 675
used by uustat 681

U

uid tcsh shell variable
description of 592
ulimit shell command 650
umask shell command 651
unalias shell command 652
unalias subcommand for dbx command 202
uname shell command 654
uncompress
data 735
Huffman-coded files 445
uncompress shell command 655
unconfigure
OCS objects 412
underscore (_) variable
description of 507
undo change 231

unexpand shell command 656
unhide
data entered on the shell command line 775
OMVS command input area 770, 775
uniq shell command 657
unique lines 657
Universal Time Coordinated (UTC)
used by the TZ environment variable 837
UNIX C shell 557
unlink shell command 659
unmount
a file system 660
hierarchical file systems 787
TFS file systems 787
unmount hierarchical file systems (HFS) 739
unmount shell command 660
UNMOUNT TSO/E command 785
unpack shell command 661
unprintable characters
displaying 99
unset
attributes of variables and functions 663
command options 482
positional parameters 482
values of variables and functions 663
unset shell command 663
unset subcommand for dbx command 203
until loop
exiting from, in a shell script 63
until shell subcommand 493
up subcommand for dbx command 203
update
data 771
uppercase
converting to lowercase 209
uppercase letters 1
Usage Notes section
explanation of 6
use subcommand for dbx command 204
user
sending messages to a 726
setting up 739
talking to another user 549
user ID
changing to superuser 540
user ID (UID)
returning 324
setting to owner 113
user identity
finding 282
returning 282
user name
displaying your 725
USER tcsh environment variable
description of 598
user tcsh shell variable
description of 592
USERID environment variable
description of 839
users
displaying information about current 723
/usr/spool/uucp spool directory 831
/usr/spool/uucp/south directory 831
usrspooluucpxq 684
UTC (Universal Time Coordinated)
used by the TZ environment variable 837
utility
invoking, while ignoring the SIGHUP signal 410
parsing options 271
utmpx file format 830
uucc shell command 664
uucico daemon 665
UUCP
configuration file
/usr/lib/uucp/config 666
reading contents of 664
copying files between systems 667
debug file
/usr/spool/uucp/LOGFILE 666
displaying
list of systems 676
status of transfers 679
events
displaying 674
lock file
/usr/spool/locks 666
searching public directories 677
spool directory (/usr/spool/uucp) 831
status file
/usr/spool/uucp/.Status 666
transfers
displaying status of 679
starting or stopping 679
validating requests by the uucpd program 671
working files
command 831
data 831
execute 831
UUCP file transfer daemon 665
uucp shell command 667
processing file transfer requests 665
uucpd daemon
handling of uucp requests 671
uudecode shell command 672
uuencode shell command 673
uulog shell command 674
uuname shell command 676
uupick shell command 677
uustat shell command 679
uuto shell command 682

uux shell command 683
processing file transfer requests 665
uuxqt daemon 687
uuxqt shell command
/usr/spool/uucp/.Xqtdir directory 684

V

value
defining, for dbx variables 190
displaying, for registers 187
variable
assigning
attributes and variables to 647
values to 162
attributes 647
bc command, for the 47
built-in, for the bc shell command 47
condition
displaying list of 165
deleting 203
description of 497
displaying
currently exported variables 242
list of 647
names of variables in procedures 170
values of variables in procedures 170
environment
displaying 235
listing their attributes 647
parameters used by shell 496, 583
printing tracing information 200
setting export attributes 242
unsetting values and attributes of 663
used in awk 27
variable records
converting to fixed records 208
variable to fixed-record conversion 208
VENDOR tcsh environment variable
description of 598
verbose tcsh shell variable
description of 592
version tcsh shell variable
description of 593
vi command
editor initialization 715
entering ex command mode 700
file recovery daemon for 245
fullword
definition of 693
regular expressions 701
scrolling commands 691
set option variables 710
word
definition of 693
vi file recovery daemon 245
vi mode
absolute movement commands 690
list of 692
command summary 690
context-dependent movement commands 691
current position pointer 690
display conventions 690
manipulation commands 691
object manipulator commands 695
scrolling commands 690
starting session in 689
text insertion commands 691, 697
vi shell command
command mode 689
creating tag files for the 150
insert mode 689
starting sessions in vi mode 689
vi text editor
using the line-editor mode 237
visiblebell tcsh shell variable
description of 593
VISUAL environment variable
description of 510
used by mailx 345, 347, 354
used by shedit 513
VISUAL tcsh environment variable
description of 598

W

wait
for child process to end 720
for jobs to end 720
wait shell command 720
wall shell command 721
watch tcsh shell variable
description of 594
wc shell command 721
whatis subcommand for dbx command 205
whence shell command 722
where subcommand for dbx command 205
whereis subcommand for the dbx command 206
which subcommand for the dbx command 207
while loop
exiting from, in a shell script 63
while shell subcommand 488, 493
who shell command 723
who tcsh shell variable
description of 594
whoami shell command 725
wildcard characters 503
within-rule circular dependency 362
word
counting 721
definition of, for vi 693

word *(continued)*
description of 493
wordchars tcsh shell variable
description of 594
words
misspelled
looking for 527
working directory
changing
to directory 100
to previous working directory 100
displaying pathname of the 465
setting to value of the HOME environment variable 100
working files
format of UUCP 831
WRAPDEBUG option of OMVS command 773
write
archive files 433, 552
arguments to standard output 225
checksum for each input file
with the cksum command 117
with the sum command 543
configuration values to standard output 267
cpio archives 140
data 207
debugging information 767
formatted output 456
number of bytes in each input file
with the cksum command 117
with the sum command 543
printouts created by programs 17
to other users 726
write mode 434
write shell command 726

X

X_ADDR environment variable
description of 841
X_SITE environment variable
description of 841
used by ocsconfig 417
X_SLATE environment variable
description of 841
xargs shell command 727
usrspooluucpxq/usr/spool/uucp/ 684
xtrace 647

Y

y.output file
used by yacc 732
y.tab.c file
used by yacck 732
y.tab.h file
used by yacc 732
yacc compiler
using the 731
yacc shell command 731
YYDEBUG option 732

Z

zcat shell command 735

Communicating Your Comments to IBM

OS/390
UNIX System Services
Command Reference

Publication No. SC28-1892-09

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

If you are mailing a readers' comment form (RCF) from a country other than the United States, you can give the RCF to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF at the back of this book.
- If you prefer to send comments by FAX, use this number:

  1-(914)-435-9405
- If you prefer to send comments electronically, use this network ID:

  mhvrcfs@us.ibm.com

Make sure to include the following in your note:

- Title and publication number of this book
- Page number or topic to which your comment applies.

Readers' Comments — We'd Like to Hear from You

OS/390
UNIX System Services
Command Reference

Publication No. SC28-1892-09

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	□	□	□	□	□

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	□	□	□	□	□
Complete	□	□	□	□	□
Easy to find	□	□	□	□	□
Easy to understand	□	□	□	□	□
Well organized	□	□	□	□	□
Applicable to your tasks	□	□	□	□	□

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? □ Yes □ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name _____ Address _____

Company or Organization _____

Phone No. _____

Readers' Comments — We'd Like to Hear from You
SC28-1892-09

Cut or Fold
Along Line

IBM®

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Department 55JA, Mail Station P384
2455 South Road
Poughkeepsie, NY 12601-5400

Fold and Tape | **Please do not staple** | Fold and Tape

Cut or Fold
Along Line

Program Number: 5647-A01

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC28-1892-09