IBM RT Advanced Interactive Executive Operating System Version 2.2

AIX Operating System Commands Reference

Volume 1

Programming Family

IBM

SC23-2011-1

IBM RT Advanced Interactive Executive Operating System Version 2.2

AIX Operating System Commands Reference

Volume 1

Programming Family

IBM

SC23-2011-1

Second Edition (September 1988)

Trademarks

About This Book

This book contains reference information on Advanced Interactive Executive (AIX) Operating System commands. It describes the commands you can use and summarizes who can run them, how to run them, what they do, how they read input, how they write output, and how to modify their actions.

Who Should Use This Book

To use this book, you should be familiar with AIX or UNIX System V commands. If you are not already familiar with AIX or UNIX System V, see *Using the AIX Operating System.* If you are familiar with the commands but need to review how to use the shell and write shell procedures, see "**sh**" on page 913.

How To Use This Book

Most of the AIX commands described in this book are in alphabetical order by command name. Some related commands are combined in one description listed with a main or key command. The related commands have an entry with the main command in the table of contents and are listed individually in alphabetical order in the index. If you are having difficulty locating a particular command, check the "Contents" or "Index" sections of this publication.

Command Information

The "Commands" section begins on page 11. A discussion of a command may include the following information:

Purpose	A single-sentence description of the major function of each command
Syntax	A ***syntax diagram*** that shows command line options (For a discussion of how to use this syntax diagram, see "Syntax Diagrams" on page 5.)
Description	A discussion of the command that provides more details about its function and use

Flags	A list of command line flags and associated parameters with an explanation of how the flags modify the action of the command
Subcommands	A list of subcommands (for interactive commands) that explains their use
Examples	Specific examples of how you can use the command
Files	A list of files used by the command
Related Information	A list of related commands in this book and related discussions in other books.

For details on other conventions used in this book, see "How to Use the Commands" on page 3.

A Task Index

"Task Index" on page TASK-1 can help you locate the commands you need to perform specific tasks. It contains lists of commands grouped by task. Next to each command is a description of what it does. To find a command that performs a specific task, locate the task in the table of contexts at the beginning of the task index, go to the indicated page and review the list of commands associated with that task, then select the desired command. For more information about the command, refer to the discussion of the command in the "Commands" section.

Other Reference Aids

A cross-reference listing of commands and program packages appears in Appendix B, "Program Cross-Reference" on page 1269. Appendix C, "Syntax Diagram Guide" on page 1277 contains a detailed description of how to read syntax diagrams. The standard system devices are described in Appendix A, "AIX Device Table" on page 1267. A "Glossary" of terms appears after the Appendixes, followed by an "Index."

In addition, a Reader's Comment Form and Book Evaluation Form are provided at the back of the second volume of this publication. Use the Reader's Comment Form at any time to give IBM information that may improve the book. After you have become familiar with the book, use the Book Evaluation Form to give IBM specific feedback about the book.

Japanese Language Support

Appendix D, "Japanese Language Support" on page 1287 contains a list of commands that have not been modified to support Japanese characters.

Special Key Sequences

You can use the AIX Operating System from any of several different display stations, each of which has a different keyboard. In some cases, you must press different keys to perform the same function from different keyboards. Throughout this publication both the function name (for example, INTERRUPT) and the necessary key sequence on the IBM RT system are identified. If you are not using an IBM RT Keyboard, look at your keyboard reference chart to find out which keys on your keyboard produce the same function.

Prerequisite Information

- *IBM RT Managing the AIX Operating System* provides instructions for performing such system management tasks as adding and deleting user IDs, creating and mounting file systems, repairing file system damage, and managing data communications facilities.
- *IBM RT Using the AIX Operating System* describes using the AIX Operating System commands, working with file systems, developing shell procedures, and using data communications facilities.

Related Information

- *IBM RT AIX Operating System Programming Tools and Interfaces* describes the programming environment of the AIX Operating System and includes information about using the operating system tools to develop, compile, and debug programs. In addition, this book describes the operating system services and how to take advantage of them in a program. This book also includes a diskette that includes programming examples, written in C language, to illustrate using system calls and subroutines in short, working programs. (Available optionally)
- *IBM RT AIX Operating System Technical Reference* is a four-volume set.

  System Calls and Subroutines, describes the system calls and subroutines that a C programmer uses to write programs for the AIX Operating System.

  Files and Extensions, contains information about the extensions to the kernel and base operating system, including file formats, special files, and GSL subroutines.

  VRM Programming Support, describes the VRM programming environment, including the internal VRM routines, VRM floating-point support, use of the VRM debugger, and the supervisor call instructions that form the Virtual Machine Interface.

  VRM Device Support, describes device IPL and configuration, minidisk management, the virtual terminal and block I/O subsystems, as well as the interfaces to VRM device driver and data link control components. This volume also describes the programming conventions for developing your own VRM code and installing it on the system. (Available optionally)

- *IBM RT Using DOS Services* provides step-by-step information for using AIX Operating System **shell**. (Available optionally; packaged with *IBM RT DOS Services Reference*)
- *IBM RT DOS Services Reference* provides reference information about the AIX Operating System **shell**. This book also includes information on sharing DOS files with Personal Computer AT Coprocessor Services, and on the differences between PC DOS and **shell**. (Available optionally; packaged with *IBM RT Using DOS Services*)
- *IBM RT C Language Guide and Reference* provides guide information for writing, compiling, and running C language programs and includes reference information about C language data structures, operators, expressions, and statements. (Available optionally)
- *IBM RT Messages Reference* lists messages displayed by the IBM RT and explains how to respond to the messages.
- *IBM RT AIX Operating System Text Formatting Guide* describes the functions and capabilities of NROFF and TROFF to perform text processing tasks. (Available optionally)
- *IBM RT Bibliography and Master Index* provides brief descriptive overviews of the books and tutorial program that support the IBM RT hardware and the AIX Operating System. In addition, this book contains an index to the RT and AIX Operating System library.

See *IBM RT Bibliography and Master Index* for order numbers of IBM RT publications and diskettes.

Ordering Additional Copies of This Book

To order additional copies of this publication (without program diskettes), use either of the following sources:

- To order from your IBM representative, use Order Number SBOF-1814.
- To order from your IBM dealer, use Part Number 27F4354.

A binder is included with the order. For information on ordering the binder and manual separately, contact your IBM representative or your IBM dealer.

Contents

VOLUME 1 1

How to Use the Commands 3
Command Input and Output 4
File Name Substitution 4
Syntax Diagrams 5
Command, Flag, and Parameter Notation 8

Commands 11
acct/* 13
  chargefee 14
  ckpacct 14
  dodisk 14
  lastlogin 15
  monacct 15
  nulladm 15
  prctmp 15
  prdaily 15
  prtacct 16
  remove 16
  shutacct 16
  startup 16
  turnacct 16
acctcms 18
acctcom 20
acctcon 24
  acctcon1 24
  acctcon2 25
acctdisk, acctdusg 26
acctmerg 28
acctprc 30
  acctprc1 30
  acctprc2 31
  accton 31
actman 32
adb 33
admin 41
ali 48
anno 50

ap 53
ar 55
arithmetic 59
as 61
at, batch 63
audit 67
auditapp 69
auditbin 71
auditpr 73
auditselect 76
auditstream 78
auditwrite 80
awk 81
back 87
backup 88
banner 94
basename, dirname 95
bc 97
bdiff 102
bellmail 104
bffcreate 108
bfs 110
biod 114
biodd_cfg 115
bj 117
bs 118
burst 129
cal 132
calendar 134
cat 137
cb 139
cc 140
cd 150
cdc 152
cflow 154
chgrp 156
chkcomp 158
chmod 160
chngstate 164
    installc 166
    updatec 167
chown 169
chparm 171
chroot 172
chtcb 174
clri 175

cmp . . . 177
col . . . 179
comb . . . 181
comm . . . 183
comp . . . 185
confer . . . 189
config . . . 194
conflict . . . 196
connect . . . 198
cp . . . 202
cpio . . . 205
cpp . . . 210
craps . . . 214
crash . . . 215
cron . . . 220
crontab . . . 222
csh . . . 225
csplit . . . 252
ct . . . 254
ctab . . . 257
ctags . . . 261
cu . . . 263
cut . . . 269
cvid . . . 272
Cvt . . . 274
cw, **checkcw** . . . 275
cxref . . . 279
date . . . 281
dbx . . . 284
dc . . . 295
dcopy . . . 299
dd . . . 301
defkey . . . 306
del . . . 308
delta . . . 310
deroff . . . 313
devices . . . 315
devnm . . . 316
df . . . 318
diff . . . 320
diff3 . . . 323
diffmk . . . 326
dircmp . . . 328
diskusg . . . 330
display . . . 332
dist . . . 336

domainname 340
dos 341
dosdel 345
dosdir 346
dosread 348
doswrite 350
dp 352
dsipc 354
dsldxprof 355
dspcat 357
dspmsg 359
dsstate 361
dsxlate 363
du 364
dump 366
dumpfmt 368
echo 369
ed 371
edconfig 385
edit 387
env 393
eqn, neqn, checkeq 395
errdead 397
errdemon 398
errpt, errpd 400
errstop 404
errupdate 405
ex 407
expr 412
factor 416
ff 417
file 420
find 422
fish 427
fmt 428
folder 429
folders 433
format 436
fortune 437
forw 438
fptype 444
fsck, dfsck 445
fsdb 450
fuser 455
fwtmp 457
    acctwtmp 458

- **wtmpfix** . . . 458
gdev . . . 460
- **hpd** . . . 460
- **erase** . . . 461
- **hardcopy** . . . 461
- **tekset** . . . 461
- **td** . . . 461

ged . . . 463
gencat . . . 470
gend . . . 475
get . . . 477
getopt . . . 485
gettext . . . 488
getty . . . 490
graph . . . 494
graphics . . . 497
greek . . . 499
grep . . . 501
groups . . . 506
gutil . . . 508
- **bel** . . . 509
- **cvrtopt** . . . 509
- **gd** . . . 510
- **gtop** . . . 510
- **pd** . . . 510
- **ptog** . . . 510
- **quit** . . . 510
- **remcom** . . . 510
- **whatis** . . . 511
- **yoo** . . . 511

hangman . . . 512
help . . . 513
hp . . . 514
hyphen . . . 516
id . . . 517
inc . . . 518
init . . . 521
install . . . 524
install-mh . . . 527
installp . . . 529
- **inusave** . . . 531
- **inurecv** . . . 532
- **inurest** . . . 533
- **ckprereq** . . . 533
- **mvmd** . . . 534

ipcrm . . . 537

ipcs 539
ipctable 544
istat 545
join 547
keyboard 551
kill 552
killall 555
ld 557
lex 562
li 567
line 574
link, unlink 575
lint 577
ln 581
locator 583
login 584
loginx 587
logname 589
logout 590
lorder 591
lp 593
ls 595

VOLUME 2 . . . . **601**

m4 603
mail, Mail 608
mailstats 623
make 625
makedbm 632
makekey 634
man 635
mark 637
mdrc 640
mesg 642
mhl 643
mhmail 646
mhpath 648
minidisks 650
mkcatdefs 651
mkdir 657
mkfs 658
mknod 661
mm, checkmm 663
mmt, checkmm 666
moo 668

mount 669
mountd 674
msgchk 675
msh 677
mv 679
mvdir 682
ncheck 683
ndtable 685
newform 686
newgrp 689
news 691
next 694
nfsd 696
nfsstat 697
nice 699
nl 701
nm 705
nohup 707
nroff, troff 709
number 721
od 723
on 726
open 728
pack 730
  pcat 731
  unpack 731
packf 733
passwd 735
paste 736
pcnfs 739
pdisable, phold 741
pg 744
pick 748
piobe 753
portmap 757
post 758
pr 761
prev 765
print 767
prof 773
profiler 775
  prfld 776
  prfstat 776
  prfdc, prfsnap 776
  prfpr 776
prompter 778

proto 780
prs 781
ps 786
pstart, penable, pshare, pdelay 791
ptx 794
puttext 796
pwck 798
pwd 800
pwtable 801
qdaemon 802
quiz 803
rc 806
rcvdist 808
rcvpack 810
rcvstore 812
rcvtty 815
refile 817
regcmp 820
repl 821
restore 826
rexd 832
rm 833
rmail 836
rmdel 837
rmdir 838
rmf 839
rmm 841
rpcgen 843
rpcinfo 845
rstatd 847
runacct 848
runcat 852
rup 854
rusers 856
rusersd 858
rwall 859
rwalld 861
sact 862
sadc 863
    sa1 864
    sa2 864
sag 865
sar 867
scan 871
sccsdiff 874
sdb 875

sdiff 883
secure 885
sed 887
send 893
sendmail 897
setdma 910
setmnt 911
sh 913
shell 938
shlib 939
show 942
showmount 945
shutdown 946
size 949
skulker 951
sleep 952
slocal 954
sno 956
sort 958
sortm 965
sound 967
spell 969
spline 972
split 974
splp 975
spost 978
spray 981
sprayd 983
stat 984
strip 1017
stty 1018
su 1026
sum 1029
sync 1030
sysck 1031
syslogd 1037
tab, untab 1040
tabs 1041
tail 1044
tapechk 1047
tar 1048
tbl 1053
tc 1056
tctl 1058
tee 1060
termdef 1062

test 1064
tic 1067
time 1068
timex 1069
tlog 1071
tlogger 1072
toc 1074
  dtoc 1074
  ttoc 1075
  vtoc 1075
touch 1077
tplot 1079
tput 1081
tr 1083
trace 1086
trcrpt 1091
trcstop 1093
trcupdate 1094
trdiag 1097
true 1099
tsh 1100
tsort 1102
ttt 1104
tty 1105
turnon 1107
tvi 1108
ugtable 1109
umask 1110
umount, unmount 1112
uname 1114
unget 1116
uniq 1118
units 1119
updatep 1122
  inudocm 1125
  inuupdt 1127
users, adduser 1129
uucpadm 1133
uucheck 1137
uucico 1139
uucleanup 1141
uucp 1144
  Path Names Used with uucp 1145
  Source and Destination File Names 1145
  Permissions 1146
uulog 1149

uuname 1151
uupick 1153
File-Handling Options 1154
uusched 1156
uustat 1158
uuto 1162
uutry, Uutry, uukick 1164
uux 1166
uuxqt 1172
val 1175
varyoff 1177
varyon 1180
vc 1182
verify 1186
vi, vedit, view 1187
vmh 1203
vrm2rtfont 1205
vrmconfig 1206
wall 1208
watch 1209
wc 1211
what 1213
whatnow 1215
who 1219
whom 1222
write 1225
writesrv 1230
wump 1231
xargs 1232
xdbx 1236
yacc 1237
ypbind 1239
ypcat 1241
ypinit 1243
ypmatch 1245
yppasswd 1247
yppasswdd 1249
yppoll 1251
yppush 1252
ypset 1254
ypserv 1256
ypwhich 1258
ypxfr 1260
300 1262
4014 1264
450 1265

Appendix A. AIX Device Table 1267

Appendix B. Program Cross-Reference 1269

Appendix C. Syntax Diagram Guide 1277

Appendix D. Japanese Language Support 1287

Glossary 1291

Task Index TASK-1

Index INDEX-1

Figures

1. SCCS Header Flags 44
2. SID Determination 481
3. Mailbox Commands 611
4. Mail Editor Commands 617
5. Binary Options 619
6. Valued Options 620
7. Delta Table Keywords 782
8. Header Flag Keywords 783
9. Other Keywords 784
10. Configuration Options 905
11. **tbl** Column and Item Specifiers 1054
12. Configuration File Parameters 1129
13. AIX Standard Devices (Special Files) 1268

VOLUME 1

How to Use the Commands

This section contains a description of:

- Command input and output
- File name substitution by the shell
- Syntax diagrams
- Command, flag, and parameter notation.

To help you determine which command you want to use, see "Task Index" on page TASK-1. To help you determine in which program a command is located, see Appendix B, "Program Cross-Reference" on page 1269.

Command Input and Output

Many commands take their input from ***standard input*** and write their output to ***standard output***. By default, standard input comes from the keyboard, and standard output goes to the display. It is important to remember this information as you read the command descriptions since they describe the default action. In this context, the verb ***display*** means "write to the standard output." Any command that reads standard input and writes to standard output can have its input or output redirected to a file and can be used in a ***pipeline***, where the standard output of a previous command is directed to the standard input of the next command. For more information on pipelines, see "**sh**" on page 913.

There are a few commands that must have a file name supplied or that must read standard input. You can see what a particular command can read by looking at the syntax diagram at the beginning of the description of the command. For instructions on interpreting syntax diagrams, see "Syntax Diagrams" on page 5.

File Name Substitution

When *file* is supplied as an argument to either a command or a flag, you can automatically produce a list of file name arguments by specifying a pattern for the shell to match with file names in a directory. Most characters in such a pattern match themselves, but you can also use some special ***pattern-matching characters*** in your pattern. These special characters are:

*	Matches any string, including the null string.
?	Matches any one character.
[. . .]	Matches any one of the characters enclosed in square brackets.
[! . . .]	Matches any character ***other than*** one of the characters that follow the exclamation mark within square brackets.

Inside square brackets, a pair of characters separated by a - (minus) specifies a set of all characters that collate within the range of that pair, as defined by the variable **NLCTAB** or **NLFILE**, so that `[a-dy]` is equivalent to `[abcdy]` if only b and c collate between a and d.

Japanese Language Support Information

You can also use ***character classes*** inside square brackets by enclosing the character class name between a `[:` and a `:]` inside the square brackets. For example, `[[:alpha:]]` matches any alphanumeric character. The character classes recognized are:

[:lower:]	All lowercase letters.

[:upper:]	All uppercase letters.
[:alpha:]	All letters.
[:digit:]	Digits 0 - 9.
[:alnum:]	All letters and digits.
[:print:]	All printable characters.
[:punct:]	All punctuation characters.
[:space:]	Space, tab, form feed, or carriage return.
[:jalpha:]	SJIS Roman characters.
[:jdigit:]	SJIS Arabic numerals.
[:jpunct:]	SJIS punctuation characters.
[:jparen:]	SJIS parentheses characters.
[:jkanji:]	SJIS kanji characters.
[:jhira:]	SJIS hiragana characters.
[:jkata:]	SJIS and half-width katakana characters.

End of Japanese Language Support Information

Using pattern-matching characters in file names on the command line has some restrictions. If the first character of a file name is a . (dot), it can be matched only by a pattern that begins with a dot. For example, `*file` matches the file names `myfile` and `yourfile`, but not `.myfile` or `.yourfile`. Use the pattern `.*file` to match these file names.

If a pattern does not match any file names, the pattern itself is returned as the result of the match.

Note: File and directory names should not contain the characters *, ?, [, or] because this may create infinite loops during pattern matching attempts.

Syntax Diagrams

Before each command discussion in the "Commands" section is a syntax diagram. These diagrams are designed to provide information about how to enter the command on the command line. A syntax diagram can tell you:

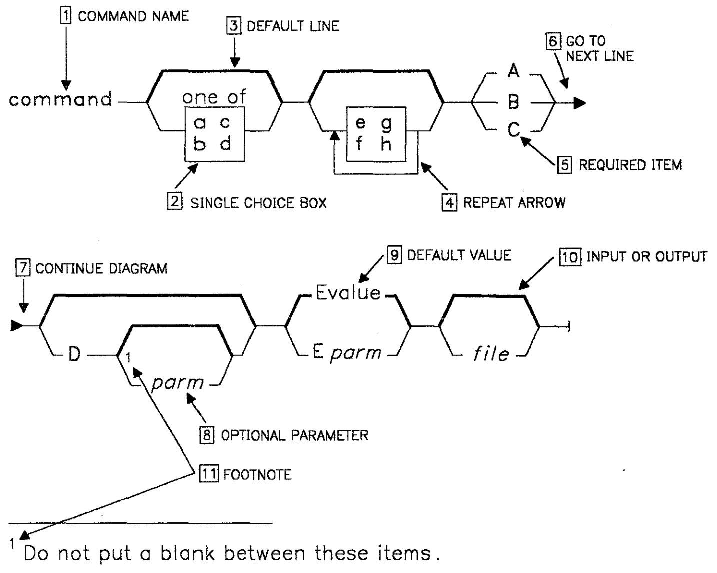
- Which flags can be entered on the command line
- Which flags must take parameters
- Which flags have optional parameters
- Default values of flags and parameters, if any
- Which flags can and cannot be entered together

- Where you must enter flags or parameters and where you have a choice
- Where you can repeat flag and parameter sequences.

This command reference uses the following conventions in the syntax diagrams:

- Diagram items that must be entered literally on the command line are in **bold**. These items include the command name, flags, and literal characters.
- Variable diagram items that must be replaced by a name are in *italics*. These items include parameters that follow flags and parameters that the command reads, such as *files* and *directories*.
- Default values that do not have to be entered are in the normal font on a **bold** path.

The following diagram is an example that illustrates the conventions used in the syntax diagrams. Each part of the diagram is labeled. An explanation of the labels follows the diagram.



1 Do not put a blank between these items.

OL805370

You interpret the diagram as follows:

1 COMMAND NAME	The first item in the diagram is the name of the command you want to invoke. It is in bold, so it must be entered exactly as it appears in the diagram. After the command name, the path branches into two paths. You can follow either path.
2 SINGLE CHOICE BOX	If you follow the lower path, you encounter a box with the words `one of` over it. You can choose only one item from this box.

3 DEFAULT LINE	If you follow the upper path, you bypass the single choice box, and enter nothing. The bold line around the box is a default line, which means that you do not have to enter anything from that part of the diagram. Exceptions are usually explained under "Description." One important exception, the blank default line around input and output files, is explained in item **10**.
4 REPEAT ARROW	When you follow a path that takes you to a box with an arrow around it, you must choose at least one item from the box. Then you can either follow the arrow back around and continue to choose items from it, or you can continue along the path. When following the arrow around just the box (rather than an arrow that includes several branches in the diagram), do not choose the same item more than once.
5 REQUIRED ITEM	Following the branch with the repeat arrow is a branch with three choices and no default line around them. This means that you must choose one of **A**, **B**, or **C**.
6 GO TO NEXT LINE	If a diagram is too long to fit on one line, this character tells you to go to the next line of the diagram to continue entering your command line. Remember, the diagram does not end until you reach the vertical mark.
7 CONTINUE DIAGRAM	This character shows you where to continue with the diagram after it breaks on the previous line.
8 OPTIONAL PARAMETER	If a flag can but does not have to take a parameter, the path branches after the flag. If you cannot enter a space between the flag and parameter, you are told in a footnote.
9 DEFAULT VALUE	Often, a command has default values or actions that it will follow if you do not enter a specific item. These default values are indicated in normal font in the default line if they are equivalent to something you could enter on the command line (for example, a flag with a value). If the default is not something you can enter on the command line, it is not indicated in the diagram. However, it is discussed under "Flags." **Note:** Default values are included in the diagram for your information. Do not enter them on the command line.
10 INPUT OR OUTPUT	A command that can read either input files or standard input has an empty default line above the file parameter. If the command can write its output to either an output file or to standard output, it is also shown with an empty default line above the output file parameter. If a command can read only from standard input, an input file is not shown in the diagram, and standard input is assumed. If a command writes only to standard output, an output file is not shown in the diagram, and standard output is assumed. When you must supply a file name for input or output, the file parameter is included in the diagram without an empty default line above it.

11 FOOTNOTE	If a command has special requirements or restrictions, a footnote calls attention to these differences.

Following are examples of valid ways this command can be entered based on this syntax diagram.

```
command name A
command name C
command name a B
command name d B
command name e A
command name e g f A
command name C D
command name C D8
command name A E7
command name B myfile
command name a e g B D3 E6 myfile
command name d f e h C D myfile
```

When the order of flags is important, it is indicated in the diagram, under "Flags," or in both places. Otherwise, the flags can be entered in any order. With this in mind, an additional example of how to enter this command is:

```
command name E9 a D g A h f myfile
```

For more detailed information on syntax diagrams, see Appendix C, "Syntax Diagram Guide" on page 1277.

Command, Flag, and Parameter Notation

The following type style conventions are used in command descriptions to distinguish different kinds of information:

bold	Commands, flags, and other items in bold are to be entered literally.
italics	Command parameters, flag parameters, and other items in italics are items for which you substitute an appropriate value in that position on the command line. For example, if you see *file*, you should type in the name of a file in that position.
[ ]	Items in brackets are optional. The only exception is brackets that are in bold. Brackets in bold are part of what should be entered literally.
. . .	Items followed by an ellipsis can be repeated. Thus, if you see *file* . . . , you can type several file names separated by blanks.

Using these conventions, the following string:

-D*name*[=*value*]

shows that, with the **-D** flag, the *name* parameter is required but assigning a *value* to *name* is optional. The following are valid ways to specify this flag and parameter combination:

```
-Daxis
-Daxis=10
```

The next string shows a parameter that can be replaced by several values:

-l *file* . . .

The following are valid ways to enter the **-l** flag:

```
-l memo letter
-l memo
-l letter
```

Commands

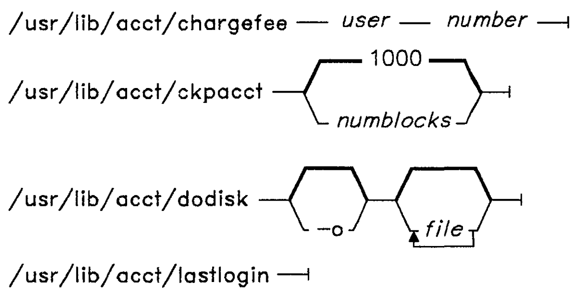
This section contains reference information for the AIX commands. This information may include the purpose of a command, one or more syntax diagrams to illustrate how a command can be entered on a command line, a description of how a command works, descriptions of command flags and subcommands, a list of related files, and cross references to related information.

acct/*

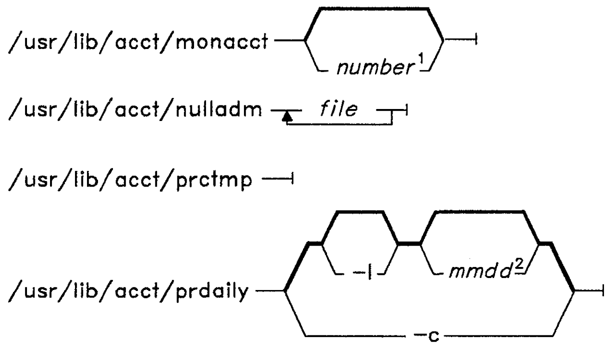
Purpose

Provides accounting shell procedures.

Syntax

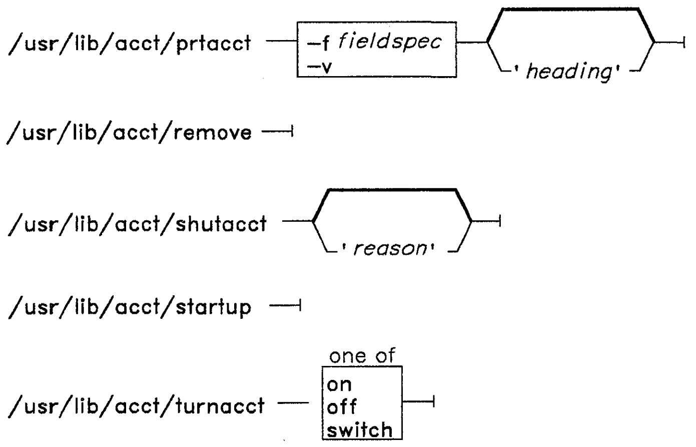


OL805236



[1] The default *number* is the current month.
[2] The default *mmdd* is the current day.

OL805237



OL805238

Description

Note: You should not share accounting files among nodes in a Distributed Services system. Each node should have its own copy of the various accounting files.

chargefee

The **chargefee** command charges the specified *number* of units to the specified *user*. *number* can have an integer or decimal value. It writes a record to **/usr/adm/fee**, to be merged with other accounting records by the **runacct** command.

ckpacct

The **ckpacct** command checks the size of **/usr/adm/pacct**. If the size exceeds the number specified in *numblocks*, **ckpacct** invokes **turnacct switch**. (The default value for *numblocks* is 1000.) If the number of free disk blocks in the **/usr** file system falls below 500, **ckpacct** automatically turns off the collection of process accounting records by invoking **turnacct off**. When 500 blocks are again available, accounting is activated again. This feature is sensitive to how frequently **ckpacct** is run (usually by **cron**).

dodisk

The **dodisk** command performs the disk-usage accounting functions. **cron** normally runs this command periodically. By default, it does disk accounting on the special files whose stanzas in **/etc/filesystems** contain the attribute **account=true**. If you specify the **-o** flag, it does a slower version of disk accounting by login directory.

The *file* parameter specifies the one or more file system names where disk accounting is to be done. If you specify any file names, disk accounting is done on only these file systems. If you do not specify **-o**, *file* names should be the special file names of mountable file systems. If you specify both **-o** and *file* names, the files should be mount points of mounted file systems.

lastlogin

The **lastlogin** command updates the file **/usr/adm/acct/sum/loginlog** to show the last date each user logged in. **runacct** normally calls this command.

monacct

The **monacct** command performs monthly (or periodic) accounting. **cron** should run this command once each month or accounting period. *number* indicates the month or period to process. The default *number* is the current month. This default is useful if **monacct** is run by **cron** on the first day of each month. The **monacct** command creates summary files in **/usr/adm/acct/fiscal** and restarts summary files in **/usr/adm/acct/sum**.

Daily reports are deleted (and thus inaccessible) each time **monacct** runs.

nulladm

The **nulladm** command creates *file*, assigns it permission code 664, and ensures that its owner and group are adm. (See "**chmod**" on page 160 for an explanation of file permissions.) Various accounting shell procedures call **nulladm.**

prctmp

The **prctmp** command displays the session record file created by the **acctcon1** command (normally **/usr/adm/acct/nite/ctmp**).

prdaily

The **prdaily** command formats a report of the day's accounting data. Use *mmdd* to specify a date other than the current day. The report resides in **/usr/adm/acct/sum/rprt***mmdd* where *mmdd* specifies the month and day of the report. **runacct** invokes this command to format a report of the previous day's accounting data.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Flags

-c Reports exceptional resource usage by command, and may be used on the current day's accounting data only.

-l Reports exceptional usage by login ID for the specified date.

prtacct

The **prtacct** command formats and displays any total accounting (**tacct**) file. You can specify a *heading* for the report by enclosing it in " " (double quotation marks).

Flags

-f*fieldspec*	Selects fields to be displayed, using the field selection mechanism of **acctmerg**.
-v	Produces verbose output in which more precise notation is used for floating-point numbers.

remove

The **remove** command deletes all **/usr/adm/acct/sum/wtmp***, **/usr/adm/acct/sum/pacct***, and **/usr/adm/acct/nite/lock*** files.

shutacct

The **shutacct** command turns process accounting off and adds a "reason" record to **/usr/adm/wtmp**. It is usually invoked during a system shutdown.

startup

The **startup** command turns on the accounting functions when the system is started up. It should be called by the **/etc/rc** command file.

turnacct

The **turnacct** command provides an interface to **accton** for turning process accounting **on** or **off**.

The **switch** flag turns accounting off, moves the current **/usr/adm/pacct** to the next free name in **/usr/adm/pacct***incr*, where *incr* is a number starting at 1 and increased by one for each additional **pacct** file. After moving the **pacct** file, **turnacct** turns accounting back on.

This command is usually called by **ckpacct**, which in turn is called by **cron**, keeping the **pacct** file down to a manageable size.

Files

/usr/adm/fee	Accumulator for fees charged to login names.
/usr/adm/pacct	Current file for process accounting.
/usr/adm/pacct*	Used if **pacct** gets large and during running of the daily accounting procedures.

/usr/adm/wtmp	Login/logout history file.
/usr/lib/acct/ptelus.awk	Shell procedure that calculates the limits for exceptional usage by login ID.
/usr/lib/acct/ptecms.awk	Shell procedure that calculates the limits of exceptional usage by command name.
/usr/adm/acct/nite	Working directory.
/usr/lib/acct	Holds all accounting commands.
/usr/adm/acct/sum	Summary directory.

Related Information

The following commands: "**acctcms**" on page 18, "**acctcom**" on page 20, "**acctcon**" on page 24, "**acctmerg**" on page 28, "**acctprc**" on page 30, "**chmod**" on page 160, "**cron**" on page 220, "**fwtmp**" on page 457, and "**runacct**" on page 848.

The **acct** system call and the **acct**, **utmp**, and **filesystems** files in *AIX Operating System Technical Reference.*

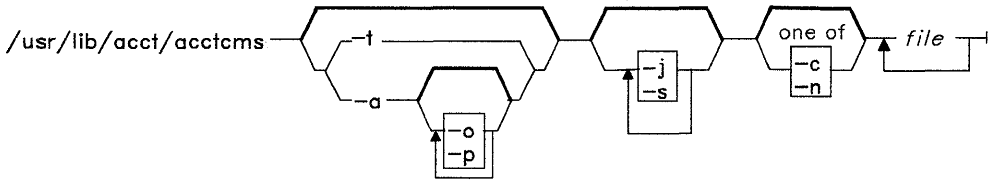
"Running System Accounting" in *Managing the AIX Operating System.*

acctcms

Purpose

Produces command usage summaries from accounting records.

Syntax



OL805421

Description

The **acctcms** command reads the specified *files*. It adds together all records for identically named processes, sorts them, and writes them to standard output in a binary format. Files are usually in the **acct** file format described in *AIX Operating System Technical Reference.*

When you use the **-o** and **-p** flags together, **acctcms** produces a report that combines prime- and nonprime-time. All the output summaries are of total usage except for number of times run, CPU minutes, and real minutes, which are split into prime and nonprime minutes.

A typical sequence for performing daily command accounting and for maintaining a running total is:

```
acctcms  file . . .   > today
cp  total  previoustotal
acctcms  -s  today  previoustotal  > total
acctcms  -a  -s  today
```

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Flags

-a	Displays output in ASCII summary format rather than binary summary format. Each output line contains the command name, the number of times the command was run, its total ***kcore-time***, its total ***CPU time***, its total ***real time***, its mean memory size (in K bytes), its mean CPU time per invocation of the command, and its ***CPU usage factor***. The listed times are all in minutes. **acctcms** normally sorts its output by total kcore-minutes. The unit kcore-minutes measures the amount of storage used (in K-bytes) multiplied by the amount of time it was in use.
-c	Sorts by total CPU time rather than total kcore-minutes.
-j	Combines under the heading `***other` all commands called only once.
-n	Sorts by the number of times the commands were called.
-o	Displays a command summary of nonprime-time commands only. You can use this flag with only the **-a** flag.
-p	Displays a command summary of prime-time commands only. You can use this flag with only the **-a** flag.
-s	Assumes that any named files that follow this flag are already in binary format.
-t	Processes all records as total accounting records. The default binary format splits each field into prime- and nonprime-time sections.

Related Information

The following commands: "**acct/***" on page 13, "**acctcom**" on page 20, "**acctcon**" on page 24, "**acctmerg**" on page 28, "**acctprc**" on page 30, "**fwtmp**" on page 457, and "**runacct**" on page 848.

The **acct** system call and the **acct** and **utmp** files in *AIX Operating System Technical Reference.*

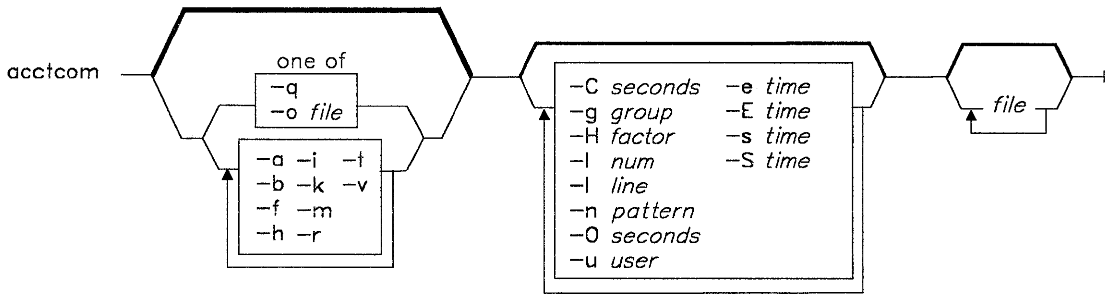
"Running System Accounting" in *Managing the AIX Operating System.*

acctcom

Purpose

Displays selected process accounting record summaries.

Syntax



OL805418

Description

The **acctcom** command reads from specified *file*s, from standard input, or from **/usr/adm/pacct** and writes records (selected by flags) to standard output. The input file format is described under **acct** in *AIX Operating System Technical Reference.*

If you do not specify any *file* parameters and if standard input is assigned to a work station or to **/dev/null** (as it is when a process runs in the background), **acctcom** reads **/usr/adm/pacct** instead of standard input.

By default, if you specify any *file* parameters, **acctcom** reads each chronologically by process completion time. Usually, **/usr/adm/pacct** is the current file that you want **acctcom** to examine. Because the **ckpacct** procedure keeps this file from growing too large, a busy system may have several **pacct** files. All but the current file have the following path name:

```
/usr/adm/pacct?
```

where *?* (question mark) is an integer incremented each time a new file is created.

Each record represents one completed process. The default display consists of the command name, user name, tty name, start time, end time, real seconds, CPU seconds, and mean memory size (in kilobytes). These default items have the following headings in the output:

```
COMMAND                  START  END  REAL   CPU    MEAN
NAME     USER  TTYNAME TIME   TIME (SECS) (SECS)  SIZE(K)
```

By using the appropriate flags, you can also display the fork/exec flag (F), the system exit value (STAT), the ratio of total CPU time to elapsed time (HOG FACTOR), the product of memory used and elapsed time (KCORE MIN), the ratio of user time to total (system and user) time (CPU FACTOR), the number of characters transferred in input/output operations (CHARS TRNSFD), and the total number of blocks read or written (BLOCKS READ).

If a process ran with superuser authority, its name is prefixed with a # (pound sign). If a process is not assigned to a known work station (for example, when **cron** runs it), a ? (question mark) appears in the TTYNAME field.

Notes:

1. The **acctcom** command only reports on processes that have finished. Use the **ps** command to examine active processes.
2. If a specified time is later than the current time, it is interpreted as occurring on the previous day.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Flags

-a	Shows some average statistics about the processes selected. The statistics will be displayed after the output records.
-b	Reads backwards, showing the most recent commands first. This flag has no effect when **acctcom** reads standard input.
-C *seconds*	Shows only processes whose total CPU time (system time + user time), exceeds number of *seconds*.
-e *time*	Selects processes existing at or before the specified time. You can use the **NLTIME** environment variable to specify the order of hours, minutes, and seconds. The default order is *hh*[*mm*[*ss*]].
-E *time*	Selects processes ending at or before the specified time. You can use the **NLTIME** environment variable to specify the order of hours, minutes, and seconds. The default order is *hh*[*mm*[*ss*]]. If you specify the same time for

	both the **-E** and **-S** flags, **acctcom** displays the process that existed at the specified time.
-f	Displays the *fork/exec* flag and the system exit value columns in the output.
-g *group*	Selects processes belonging to *group*. You can specify either the group ID or the group name.
-h	Instead of mean memory size, shows the fraction of total available CPU time consumed by the process while it ran (***hog factor***). This factor is computed as: (total CPU time)/(elapsed time)
-H *factor*	Shows only processes that exceed *factor*. (See the **-h** flag for a discussion of how this factor is calculated.)
-i	Displays columns showing the number of characters transferred in read or write operations (the I/O counts).
-I *num*	Shows only processes transferring more than *num* characters.
-k	Instead of memory size, shows total kcore minutes.
-l *line*	Shows only processes belonging to work station **/dev/***line*.
-m	Shows mean main memory size. This flag is on by default. Specifying the **-h** or **-k** flags turns off **-m**.
-n *pattern*	Shows only commands matching *pattern*, where *pattern* is a regular expression like those in the **ed** command (see page 371), except that here you can use a + (plus sign) as a special symbol for one or more occurrences of the preceding character.
-o *file*	Copies selected process records to *file*, keeping the input data format. This flag suppresses writing to standard output.
-O *seconds*	Shows only processes with CPU system time exceeding *seconds*.
-q	Does not display any output records; just displays the average statistics that are displayed with the **-a** flag.
-r	Shows CPU factor. This factor is computed as: (user-time) / (system-time + user-time).
-s *time*	Shows only those processes that existed on or after the specified time. You can use the **NLTIME** environment variable to specify the order of hours, minutes, and seconds. The default order is *hh*[*mm*[*ss*]].
-S *time*	Shows only those processes starting at or after the specified time. You can use the **NLTIME** environment variable to specify the order of hours, minutes, and seconds. The default order is *hh*[*mm*[*ss*]].

-t	Shows separate system and user CPU times.
-u *user*	Shows only processes belonging to *user*. For *user*, you can give a user ID, a login name that is converted to a user ID, a # to select processes run with superuser authority, or a ? to select processes associated with unknown user IDs.
-v	Eliminates column headings from the output.

Files

/usr/adm/pacct	Current process accounting file.
/etc/passwd	User names and user IDs.
/etc/group	Group names and group IDs.

Related Information

The following commands: "**acctdisk, acctdusg**" on page 26, "**acctcms**" on page 18, "**acctcon**" on page 24, "**acctmerg**" on page 28, "**acctprc**" on page 30, "**acct/***" on page 13, "**fwtmp**" on page 457, "**ps**" on page 786, "**runacct**" on page 848, and "**su**" on page 1026.

The **acct** system call, the **acct** and **utmp** files and the **environment** miscellaneous facility in *AIX Operating System Technical Reference.*

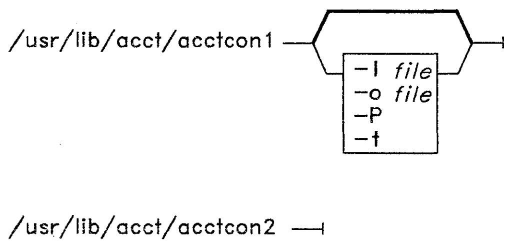
"Running System Accounting" and "Overview of International Character Support" in *Managing the AIX Operating System.*

acctcon

Purpose

Performs connect-time accounting.

Syntax



OL805233

Description

acctcon1

The **acctcon1** command converts a sequence of login and logoff records (read from standard input) to a sequence of login session records (written to standard output). its input should normally be redirected from **/usr/adm/wtmp**.

The **acctcon1** command displays, in ASCII format, the login device, user ID, login name, ***prime connect time*** (seconds), ***nonprime connect time*** (seconds), session starting time (numeric), and starting date and time (in date/time format). It also maintains a list of ports on which users are logged in. When it reaches the end of its input, it writes a session record for each port that still appears to be active. It normally assumes that its input is a current file, so that it uses the current time as the ending time for each session still in progress (see the **-t** flag on page 25).

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Flags

-l *file*	Writes to *file* a line-usage summary showing the line name, the number of minutes used, the percentage of total elapsed time used, the number of sessions charged, the number of logins, and the number of logoffs. This file helps track line usage and identify bad lines. All hang-ups, terminations of **login**, and terminations of the login shell cause the system to write logoff records, so the number of logoffs is often much higher than the number of sessions.
-o *file*	Writes to *file* an overall record for the accounting period, giving starting time, ending time, number of restarts, and number of date changes.
-p	Displays input only, showing line name, login name, and time in both numeric and date/time formats.
-t	Uses the last time found in the input as the ending time for any current processes instead of the current time. This is necessary in order to have reasonable and repeatable values for noncurrent files.

acctcon2

The **acctcon2** command converts a sequence of login session records, produced by the **acctcon1** command, into total accounting records.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Files

/usr/adm/wtmp	Login/logoff history file.

Related Information

The following commands: "**acctdisk, acctdusg**" on page 26, "**acctcms**" on page 18, "**acctcom**" on page 20, "**acctmerg**" on page 28, "**acctprc**" on page 30, "**acct/***" on page 13, "**fwtmp**" on page 457, "**init**" on page 521, "**login**" on page 584, and "**runacct**" on page 848.

The **acct** system call and the **acct** and **utmp** files in *AIX Operating System Technical Reference.*

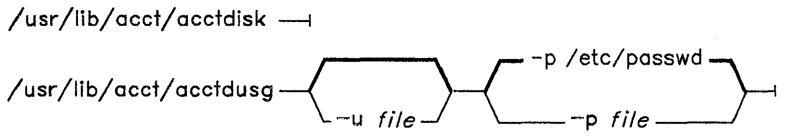
"Running System Accounting" in *Managing the AIX Operating System.*

acctdisk, acctdusg

Purpose

Performs disk-usage accounting.

Syntax



OL805192

Description

acctdisk

The **acctdisk** command reads lines from standard input that contain a user ID, the user's login name, and the number of disk blocks occupied by the user's files. It converts these lines to total accounting records that can be merged with other accounting records and writes those records to standard output.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

acctdusg

The **acctdusg** command reads a list of file names from standard input (usually piped from a **find / -print** command), computes disk resource usage (including ***indirect blocks***) using the login name of the owner of the files, and writes the results to standard output.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Flags

-p *file*	Searches *file* for login names and numbers, instead of searching **/etc/passwd**.
-u *file*	Places in *file* records of file names for which it does not charge.

Files

/etc/passwd	Used to convert login names to user IDs.
/usr/lib/acct	Directory holding all accounting commands.

Related Information

The following commands: "**acct/***" on page 13, "**acctcms**" on page 18, "**acctcom**" on page 20, "**acctcon**" on page 24, "**acctmerg**" on page 28, "**acctprc**" on page 30, "**fwtmp**" on page 457, and "**runacct**" on page 848.

The **acct** system call and the **acct** and **utmp** files in *AIX Operating System Technical Reference.*

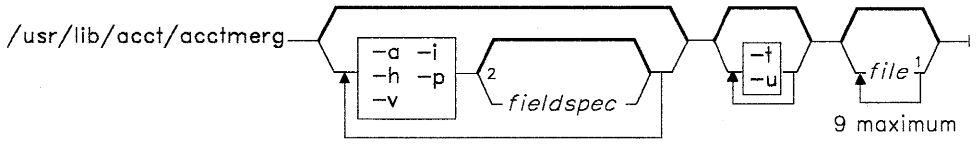
"Running System Accounting" in *Managing the AIX Operating System.*

acctmerg

Purpose

Merges total accounting files.

Syntax



[1]acctmerg always reads standard imput in addition to any named *files*.
[2]Do not put a blank between these items.

OL805234

Description

The **acctmerg** command reads records from standard input and up to nine additional *file*s, all in the **tacct** binary format or the **tacct** ASCII format. It merges these by adding records with keys (normally user ID and name) that are identical, and expects the input records to be sorted by those key fields. It writes these merged records to standard output.

The optional *fieldspec*s allow you to select input or output fields. A field specification is a comma-separated list of fields or field ranges. Field numbers are in the order specified in the **tacct** file in *AIX Operating System Technical Reference*, with array sizes, except for the *ta_name* characters, taken into account. For example, `-h2-3,7,15-13,2` displays the login name, prime CPU and connect times, fee, queueing system, and disk usage data, and the login name again, in that order, with column headings. The default specification is "all fields" (`1-18` or `1-`), which produces very wide output lines containing all the available accounting data.

Queueing system, disk usage, or fee data can be converted into **tacct** records using the **-i***fieldspec* argument. For example, disk accounting records, produced by **acctdisk**, consist of lines containing the user ID, login name, number of blocks, and number of disk samples (always one). A file, **dacct**, containing these records can be merged into an existing total accounting file, **tacct**, with:

```
acctmerg  -i1-2,13,18  <dacct |. acctmerg  tacct  >output
```

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Flags

-a[*fieldspec*]	Produces output in the form of ASCII records.
-h[*fieldspec*]	Displays column headings. This flag implies **-a** but is effective with **-p** or **-v**.
-i[*fieldspec*]	Expects input files composed of ASCII records.
-p[*fieldspec*]	Displays input without processing.
-t	Produces a single record that contains the totals of all input.
-u	Summarizes by user ID rather than by user name.
-v[*fieldspec*]	Produces output in ASCII format, with more precise notation for floating-point numbers.

Example

The following sequence is useful for making repairs to any file in **tacct** format:

```
acctmerg  -v  <file1  >file2
     edit file2 as desired . . .
acctmerg  -a  <file2  >file1
```

Related Information

The following commands: "**acct/***" on page 13, "**acctcms**" on page 18, "**acctcom**" on page 20, "**acctcon**" on page 24, "**acctdisk, acctdusg**" on page 26, "**fwtmp**" on page 457, "**acctprc**" on page 30, and "**runacct**" on page 848.

The **acct** system call and the **acct** and **utmp** files in *AIX Operating System Technical Reference.*

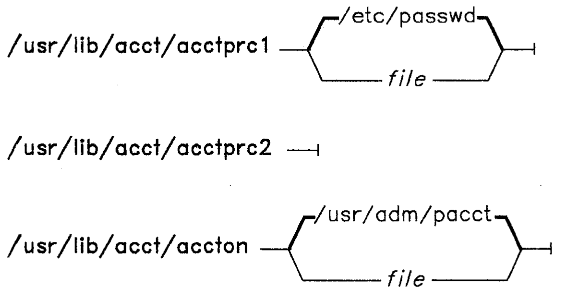
"Running System Accounting" in *Managing the AIX Operating System.*

acctprc

Purpose

Performs process accounting.

Syntax



OL805235

Description

acctprcl

The **acctprcl** command reads records from standard input that are in the **acct** format (described in *AIX Operating System Technical Reference*), adds the login names that correspond to user IDs, and then writes an ASCII record to standard output. This record contains the user ID, login name, prime CPU time, nonprime CPU time, the total number of characters transferred (in 512-byte units), the total number of blocks read and written, and mean memory size (in 64-byte units) for each process.

If specified, *file* contains a list of login sessions in **utmp** format (described in *AIX Operating System Technical Reference*), sorted by user ID and login name. By default, **acctprcl** gets login names from the password file, **/etc/passwd**. The information in *file* helps distinguish among different login names that share the same user ID.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

---

acctprc2

The **acctprc2** command reads (from standard input) the records written by **acctprcl**, summarizes them by user ID and name, and writes the sorted summaries to standard output as total accounting records.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

accton

The **accton** command without arguments turns process accounting off. If you specify *file* (the name of an existing file), the kernel adds process accounting records to it (**/usr/adm/pacct** by default).

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Files

/etc/passwd	Password file; contains user IDs.
/usr/adm/pacct	Contains process accounting records.

Related Information

The following commands: "**acct/***" on page 13, "**acctdisk, acctdusg**" on page 26, "**acctcms**" on page 18, "**acctcom**" on page 20, "**acctcon**" on page 24, "**acctmerg**" on page 28, "**fwtmp**" on page 457, and "**runacct**" on page 848.

The **acct** system call and the **acct** and **utmp** files in *AIX Operating System Technical Reference.*

"Running System Accounting" in *Managing the AIX Operating System.*

actman

Purpose

Permits interaction with multiple virtual terminals.

Syntax

```
actman —|
```

OL805323

Description

The **actman** command is the ***Activity Manager*** for the AIX Operating System. It is normally run by the AIX logger in the same manner as any program listed in the **/etc/passwd** file. Once started by the logger, **actman** creates the initial shell (**/bin/sh**) and monitors the number of open virtual terminals until all have been closed. It then exits to the AIX **init** process. If you try to end the initial shell when other virtual terminals are still open, **actman** restarts the initial shell.

To take advantage of the multiple virtual terminal capability, use the **open** command (see page 728) to execute another shell in a separate virtual terminal.

Notes:

1. You must log off of each existing shell to end your login session.
2. You do not need an Activity Manager if you do not have virtual terminal capabilities. Thus if you do not log in from the local console, **actman** overlays itself with the initial shell.

Related Information

The following command: "**open**" on page 728.

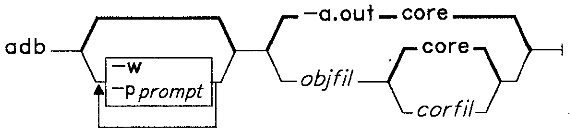
"Using Display Station Features" in *Using the AIX Operating System.*

adb

Purpose

Provides a general purpose debugger.

Syntax



OL805465

Description

The **adb** command provides a debugger for C and assembler language programs. With it, you can examine object and core files and provide a controlled environment for running a program.

Normally, *objfil* is an executable program file that contains a symbol table. If *objfil* does not contain a symbol table, the symbolic features of **adb** cannot be used, although the file can still be examined. The default *objfil* is **a.out**.

The *corfil* is assumed to be a core image file produced by running *objfil*. The default *corfil* is **core**.

While running, **adb** takes input from standard input and writes to standard output. **adb** does not recognize the **Quit** or **Interrupt** keys. These keys cause **adb** to wait for a new command.

In general, requests to **adb** are of the form

[*address*] [,*count*] [*command*] [;]

where *address* and *count* are expressions. The default *count* is 1. If *address* is specified, then the expression . (dot) is set to *address*.

The interpretation of an address depends on the context in which it is used. If a subprocess is being debugged, addresses are interpreted in the usual way in the address space of the subprocess. For more information, see "Addresses" on page 39.

You can enter more than one command at a time by separating the commands with a **;** (semicolon).

Expressions

.	Specifies the last address used by a command; this is also known as the current address.
+	Increases the value of . (dot) by the current increment.
^	Decreases the value of . (dot) by the current increment.
"	Specifies the last address typed by a command.
integer	Specifies an octal number if *integer* begins with 0o, a hexadecimal number if preceded by 0x or #, or a decimal number if preceded by 0t; otherwise, a number interpreted in the current radix. The radix is initially 16.
'cccc'	Specifies the ASCII value of up to 4 characters. \ (slash) can be used to escape an ′ (apostrophe).
< *name*	Reads the current value of *name*. *name* is either a variable name or a register name. **adb** maintains a number of variables (see "Variables" on page 39) named by single letters or digits. If *name* is a register name, the value of the register is obtained from the system header in *corfil.* The register names are **r0...r15, pc, ics, cs, mq**; the names **fp**, **pcp**, and **link** are recognized as synonyms for **r1**, **r14**, and **r15**.
symbol	Specifies a sequence of upper- or lower-case letters, underscores, or digits, not starting with a digit. The value of the *symbol* is taken from the symbol table in *objfil.* An initial _ (underscore) is prefixed to *symbol* if needed.
_symbol	Specifies, in C, the true name of an external symbol begins with _ (underscore), as does the name of the constant pool of an external function. It may be necessary to use this name to distinguish it from internal or hidden variables of a program.
.symbol	Specifies the entry point of the function named by *symbol.*
routine.name	Specifies the address of the variable *name* in the specified C routine. Both *routine* and *name* are *symbols.* If *name* is omitted, the value is the address of the most recently activated C stack frame corresponding to *routine.*
(*exp*)	Specifies the value of the expression *exp.*

Operators

Integers, symbols, variables, and register names can be combined with the following operators:

Unary

**exp*	Contents of location addressed by *exp* in *corefile.*
@exp	Contents of the location addressed by *exp* in *objfil.*
-exp	Integer negation.
~exp	Bitwise complement.

Binary

e1+e2	Integer addition.
e1-e2	Integer subtraction.
*e1*e2*	Integer multiplication.
e1%e2	Integer division.
e1&e2	Bitwise conjunction.
e1|e2	Bitwise disjunction.
e1#e2	*e1* rounded up to the next multiple of *e2.*

Binary operators are left associative and are less binding than unary operators.

Commands

You can display the contents of a text or data segment with the **?** (question mark) or the / (slash) command. The = (equal) command displays a given address in the specified format *f*. (The commands **?** and / may be followed by * (asterisk); see "Addresses" on page 39.)

?*f*	Displays, in the format *f*, the contents of the *objfil* starting at *address*. The value of **.** (dot) increases by the sum of the increment for each format letter.
/*f*	Displays, in the format *f*, the contents of the *corfil* starting at *address*. The value of **.** (dot) increases by the sum of the increment for each format letter.
=*f*	Displays the value of *address* in the format *f*. The **i** and **s** format letters are not meaningful for this command.

The format consists of one or more characters that specify print style. Each format character may be preceded by a decimal integer that is a repeat count for the format character. While stepping through a format, **.** (dot) increments by the amount given for each format letter. If no format is given, the last format is used.

The format letters available are as follows:

o 2	Prints 2 bytes in octal.
O 4	Prints 4 bytes in octal.
q 2	Prints 2 bytes in the current radix, unsigned.
Q 4	Prints 4 bytes in the current radix, unsigned.
d 2	Prints in decimal.
D 4	Prints long decimal.
x 2	Prints 2 bytes in hexadecimal.
X 4	Prints 4 bytes in hexadecimal.
u 2	Prints as an unsigned decimal number.
U 4	Prints long unsigned decimal.
b 1	Prints the addressed byte in the current radix, unsigned.
c 1	Prints the addressed character.
C 1	Prints the addressed character using the following escape conventions: 1. Prints control characters as ~ followed by the corresponding printing character. 2. Prints nonprintable characters as ~<*n*> where *n* is a hexadecimal value of the character. The character ~ prints as ~ ~.
s *n*	Prints the addressed character until a zero character is reached.
S *n*	Prints a string using the ~ escape convention. *n* specifies the length of the string including its zero terminator.
Y 4	Prints 4 bytes in date format (see "**ctime**" in *AIX Operating System Technical Reference*).
i n	Prints as instructions. *n* is the number of bytes occupied by the instruction.
a 0	Prints the value of **.** (dot) in symbolic form. Symbols are checked to ensure that they have an appropriate type as follows: **/** local or global data symbol; **?** local or global text symbol; **=** local or global absolute symbol
p 4	Prints the addressed value in symbolic form using the same rules for symbol lookup as **a**.
t 0	When preceded by an integer, tabs to the next appropriate tab stop. For example, **8t** moves to the next 8-space tab stop.

r 0	Prints a space.
n 0	Prints a new line.
"..." 0	Prints the enclosed string.
^	Decreases **.** (dot) by the current increment. Nothing prints.
+	Increases **.** (dot) by 1. Nothing prints.
−	Decreases **.** (dot) decrements by 1. Nothing prints.

newline Repeats the previous command incremented with a *count* of 1.

[?/]l*value mask* Words starting at **.** (dot) are masked with *mask* and compared with *value* until a match is found. If **L** is used, the match is for 4 bytes at a time instead of 2. If no match is found, **.** (dot) is unchanged; otherwise **.** (dot) is set to the matched location. If *mask* is omitted, -1 is used.

[?/]**w***value...* Writes the 2-byte *value* into the addressed location. If the command is **W**, write 4 bytes. If the command is **V**, write 1 byte. Alignment restrictions may apply when using **w** or **W**.

[?/]**m** *b1 e1 f1*[?/] Records new values for *b1, e1, f1.* If less than three expressions are given then the remaining map parameters are left unchanged. If the ? or / is followed by * then the second segment (*b2, e2, f2*) of the mapping is changed. If the list is terminated by ? or / then the file (*objfil* or *corfil* respectively) is used for subsequent requests. (For example, **/m?** causes / to refer to *objfil*).

>*name* Assigns . (dot) to the variable or register *name.*

! Calls a shell to read the rest of the line following !.

$*modifier* Miscellaneous commands. The available *modifiers* are:

<*file*	Reads commands from *file* and returns to the standard input.
>*file*	Sends output to *file.* If *file* is omitted, output returns to the standard output. *file* is created if it does not exist.
r	Prints the general registers and the instruction addressed by **pc** and sets **.** (dot) to **pc**.
b	Prints all breakpoints and their associated counts and commands.
c	C stack back trace. If *address* is given, it is taken as the address of the current frame (instead of using the frame pointer register). If **C** is used, then the names and values of all automatic and static variables are printed for each active function. If *count* is given then only the first *count* frames are printed.

e	Prints the names and values of external variables.
w	Sets the output page width for *address*. The default is 80.
s	Sets the limit for symbol matches to *address*. The default is 255.
o	Sets the current radix to 8.
d	Sets the current radix to *address* or 16, if none is specified.
q	Exits **adb**.
v	Prints all non-zero variables in octal.
m	Prints the address map.
p	Uses the remainder of the line as a prompt string.

:modifier Manages a subprocess. Available *modifier*s are:

b*c*	Sets the breakpoint at *address*. The breakpoint runs *count* -1 times before causing a stop. Each time the breakpoint is encountered, the command *c* runs. If this command sets **.** (dot) to 0, the breakpoint causes a stop.
d	Deletes the breakpoint at *address*.
r	Runs *objfil* as a subprocess. If *address* is given explicitly, the program is entered at this point; otherwise, the program is entered at its standard entry point. *count* specifies how many breakpoints are to be ignored before stopping. Arguments to the subprocess may be supplied on the same line as the command. An argument starting with < or > causes the standard input or output to be established for the command. On entry to the subprocess, all signals are turned on.
c*s*	Continues the subprocess with signal *s* (see the **signal** system call in *AIX Operating System Technical Reference*). If *address* is given, the subprocess is continued at this address. If no signal is specified, the signal that caused the subprocess to stop is sent. Breakpoint skipping is the same as for **r**.
s*s*	Continues the subprocess in single steps *count* times. If there is no current subprocess, *objfil* is run as a subprocess. In this case no signal can be sent; the remainder of the line is treated as arguments to the subprocess.
k	Stops the current subprocess, if one is running.

Variables

adb provides a number of variables. On entry to **adb**, the following variables are set from the system header in the *corfil.* If *corfil* does not appear to be a **core** file, then these values are set from *objfil.*

b	The base address of the data segment
d	The size of the data segment
e	The entry address of the program
m	The "magic" number (0405, 0407, 0410, or 0411)
s	The size of the stack segment
t	The size of the text segment.

Addresses

The address in a file associated with a written address is determined by a mapping associated with that file. Each mapping is represented by two triples (*b1, e1, f1*) and (*b2, e2, f2*). The *file address* that corresponds to a written *address* is calculated as follows:

$b1 \leq address < e1 => file\ address = address + f1 - b1$

or

$b2 \leq address < e2 => file\ address = address + f2 - b2$

Otherwise, the requested *address* is not legal. In some cases (for example, programs with separated I and D space) the two segments for a file may overlap. If a **?** or **/** is followed by an *****, then only the second triple is used.

The initial setting of both mappings is suitable for normal **a.out** and **core** files. If either file is not of the kind expected, then for that file *b1* is set to 0, *e1* is set to the maximum file size, and *f1* is set to 0; in this way, the whole file can be examined with no address translation.

In order for **adb** to be used on large files, all appropriate values are kept as signed 32-bit integers.

Flags

-p*prompt* Sets the prompt used by **adb** to *prompt.* If the prompt includes spaces, enclose the prompt in quotation marks.

-w Opens the *objfil* and *corfil* for writing. This flag makes either file if they do not exist.

Files

/dev/mem
/dev/swap
a.out
core

Related Information

The **ptrace** system call in *AIX Operating System Technical Reference.*

The **a.out** and **core** files in *AIX Operating System Technical Reference.*

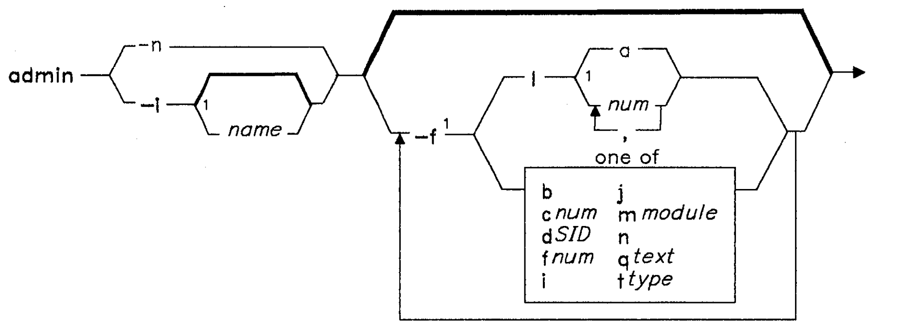
admin

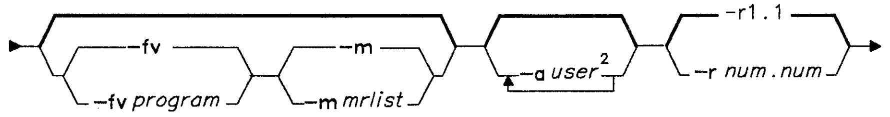
Purpose

Creates and initializes SCCS files.

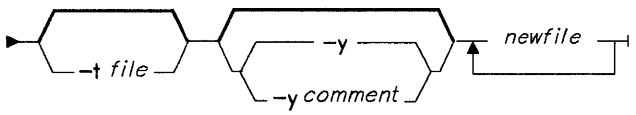
Syntax

To Create SCCS Files:





OL805376

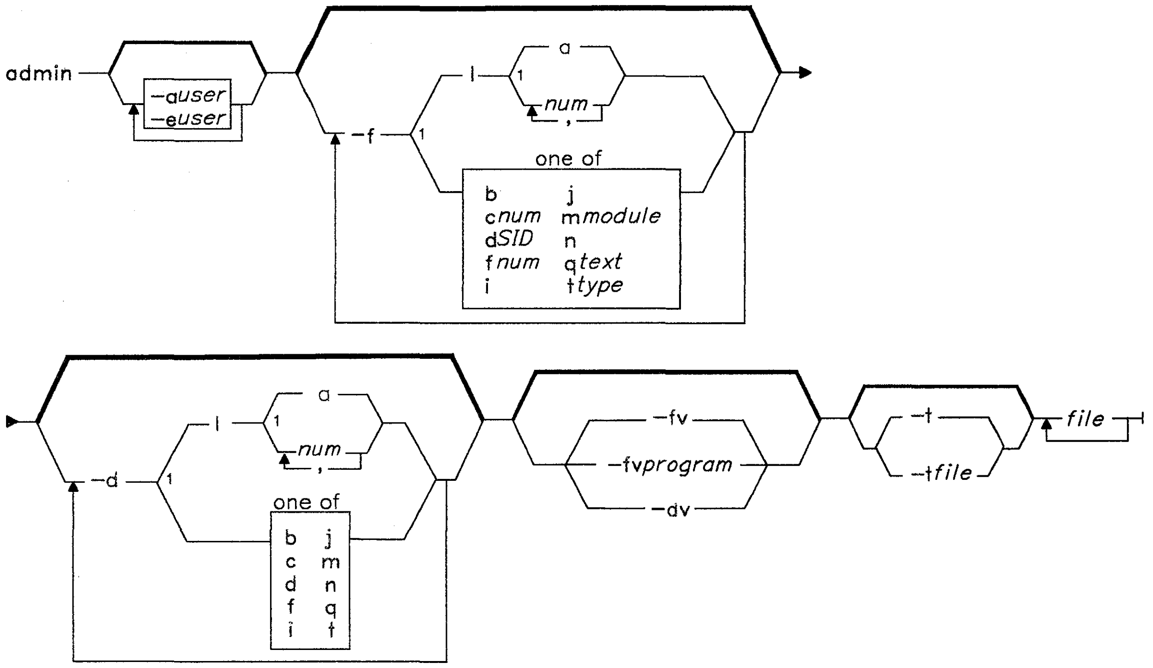


OL805160

[2] If −a is never used to specify **users**, then any user can run **get** −e on the file.

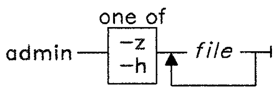
OL805417

To Change Existing SCCS Files:



OL805385

To Check and Correct Damaged SCCS Files:



OL805158

[1] Do not put a blank between these items.

OL805308

Description

The **admin** command creates new Source Code Control System (SCCS) *file*s or changes specified parameters in existing SCCS *file*s. These parameters control how the **get** command builds the files that you can edit. They also provide information about who can access the file, who can make changes, and when changes were made.

If the named *file* exists, **admin** modifies its parameters as specified by the flags. If it does not exist and you supply the **-i** or the **-n** flag, **admin** creates the new file and provides default values for unspecified flags. If you specify a directory name for *file*, **admin** performs the requested actions on all SCCS files in that directory (all files with the **s.** prefix). If you specify a **-** (minus) as a *file* name, **admin** reads standard input and interprets each line as the name of an SCCS file. An end-of-file character (**Ctrl-D**) ends input.

The **admin** command is most often used to create new SCCS files without setting parameters. See "Examples" on page 46 for the syntax used to create an SCCS file with no parameters set in the new file.

If you are not familiar with the delta numbering system, see *AIX Operating System Programming Tools and Interfaces* for more information.

SCCS File Conventions

All SCCS file names must have the form **s.***name*. New SCCS files are created with read-only permission. You must have write permission in the directory to create a file (see "**chmod**" on page 160 for an explanation of file permissions). **admin** writes to a temporary x-file, which it calls **x.***name*. The x-file has the same permissions as the original SCCS file if it already exists, and it is read-only if **admin** creates a new file. After successful completion of **admin**, the x-file is moved to the name of the SCCS file. This ensures that changes are made to the SCCS file only if **admin** does not detect any errors while it is running.

Directories containing SCCS files should be created with permission code 755 (read, write, and execute permissions for owner, read and execute permissions for group members and others). SCCS files themselves should be created as read-only files (444). With these permissions, only the owner can use non-SCCS commands to modify SCCS files. If a group can access and modify the SCCS files then the directories should include group write permission.

The **admin** command also uses a temporary lock file (called **z.***name*), to prevent simultaneous updates to the SCCS file by different users. See "SCCS Files" on page 478 for additional information on the **z.***name* file.

The following table contains the header flags that can be set with the **-f** flag and unset with the **-d** flags (see page 45). The header flags control the format of the g-file created with the **get** command (see "SCCS Files" on page 478 for details on the g-file).

Header Flag	**Header Flag Purpose**
b	Lets you use the **-b** flag of a **get** command to create branch deltas.
c*num*	Makes *num* the highest release number that a **get -e** can use. The value of *num* must be less than or equal to 9999. (Its default value is 9999.)
f*num*	Makes *num* the lowest release number that a **get -e** can retrieve. *num* must be greater than 0 and less than 9999. (Its default value is 1.)
d*SID*	Makes *SID* the default delta supplied to a **get** command.
i	Treats the `No id keywords (ge6)` message issued by the **get** or **delta** command as an error (see "Identification Keywords" on page 480).
j	Permits concurrent **get** commands for editing the same SID of an SCCS file. This allows multiple concurrent updates to the same version of the SCCS file.
l*num*[,*num*] . . .	Locks the releases specified by *num* . . . against editing, so that a **get -e** against one of these releases fails. You can lock all releases against editing by specifying **-fla** and unlock specific releases with the **-d** flag.
n	Causes **delta** to create a null delta in any releases that are skipped when a delta is made in a new release. For example, if you make delta 5.1 after delta 2.7, releases 3 and 4 will be null. The resulting null deltas can serve as points from which to build branch deltas. Without this flag, skipped releases do not appear in the the SCCS file.
q*text*	Substitutes *text* for all occurrences of the %Q% keyword in an SCCS text file retrieved by a **get** command. (See "Identification Keywords" on page 480 for more information on keywords.)
m*module*	Substitutes *module* for all occurrences of the %M% keyword in an SCCS text file retrieved by a **get** command. The default *module* is the name of the SCCS file without the **s.** prefix.
t*type*	Substitutes *type* for all %Y% keywords in a g-file retrieved by a **get**.
v[*program*]	Makes **delta** prompt for Modification Request (MR) numbers as the reason for creating a delta. *program* specifies the name of an MR number validity checking program (see "**delta**" on page 310). If **v** is set in the SCCS file, the **admin -m** flag must also be used, even if its value is null.

Figure 1. SCCS Header Flags

Flags

You can enter the flags and input file names in any order. All flags apply to all the files.

-a*user*	Adds the specified *user* to the list of users that can make sets of changes (***deltas***), to the SCCS file. *user* can be either a user name, a group name, or a group ID. Specifying a group name or number is the same as specifying the names of all users in that group. You can specify more than one **-a** flag on a single **admin** command line. If an SCCS file contains an empty user list, then anyone can add deltas. If a file has a user list, the creator of the file must be included in the list in order for the creator to make deltas to the file.
-d*hdrflag*	Removes the specified header flag from the SCCS file. You can specify this flag only with existing SCCS files. You can also specify more than one **-d** flag in a single **admin** command. See Figure 1 on page 44 for the header flags that **admin** recognizes.
-e*user*	Removes the specified *user* from the list of users allowed to make deltas to the SCCS file. Specifying a group ID is equivalent to specifying all *user* names common to that group. You can specify several **-e** flags on a single **admin** command line.
-f*hdrflag*[*value*]	Places the specified header flag and value in the SCCS file. You can specify more than one header flag in a single **admin** command. See Figure 1 on page 44 for the header flags that **admin** recognizes.
-h	Checks the structure of the SCCS file and compares a newly computed checksum with the checksum that is stored in the first line of the SCCS file. When the checksum value is not correct, the file has been improperly modified or has been damaged. This flag helps you detect damage caused by the improper use of non-SCCS commands to modify SCCS files, as well as accidental damage. The **-h** flag prevents writing to the file, so it cancels the effect of any other flags supplied. If an error message is returned indicating the file is damaged, use the **-z** flag to recompute the checksum. Then test to see if the file is corrected by using the **-h** flag again.
-i[*name*]	Gets the text for a new SCCS file from *name*. This text is the first delta of the file. If you specify the **-i** flag but you omit the file name, **admin** reads the text from standard input until it reaches end-of-file (**Ctrl-D**). If you do not specify the **-i** flag, but you do specify the **-n** flag, **admin** creates an empty SCCS file. **admin** can only create one file containing text at a time. If you are creating two or more SCCS files with one call to **admin**, you must use the **-n** flag, and the SCCS files created are empty.

-m[*mrlist*]	Specifies a list of Modification Requests (MR) numbers to be inserted into the SCCS file as the reason for creating the initial delta. The **v** flag must be set. The MR numbers are validated if the **v** flag has a value (the name of an MR number validation program). **admin** reports an error if the **v** flag is not set or if MR validation fails.
-n	Creates a new, empty SCCS file. Do not specify this flag when you use the **-i** flag.
-r*num.num*	Inserts the initial delta into *num.num*, the release and version respectively. You can specify **-r** only if you also specify the **-i** or **-n** flag. If you do not specify this flag, the initial delta becomes Release 1, Version 1. Use this flag only when creating an SCCS file.
-t[*file*]	Takes descriptive text for the SCCS file from *file*. If you use **-t** when creating a new SCCS file, you must supply a file name. In the case of existing SCCS files: • Without a file name, **-t** causes removal of the descriptive text (if any) currently in the SCCS file. • With a file name, **-t** causes text in the named file to replace the descriptive text (if any) currently in the SCCS file.
-y[*comment*]	Inserts *comment* text into the initial delta in a manner identical to that of the **delta** command. Use this flag only when you create an SCCS file. If you do not specify a comment, **admin** inserts a line of the following form: `date and time created` *YY/MM/DD HH:MM:SS* `by login`
-z	Recomputes the SCCS file checksum and stores it in the first line of the SCCS file (see the **-h** flag on page 45). **Warning:** Using **admin** with this flag on a damaged file can prevent future detection of the damage. This flag should only be used if the SCCS file is changed using non-SCCS commands because of a serious error.

Examples

1. To create an empty SCCS file named **s.prog.c**:

```
admin  -n  s.prog.c
```

2. To convert an existing text file into an SCCS file:

```
admin  -iprogram.c  s.prog.c
```

This converts the text file `program.c` into the SCCS file `s.prog.c`. The original file remains intact, but it is no longer needed. You must rename or delete it before you can use the **get** command on `s.prog.c`.

Related Information

The following commands: "**delta**" on page 310, "**ed**" on page 371, "**get**" on page 477, "**help**" on page 513, "**prs**" on page 781, and "**what**" on page 1213.

The **sccsfile** file in *AIX Operating System Technical Reference.*

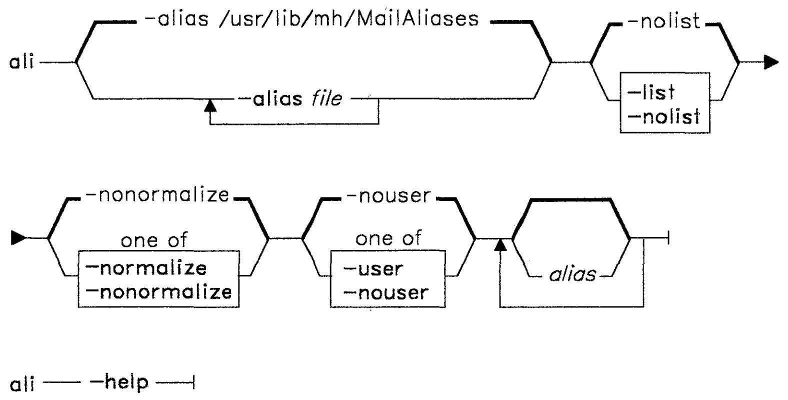
"Maintaining Different Versions of a Program" in *AIX Operating System Programming Tools and Interfaces.*

ali

Purpose

Lists mail aliases and their addresses.

Syntax



AJ2FL150

Description

The **ali** command is used to list mail aliases and the addresses that the aliases represent. **ali** is part of the MH (Message Handling) package and can be used with other MH and AIX commands.

The **ali** command searches the specified mail alias files for each given *alias*, and writes to standard output the addresses of each *alias*. If you specify the **-user** flag, **ali** interprets the **alias** arguments as actual addresses, searches the alias files for the addresses, and writes to standard output the aliases that contain definitions of the addresses. Thus, if you want to find the address of an alias, use the default **-nouser** flag. If you want to find the aliases that represent an address, use the **-user** flag.

Flags

-alias *file*	Specifies that *file* is a mail alias file to be searched for each given *alias*. The default alias file is **/usr/lib/mh/MailAliases**.
-help	Displays help information for the command.

-list	Displays each address on a separate line.
-nolist	Displays addresses separated by commas on as few lines as possible. This flag is the default.
-nonormalize	Does not attempt to convert local nicknames of hosts to their official host names. This flag is the default.
-normalize	Attempts to convert local nicknames of hosts to their official host names.
-nouser	Lists the addresses that the specified aliases represent. This flag is the default.
-user	Lists the aliases that contain the specified addresses. When the **-user** and **-nonormalize** flags are used together, the result may be a partial list of aliases that contain the specified addresses.

Files

/usr/lib/mh/MailAliases	The default mail alias file.
$HOME/.mh_profile	The MH user profile.
/etc/passwd	List of users.
/etc/group	List of groups.

Related Information

The following commands: "**comp**" on page 185, "**dist**" on page 336, "**forw**" on page 438, "**repl**" on page 821, "**send**" on page 893, "**whom**" on page 1222.

The **mh-alias** and **mh-profile** files in *AIX Operating System Technical Reference.*

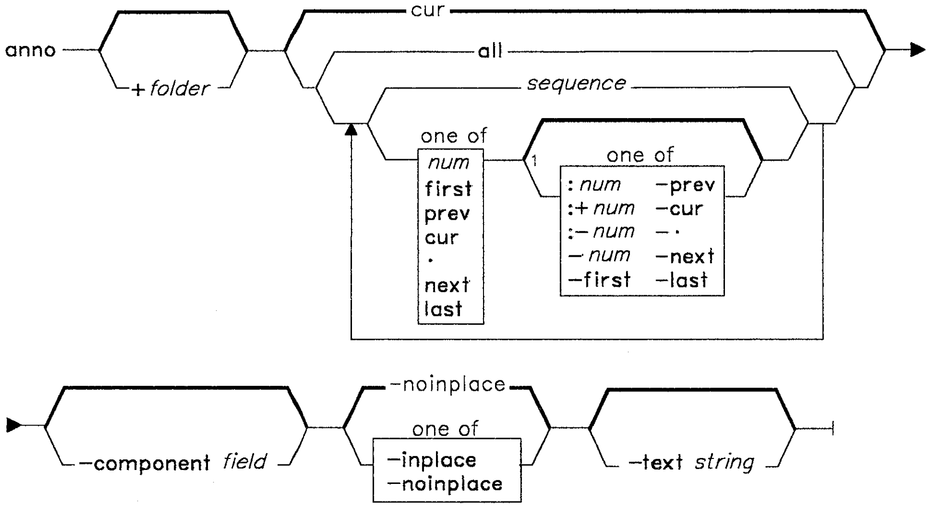
The "Overview of the Message Handling Package" in *Managing the AIX Operating System.*

anno

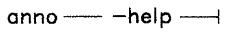
Purpose

Annotates messages.

Syntax



AJ2FL221



AJ2FL166

[1] Do not put a blank between these items.

OL805308

Description

The **anno** command is used to annotate messages with specified text and dates. **anno** is part of the MH (Message Handling) package and can be used with other MH and AIX commands.

The **anno** command annotates messages with the lines:

```
field:date
field:body
```

Although **dist**, **forw**, and **repl** enable you to perform annotations, their annotations are limited to adding distribution information to messages. **anno** enables you to perform arbitrary annotations. The annotation fields must contain alphanumeric characters and dashes only.

Flags

-component *field* Specifies the field name for the annotation text. The field name must be a valid message field name, consisting of alphanumeric characters and dashes only. If you do not specify this flag, **anno** prompts you for the name of the field.

+*folder msgs* Specifies the messages that you want to annotate. *msgs* can be several messages, a range of messages, or a single message. You can use the following message references when specifying *msgs*:

num	**first**	**prev**
cur	**.**	**next**
last	**all**	*sequence*

The default message is the current message in the current folder. If several messages are specified, the first message annotated becomes the current message. If you specify a folder, that folder becomes the current folder.

-help Displays help information for the command.

-inplace Forces annotation to be done in place in order to preserve links to the annotated messages.

-noinplace Does not perform annotation in place. This flag is the default.

-text *string* Specifies the text to be annotated to the messages.

Profile Entries

Current-Folder:	Sets your default current folder.
Path:	Specifies your *user_mh_directory*.

Files

$HOME/.mh_profile	The MH user profile.

Related Information

The following commands: "**dist**" on page 336, "**forw**" on page 438, "**repl**" on page 821.

The **mh-profile** file in *AIX Operating System Technical Reference.*

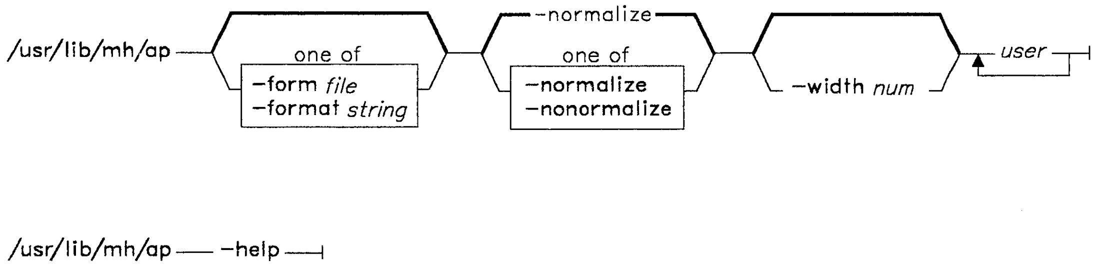
The "Overview of the Message Handling Package" in *Managing the AIX Operating System.*

ap

Purpose

Parses and reformats addresses.

Syntax



AJ2FL224

Description

The **ap** command is used to parse and reformat addresses. **ap** is not designed to be run directly by the user; it is designed to be called by other programs. The **ap** command is typically called by its full path name. The **ap** command is part of the MH (Message Handling) package.

The **ap** command parses each string specified as an address and attempts to reformat the string. The default output format for **ap** is the ARPA RFC822 standard. When the default format is used, **ap** displays an error message for each string it is unable to parse.

Flags

-form *file*	Reformats the given addresses into the alternate format described in *file*.
-format *string*	Reformats the given addresses into the alternate format specified by *string*. The default format string is: %<{error}%{error}:%{*address*}%|%(putstr(proper{*address*}))%>
-help	Displays help information for the command.
-nonormalize	Does not attempt to convert local nicknames of hosts to their official host names.

-normalize	Attempts to convert local nicknames of hosts to their official host names. This flag is the default.
-width *num*	Sets the maximum number of columns that **ap** uses to display dates and error messages. The default is the width of the display.

Files

$HOME/.mh_profile	The MH user profile.
/usr/lib/mh/mtstailor	The MH tailor file.

Related Information

Other MH commands: "**ali**" on page 48, "**dp**" on page 352, "**scan**" on page 871.

The **mh-alias**, **mh-format**, and **mh-profile** files in *AIX Operating System Technical Reference.*

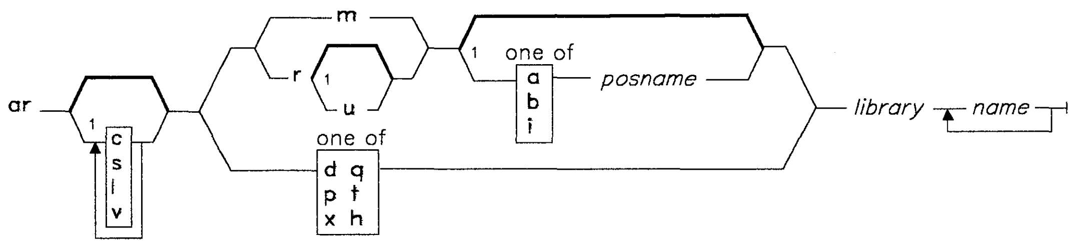
The "Overview of the Message Handling Package" in *Managing the AIX Operating System.*

ar

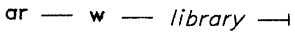
Purpose

Maintains portable libraries used by the linkage editor.

Syntax



OL805377



OL805349

1 Do not put a blank between these items.

OL805308

Description

The **ar** command combines one or more named files into a single *library* file written in **ar** archive format. When **ar** creates a library, it creates headers in a transportable format; when it creates or updates a library, it rebuilds the symbol table that the ***linkage editor*** (the **ld** command) uses to make efficient multiple passes over object file libraries. See the **ar** file entry in *AIX Operating System Technical Reference* for information on the format and structure of portable archives and symbol tables.

Flags

In an **ar** command, you must list all selected flags together on the command line without blanks between them. You must specify one from the set **dhmpqrtxw**. You can also specify any number of optional flags from the set **abcilsuv**. If you select a positioning flag (**a**, **b**, or **i**), you must also specify the name of a file within *library* (*posname*), immediately following the flag list and separated from it by a blank.

a *posname*	Positions the named files after the existing file identified by *posname.*
b *posname*	Positions the named files before the existing file identified by *posname.*
c	Suppresses the normal message that is produced when *library* is created.
d	Deletes the named files from the library.
h	Sets the modification times in the member headers of the named files to the current date and time. If you do not specify any file names, **ar** sets the time stamps of all member headers.
i *posname*	Positions the named files before the existing file identified by *posname* (same as **b**).
l	Places temporary files in the current (local) directory instead of directory **/tmp**.
m	Moves the named files to some other position in the library. By default, it moves the named files to the end of the library. Use a positioning flag (**abi**) to specify some other position.
p	Writes to the standard output the contents of the named *file*s or all files in a *library* if you do not specify any files.
q	Adds the named files to the end of the library. Positioning flags, if present, do not have any effect. Note that this process does not check to see if the named files are already in the library. In addition, if you name the same file twice, it may be put in the library twice.
r	Replaces a named file if it already appears in the library. Since the named files occupy the same position in the library as the files they replace, a positioning flag does not have any additional effect. When used with the **u** flag (update), **r** replaces only files modified since they were last added to the library file. If a named file does not already appear in the library, **ar** adds it. In this case, positioning flags do affect placement. If you do not specify a position, new files are placed at the end of the library. If you name the same file twice, it may be put in the library twice.
s	Forces the regeneration of the library symbol table whether or not **ar** modifies the library contents. Use this flag to restore the library symbol table after using the **strip** command on the library.
t	Writes to the standard output a table of contents for the library. If you specify file names, only those files appear. If you do not specify any files, **t** lists all files in the library.
u	Copies only files which have been changed since they were last copied (see the **r** flag discussed previously).

v	Writes to standard output a verbose file-by-file description of the making of the new library. When used with the **t** flag, it gives a long listing similar to that of the **ls -l** command, described under "**ls**" on page 595. When used with the **x** flag, it precedes each file with a name. When used with the **h** flag, it lists the member name and the updated modification times. The environment variables **NLLDATE** and **NLTIME** control the format of the archive date and time.
w	Displays the archive symbol table. Each symbol is listed with the name of the file in which the symbol is defined.
x	Extracts the named files by copying them into the current directory. These copies have the same name as the original files, which remain in the library. If you do not specify any files, **x** copies all files out of the library. This process does not alter the library.

Examples

1. To create a library:

   ```
   ar  vq  lib.a  strlen.o  strcpy.o
   ```

   If `lib.a` does not exist, then this creates it and enters into it copies of the files `strlen.o` and `strcpy.o`. If `lib.a` does exist, then this adds the new members to the end without checking for duplicate members. The **v** flag sets verbose mode, in which **ar** displays progress reports as it proceeds.

2. To list the table of contents of a library:

   ```
   ar  vt  lib.a
   ```

   This lists the table of contents of `lib.a`, displaying a long listing similar to **ls -l**. To list only the member file names, omit the **v** flag.

3. To replace or add new members to a library:

   ```
   ar  vr  lib.a  strlen.o  strcat.o
   ```

   This replaces the members `strlen.o` and `strcat.o`. If `lib.a` was created as shown in Example 1, then the `strlen.o` member is replaced. A member named `strcat.o` does not already exist, so it is added to the end of the library.

4. To specify where to insert a new member:

   ```
   ar vrb strlen.o lib.a strcmp.o
   ```

   This adds `strcmp.o`, placing the new member before `strlen.o`.

5. To update a member if it has been changed:

   ```
   ar  vru  lib.a  strcpy.o
   ```

This replaces the existing strcpy.o member, but only if the file strcpy.o has been modified since it was last added to the library.

6. To change the order of the library members:

```
ar  vma  strcmp.o  lib.a  strcat.o  strcpy.o
```

   This moves the members strcat.o and strcpy.o to positions immediately after strcmp.o. The relative order of strcat.o and strcpy.o is preserved. In other words, if strcpy.o preceded strcat.o before the move, then it still does.

7. To extract library members:

```
ar  vx  lib.a  strcat.o  strcpy.o
```

   This copies the members strcat.o and strcpy.o into individual files named strcat.o and strcpy.o, respectively.

8. To extract and rename a member:

```
ar  p  lib.a  strcpy.o  >stringcopy.o
```

   This copies the member strcpy.o to a file named stringcopy.o.

9. To delete a member:

```
ar  vd  lib.a  strlen.o
```

   This deletes the member strlen.o from the library lib.a.

Files

/tmp/ar*	Temporary files.

Related Information

The following commands: "**backup**" on page 88, "**ld**" on page 557, "**lorder**" on page 591, "**make**" on page 625, "**nm**" on page 705, "**size**" on page 949, and "**strip**" on page 1017.

The **a.out** and **ar** files and **environment** miscellaneous facility in *AIX Operating System Technical Reference.*

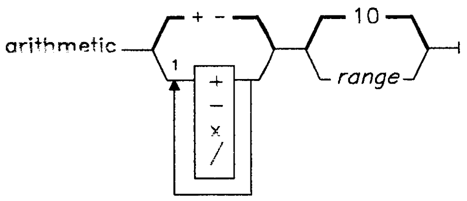
The "Overview of International Character Support" in *Managing the AIX Operating System.*

arithmetic

Purpose

Tests arithmetic skills.

Syntax



OL805164

¹ Do not put a blank between these items.

OL805308

Description

The **arithmetic** command displays simple arithmetic problems and waits for you to enter an answer. If your answer is correct, the program displays `Right!` and presents a new problem. If your answer is wrong, it displays `What?` and waits for another answer. Every 20 problems, **arithmetic** displays the number of correct and incorrect responses and the time required to answer.

The **arithmetic** command does not give the correct answers to the problems it displays. It provides practice rather than instruction in performing arithmetic calculations.

The *range* is a decimal number specifying the permissible range of all numbers (except answers). The default range is 10. At the start, all numbers within this range are equally likely to appear. If you make a mistake, the numbers in the problem you missed become more likely to reappear.

To quit the game, press **INTERRUPT** **(Alt-Pause)**; **arithmetic** displays the final game statistics and exits.

Flags

Two types of optional flags modify the action of **arithmetic**. The first set specifies the type of arithmetic problem:

+	Specifies addition problems.
-	Specifies subtraction problems.
x	Specifies multiplication problems.
/	Specifies division problems.

If you do not select any flags, **arithmetic** selects addition and subtraction problems. If you give more than one problem specifier (+**-x**/), the program mixes the specified types of problems in random order.

Examples

1. To drill on addition and subtraction of integers from 0 to 10:

```
/usr/games/arithmetic
```

2. To drill on addition, multiplication, and division of integers from 0 to 50:

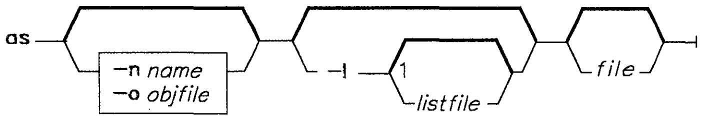
```
/usr/games/arithmetic  +x/  50
```

as

Purpose

Assembles a source file.

Syntax



OL805165

¹ Do not put a blank between these items.

OL805308

Description

The **as** command reads and assembles the named *file* (conventionally this file ends with a **.s** suffix). If you do not specify a *file*, **as** reads and assembles standard input. It stores its output, by default, in a file named **a.out**. The output file is executable if no errors occur and if there are no unresolved external references.

Flags

-l[*listfile*]	Produces an assembler listing. If you do not specify a file name, a default name is produced by replacing the **.s** extension of the source file name with an **.lst** extension.
-n *name*	Specifies the name that appears in the header of the assembler listing. By default, the header contains the name of the assembler source file.
-o *objfile*	Writes the output of the assembly process to the specified file instead of to **a.out**.

Files

a.out	Default output file.

Related Information

The following commands: "**cc**" on page 140 and "**ld**" on page 557.

The **a.out** file in *AIX Operating System Technical Reference.*

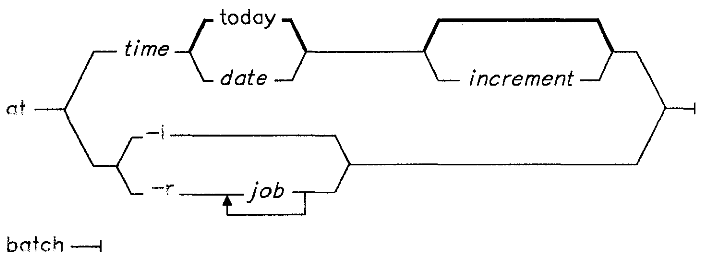
The discussion of **as** in *Assembler Language Reference* and *AIX Operating System Programming Tools and Interfaces.*

at, batch

Purpose

Runs commands at a later time.

Syntax



OL805002

Description

The **at** and **batch** commands read from standard input the names of commands to be run at a later time:

- **at** allows you to specify when the commands should be run.
- **batch** runs jobs when the system load level permits.

Both **at** and **batch** mail you all output from standard output and standard error for the scheduled commands, unless you redirect that output. They also write the job number and the scheduled time to standard error.

Variables in the shell environment, the current directory, **umask**, and **ulimit** are retained when the commands are run. Open file descriptors, traps, and priority are lost.

You can use **at** if your name appears in the file **/usr/lib/cron/at.allow**. If that file does not exist, **at** checks the file **/usr/lib/cron/at.deny** to determine if you should be denied access to **at**. If neither file exists, only the superuser can submit a job. The **allow/deny** files contain one user name per line. If **at.allow** does exist, the superuser's login name must be included in it for the superuser to be able to use the command.

The required *time* parameter can be one of the following:

1. A number followed by an optional suffix. **at** interprets one- and two-digit numbers as hours. It interprets four digits as hours and minutes. The **NLTIME** environment variable specifies the order of hours and minutes. The default order is the hour followed by the minute. You can also separate hours and minutes with a : (colon). The default order is *hour:minute.*

   In addition, you may specify a suffix of **am**, **pm**, or **zulu**. If you do not specify **am** or **pm**, **at** uses a 24 hour clock. The suffix **zulu** indicates that the time is **GMT** (Greenwich Mean Time). The **NLTMISC** environment variable controls the suffixes that **at** recognizes.

2. **at** also recognizes the following keywords as special *time*s: **noon**, **midnight**, and **now**. Note that you can use the special word **now** only if you also specify a *date* or an *increment*. Otherwise, **at** tells you: `too late`. The **NLTSTRS** environment variable controls the additional keywords that **at** recognizes.

You may specify the *date* parameter as either a month name and a day number (and possibly a year number preceded by a comma), or a day of the week. The **NLDATE** environment variable specifies the order of the month name and day number (by default, month followed by day). The **NLLDAY** environment variable specifies long day names; **NLSDAY** and **NLSMONTH** specify short day and month names. (By default, the long name is fully spelled out; the short name abbreviated to three characters.) **at** recognizes two special "days," **today** and **tomorrow** by default. (The **NLTSTRS** environment variable specifies these special days.) **today** is the default *date* if the specified time is later than the current hour; **tomorrow** is the default if the time is earlier than the current hour. If the specified month is less than the current month (and a year is not given), next year is the default year. The optional *increment* can be one of the following:

1. A + (plus sign) followed by a number and one of the following words: **minute**[s], **hour**[s], **day**[s], **week**[s], **month**[s], **year**[s] (or their non-English equivalents).
2. The special word **next** followed by one of the following words: **minute**[s], **hour**[s], **day**[s], **week**[s], **month**[s], **year**[s] (or their non-English equivalents).

The **NLTUNITS** environment variable specifies the non-English equivalents of the English defaults.

Flags

-l	Reports your scheduled jobs.
-r *job* . . .	Removes *job*s previously scheduled by **at** or **batch**, where *job* is the number assigned by **at** or **batch**. If you do not have superuser authority (see "**su**" on page 1026), you can remove only your own jobs.

Examples

1. To schedule the command from the terminal, use a command similar to one of the following:

   ```
   at  5 pm  Friday uuclean
   Ctrl-D
   at  now  next  week uuclean
   Ctrl-D
   at  now  +  2  days uuclean
   Ctrl-D
   ```

2. To run **uuclean** at 3:00 in the afternoon on the 24th of January, use any one of the following commands:

   ```
   echo  uuclean  |  at  3:00  pm  January  24
   echo  uuclean  |  at  3pm  Jan  24
   echo  uuclean  |  at  1500  jan  24
   ```

3. To run a job when the system load permits:

   ```
   batch  <<!
   longjob  2>&1  >outfile  |  mail  myID
   !
   ```

   This example shows the use of a ***here document*** to send standard input to **at** (see "Inline Input Documents" on page 928).

   The order of redirections is important here, so that only error messages are sent into the pipe to the **mail** command. If you reverse the order, both standard error and standard output are sent to `outfile` (see the discussion of "Input and Output Redirection Using File Descriptors" on page 928 for details).

4. To have a job reschedule itself, invoke **at** from within the shell procedure by including code similar to the following within the shell file:

   ```
   echo  "sh  shellfile"  |  at  now  tomorrow
   ```

5. To list the jobs you have sent to be run later:

   ```
   at  -l
   ```

6. To cancel jobs:

   ```
   at  -r  103  227
   ```

   This cancels jobs `103` and `227`. Use **at -l** to list the job numbers assigned to your jobs.

Files

/usr/lib/cron	Main cron directory.
/usr/lib/cron/at.allow	List of allowed users.
/usr/lib/cron/at.deny	List of denied users.
/usr/spool/cron/atjobs	Spool area.

Related Information

The following commands: "**cron**" on page 220, "**kill**" on page 552, "**mail, Mail**" on page 608, "**nice**" on page 699, "**ps**" on page 786, and "**sh**" on page 913.

The **environment** special facility in *AIX Operating System Technical Reference.*

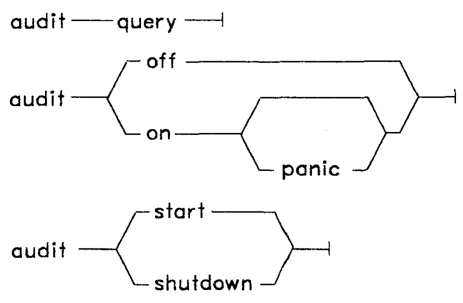
"Running Commands at Pre-set Times" and "Overview of International Character Support" in *IBM RT Managing the AIX Operating System.*

audit

Purpose

Controls system auditing

Syntax



AJ2FL131

Description

The **audit** command controls system auditing. The **audit** command enables or disables auditing, and no audit records are generated if the audit system is disabled. You must have superuser authority to run this command.

The following arguments are available with the **audit** command:

query Gives the current status of the auditing system in the form:

```
auditing on             (or auditing off)
bin processing off      (or bin manager is process number)
audit events:
           audit class:  auditevent,auditevent,auditevent
           (or none)
```

start Sets up and enables the auditing system. The system initialization file, **/etc/rc**, normally includes the **audit start** command.

shutdown Terminates the operation of the auditing system. This argument forces all audit records out to the audit trail. It then empties the bin files, invalidates the current configuration, and stops the collection process until the next **audit start** command is given.

off	Stops the auditing system, but leaves the auditing collection configuration valid; no records are lost, and the collection process pauses temporarily until the **audit on** or **audit shutdown** command is given.
on [panic]	Enables auditing. Audit records are generated for enabled events. This argument assumes that Bin audit collection has already been established (see the discussion of information collection in *Managing the AIX Operating System*). **Note:** If you specify the **panic** option, reliable long-term storage of audit records is required. The **auditbin** procedure must already have been started to manage the disposition of audit bins. If the kernel is unable to write a record into a bin for archival, the audit system shuts down the system.

To start the auditing system, **audit** reads configuration information from the **/etc/security/config** file. To start auditing, **audit** does the following:

1. Starts the **auditbin** collection procedure if Bin audit collection is enabled. The procedure synchronously recovers any unprocessed bins.
2. Enables the audit classes defined in the **auditclasses** stanza of the **/etc/security/config** file.
3. Starts auditing.

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Files

/etc/security/audit/events	Lists audit events.
/etc/security/config	Specifies the audit configuration.
/etc/security/audit/cmds	Lists audit bin backend programs.
/etc/security/passwd	Lists audit classes for which users will be audited.

Related Information

The following commands: "**auditbin**" on page 71 and "**auditpr**" on page 73.

The **audit**, **auditbin**, **auditevents**, **auditlog**, and **auditproc** system calls in the *AIX Operating System Technical Reference.*

The **audit**, **events**, **passwd**, and **config** file formats in the *AIX Operating System Technical Reference.*

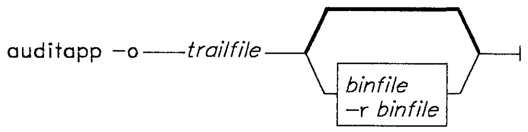
The discussion of accountability in *Managing the AIX Operating System.*

auditapp

Purpose

Adds an audit bin file to the end of the audit trail file.

Syntax



OL805474

Description

The **auditapp** command adds the audit records read from standard input to the audit trail file specified in the **/etc/security/audit/cmds** file. This command is part of the auditing system, which is fully discussed in *Managing the AIX Operating System.*

If you specify a bin file, (*binfile*), then **auditapp** reads from *binfile.*

If *trailfile* does not exist, **auditappend** creates the file.

This command is designed to be used by **auditbin** (a daemon) and should not be used on the command line. The **auditapp** command expects to find a complete bin (has both header and trailer portions). Entering **audit shutdown** completes the bin for processing by **auditapp**. Error conditions occur if the **auditapp** command is executed when bins are not properly completed with header and trailer portions.

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Flags

-o	Specifies the the audit trail to which **auditapp** appends records. You must specify this flag.

-r	Recovers bin files before processing them. When you specify **-r** during a recovery procedure, **auditapp** will recover and process any unprocessed audit records. If you specify the **-r** flag, you must also specify *binfile*; however, you can specify *binfile* without the **-r** flag.

Files

etc/security/audit/cmds	Contains audit bin backend programs.

Related Information

The following commands: "**audit**" on page 67, "**auditbin**" on page 71, and "**auditselect**" on page 76.

The discussion of the audit trail in *Managing the AIX Operating System.*

auditbin

Purpose

Manages bins of audit information.

Syntax

```
auditbin ──┤
```

OL805475

Description

The **auditbin** command (a daemon) delivers ***bins*** of audit records to audit ***backends***. A bin is a file for storing audit information prior to processing. A backend is a program that sends its output, in this case the processed audit records, to a particular device or file. This device may then provide long-term storage. The default backend command is **auditapp** (see "**auditapp**" on page 69). This command is part of the auditing system, which is fully discussed in *Managing the AIX Operating System.*

Each audit backend is a command listed in **/etc/security/audit/cmds**. When **auditbin** receives a bin from the kernel, **auditbin** invokes each command in the **/etc/security/audit/cmds** file to process the bin. The **auditbin** command searches each command line for the keyword **$bin** and replaces it with the path name of the bin file.

If a backend command fails, **auditbin** stops processing the bins. It sends a message to **/dev/console** alerting the user of the problem and indicating that the command be terminated. The message repeats every 60 seconds until the command is terminated.

The **auditbin** command assures that each backend encounters each bin at least once. In the case of multiple commands, the **auditbin** command does not guarantee that each audit backend command will complete before the next one begins. Synchronization depends on the individual commands. Each backend command must wait for any duplicate command to complete.

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Files

/etc/security/audit/cmds	Lists audit backend commands.
/etc/security/config	Specifies the audit configuration.

Related Information

The following commands: "**audit**" on page 67 and "**auditpr**" on page 73.

The **audit**, **auditevents**, **auditlog** and **auditproc** system calls in *AIX Operating System Technical Reference.*

The discussion of the audit trail in *Managing the AIX Operating System.*

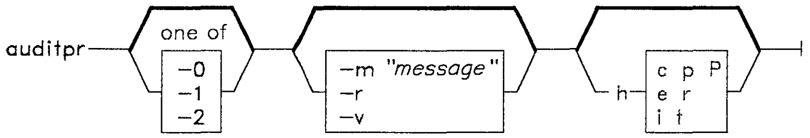
The following file format: **config** in *AIX Operating System Technical Reference.*

auditpr

Purpose

Displays audit trail files.

Syntax



A5AC5015

Description

The **auditpr** command reads kernel audit records from standard input and sends formatted records to standard output. This command is part of the auditing system, which is fully discussed in *Managing the AIX Operating System.*

By default **auditpr** searches the local **/etc/passwd** file to convert user and group IDs to names.

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Flags

The first series of flags, **-0**, **-1**, and **-2**, specify how often to print a title.

-0	Never print a title.
-1	This flag is the default. It specifies that a title be printed only once.
-2	This flag specifies that a title be printed before each record.
-m "*message*"	Displays *message* before each output record.
-r	Displays numeric user IDs.

-v

Displays the tail of each audit record.

If the **-v** option is selected, **auditpr** prints the tail of each audit record in the format specified. The information in the tail is specific to the event that the record signifies. To print the tail of a record, **auditpr** searches the **auditpr** stanza of **/etc/security/audit/events** for an attribute name (audit event).

Note: The audit event is the attribute name. The attribute value has the form:

event = *path*[,"*arguments*"]

Where *path* specifies a command to be executed to print the tail of the record. Invoke this command as:

program arguments

The tail of the audit record is written to the program's standard input, and a formatted version is written to the program's standard output.

If an attribute is not found, **auditpr** will print as the tail the warning: `unknown event.`

-h *field*

Displays header fields specified by *field.* The **-h** flag specifies the header fields to be printed. Field names and their widths are:

ID	Field	Width	Description
e	event	17	Audit event name.
c	command	17	Command name.
l	luid	6	User's login ID.
r	ruid	6	Process real user ID.
u	euid	6	Process effective user ID.
p	pid	6	Process ID.
P	ppid	6	Process ID of parent.
R	result	2	Result code of the action.
t	time	26	Time at which record was written.

The default header format is the combination **eclt**. The records that result from this default format appear as follows:

```
event      command luid   time
-----      ------- ----   ----
login      login   dick   Fri Feb 8, 1988 14:03:57
            . . .  tail portion, if requested  . . .
users      adduser jane   Fri Feb 8, 1988 14:04:33
            . . .  tail portion, if requested  . . .
```

For system calls, the tail portion consists of:

1. The arguments to the system call
2. A list of path names, each followed by two digits:
   - Nonzero - indicates that the path name is a symbolic link to the next path name. A zero (0) indicates a non-symbolic link file.
   - A return code by the kernel after trying to access the path name.

For items with the **printf** specification, the tail consists of the string of information specified in the quoted string that follows the specification.

File

/etc/security/audit/events Lists audit events.

Related Information

The following commands: "**audit**" on page 67 and "**auditselect**" on page 76.

The following system call: **audit** in *AIX Operating System Technical Reference.*

The following file formats: **attributes**, **events**, and **config** in *AIX Operating System Technical Reference.*

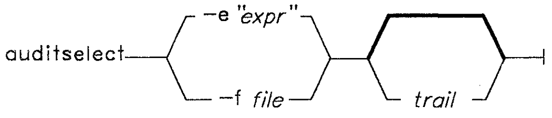
The discussions of hard copy labeling and the printer subsystem in *Managing the AIX Operating System.*

auditselect

Purpose

Selects audit records.

Syntax



OL805476

Description

The **auditselect** command reads audit records from standard input and writes records to standard output. This command is part of the auditing system, which is fully discussed in *Managing the AIX Operating System.*

If *trail* is specified, **auditselect** extracts records from audit *trail* and writes selected records to standard output.

The *file* is a file containing an expression.

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Flags

-e "*expr*" Specifies an expression, *expr*, which consists of terms in the following form:

field relop value

These terms are defined as:

field One of the following:

- **event**
- **command**
- **login**
- **real**

- **effective**
- **pid**
- **ppid**
- **time**
- **prepend**

relop One of the following relational operation signs: ==(equal equal), !=(exclamation point equal), <(less than), >(greater than),. >=(greater than equal), or <=(less than equal).

value A quoted string if the event or command field was specified; a time in the format specified by the NLTIME environment variable; a date in the format specified by the NLDATE environment variable; or an integer if one of the following fields was specified: **pip, ppid, login, real**, and **effective**.

Combine these terms using the logical operators &&, (*and*) | |, (*or*) and ! (*not*). Use **()** (parentheses) to force the order of evaluation. Otherwise, normal precedence rules apply.

-f *file* Specifies a file containing an expression.

Related Information

The following commands: "**auditbin**" on page 71 and "**auditpr**" on page 73.

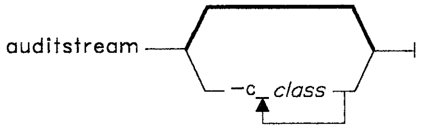
The **NLtmtime** subroutine in *AIX Operating System Technical Reference.*

auditstream

Purpose

Creates a channel for the reading of audit records.

Syntax



OL805477

Description

The **auditstream** command creates a channel to the audit device, **/dev/audit**. Audit records are read from **/dev/audit** by means of this channel and copied to standard output. The **auditstream** command can be used as the first command in an audit stream pipeline.

This command is part of the auditing system, which is fully discussed in *Managing the AIX Operating System.*

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Flags

-c*class*	Specifies audit classes as found in **etc/security/config**. Each audit record that belongs to an audit class specified by a **-c** option is read through the channel created by **auditstream**. If no audit classes are specified by a **-c** option in the **/etc/security/config** file, all currently enabled audit events are read through this channel.

Files

/etc/security/config	Specifies the audit configuration.
/dev/audit	The audit device.

Related Information

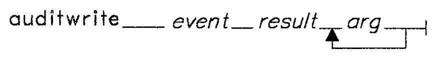
The discussion of the auditing subsystem in *Managing the AIX Operating System.*

auditwrite

Purpose

Generates an audit record at the command level.

Syntax



OL805478

Description

The **auditwrite** command combines an *event*, its *result*, and any *arguments* of supplied strings of data.

The *event* is the audit event to be audited (audit events can be found in the **/etc/security/audit/events** file), and the *result* is an indicator of the outcome of the *event*. The *arg* includes the audit information pertaining to the *event*.

This command is part of the auditing system, which is fully discussed in *Managing the AIX Operating System*. You must be a superuser to use this command.

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Files

/etc/security/audit/events	Lists audit events.

Related Information

The following commands: "**audit**" on page 67, "**auditbin**" on page 71, "**auditpr**" on page 73, and "**auditselect**" on page 76.

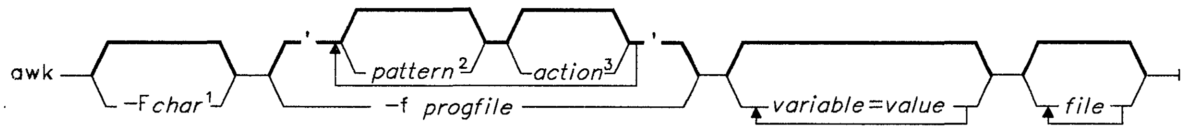
The discussion of auditing in *Managing the AIX Operating System*.

awk

Purpose

Finds lines in files matching specified patterns and performs specified actions on them.

Syntax



[1] The default *char* is a tab.
[2] The default pattern is every line.
[3] The default action is to print the line.

OL805422

Description

The **awk** command is a more powerful pattern matching command than the **grep** command. It can perform limited processing on the input lines, instead of simply displaying lines that match. Some of the features of **awk** are:

- It can perform convenient numeric processing.
- It allows variables within actions.
- It allows general selection of patterns.
- It allows control flow in the actions.
- It does not require any compiling of programs.

For a detailed discussion of **awk**, see *AIX Operating System Programming Tools and Interfaces.*

The **awk** command, reads *file*s in the order stated on the command line. If you specify a file name as - (minus) or do not specify a file name, **awk** reads standard input.

The **awk** command searches its input line by line for *pattern*s. When it finds a match, it performs the associated *action* and writes the result to standard output. Enclose *pattern-action* statements on the command line in single quotation marks to protect them from interpretation by the shell.

The **awk** command first reads all pattern-action statements, then it reads a line of input and compares it to each pattern, performing the associated actions on each match. When it has compared all patterns to the input line, it reads the next line.

The **awk** command treats input lines as fields separated by spaces, tabs, or a field separator you set with the **FS** variable. Fields are referenced as **$1**, **$2**, and so on. **$0** refers to the entire line.

On the **awk** command line, you can assign *values* to variables as follows:

variable=value

Pattern-Matching Statements

Pattern-matching statements follow the form:

pattern { *action* }

If a *pattern* lacks a corresponding *action*, **awk** writes the entire line that contains the pattern to standard output. If an *action* lacks a corresponding *pattern*, it matches every line.

Actions

An action is a sequence of statements that follow C Language syntax. These statements can include:

statement	***format***
if	**if** (*conditional*) *statement* [**else** *statement*]
while	**while** (*conditional*) *statement*
for	**for** (*expression* ; *conditional* ; *expression*) *statement*
break	
continue	
{ *statement* . . . }	
(*assignment*)	*variable=expression*
print	**print** [*expression-list*] [>*expression*]
printf	**printf** *format*[, *expression-list*] [>*expression*]
next	
exit	

Statements can end with a semicolon, a new-line character , or the right brace enclosing the action.

If you do not supply an action, **awk** displays the whole line. Expressions can have string or numeric values and are built using the operators +, -, *, /, %, a blank for string concatenation, and the C operators ++, --, +=, -=, *=, /=, and %=.

Variables can be scalars, array elements (denoted x[i]) or fields. Variable names can consist of upper- and lower-case alphabetic letters, the underscore character, the digits (0-9), and SJIS characters.

Japanese Language Support Information

Variable names can also include kanji characters.

Variable names cannot begin with a digit. Variables are initialized to the null string. Array subscripts can be any string; they do not have to be numeric. This allows for a form of associative memory. String constants in expressions should be enclosed in double quotation marks.

There are several variables with special meaning to **awk**. They include:

FS	Input field separator (default is a blank). This separator character cannot be a 2-byte extended character.
NF	The number of fields in the current input line (record).
NR	The number of the current input line (record).
FILENAME	The name of the current input file.
OFS	The output field separator (default is a blank). This separator character cannot be a 2-byte extended character.
ORS	The output record separator (default is a new-line character). This separator character cannot be a 2-byte extended character.
OFMT	The output format for numbers (default `%.6g`).

Since the actions process fields, input white space is not preserved on the output.

The **printf** expression list formats like the **printf** subroutine (see *AIX Operating System Technical Reference*). It writes arguments to standard output, separated by the output field separator and terminated by the output record separator. You can redirect the output using the **print** > *file* or **printf** > *file* statements.

Note: You must enclose the file name in double quotes when redirecting output with the **awk** command.

You have two ways to designate a character other than white space to separate fields. You can use the **-F***c* flag on the **awk** command line, or you can start *progfile* with:

```
BEGIN { FS = c }
```

Either action changes the field separator to *c*.

There are several built-in functions that can be used in **awk** actions.

length [(*arg***)]**	Returns the length in characters of the whole line if there is no argument or the length of its argument taken as a string.
blength [(*arg***)]**	Returns the length in bytes of the whole line if there is no argument or the length of its argument taken as a string.
exp(*n***)**	Takes the exponential of its argument.

log(*n***)**	Takes the base e logarithm of its argument.
sqrt(*n***)**	Takes the square root of its argument.
int(*n***)**	Takes the integer part of its argument.
substr(*s*,*m*,*n***)**	Returns the substring *n* characters long of *s*, beginning at position *m*.
sprintf(*fmt*,*expr*,*expr*, . . . **)**	Formats the expressions according to the **printf** format string *fmt* and returns the resulting string.

Patterns

Patterns are arbitrary Boolean combinations of patterns and relational expressions (the !, ||, and && operators and parentheses for grouping). You must start and end patterns with slashes (/). You can use regular expressions like those allowed by the **egrep** command (see "**grep**" on page 501), including the following special characters:

+	One or more occurrences of the pattern.
?	Zero or one occurrences of the pattern.
|	Either of two statements.
( )	Grouping of expressions.

Isolated patterns in a pattern apply to the entire line. Patterns can occur in relational expressions. If two patterns are separated by a comma, the action is performed on all lines between an occurrence of the first pattern and the next occurrence of the second.

Regular expressions can contain extended characters with one exception: range constructs in character class specifications using square brackets cannot contain 2-byte extended characters. Individual instances of extended characters can appear within square brackets; however, 2-byte extended characters are treated as two separate 1-byte characters.

Japanese Language Support Information

Regular expressions can contain kanji characters. In that case, range constructs in character class specifications using square brackets can contain 2-byte kanji characters, which are treated as 2-byte characters.

Regular expressions can also occur in relational expressions. There are two types of relational expressions that you can use. One has the form:

expression matchop pattern

where *matchop* is either: ~ (for "contains") or !~ (for "does not contain"). The second has the form:

expression relop expression

where *relop* is any of the six C relational operators: <, >, <=, >=, ==, and !=. A conditional can be an arithmetic expression, a relational expression, or a Boolean combination of these.

You can use the special patterns **BEGIN** and **END** to capture control before the first and after the last input line is read, respectively. You can only use these patterns before the first and after the last line in *progfile.*

There are no explicit conversions between numbers and strings. To force an expression to be treated as a number, place a 0 at the beginning of the expression. However, note that only ASCII digits are treated as numeric. To force a regular expression to be treated as a string, append a null string ("").

Flags

-f *progfile*	Searches for the patterns and performs the actions found in the file *progfile.*
-F*char*	Uses *char* as the field separator character (by default a blank).

Examples

1. To display the lines of a file that are longer than 72 characters:

   ```
   awk  "length  >72"  chapter1
   ```

   This selects each line of the file `chapter1` that is longer than 72 characters. **awk** then writes these lines to standard output because no *action* is specified.

2. To display all lines between the words `start` and `stop`:

   ```
   awk  "/start/,/stop/"  chapter1
   ```

3. To run an **awk** program (`sum2.awk` .) that processes a file (`chapter1`):

   ```
   awk  -f  sum2.awk  chapter1
   ```

   The following **awk** program computes the sum and average of the numbers in the second column of the input file:

   ```
       {
          sum += $2
       }

   END {
          print "Sum: ", sum;
          print "Average:", sum/NR;
       }
   ```

The first action adds the value of the second field of each line to the variable `sum`. **awk** initializes `sum` (and all variables) to zero before starting. The keyword **END** before the second action causes **awk** to perform that action after all of the input file has been read. The variable **NR**, which is used to calculate the average, is a special variable containing the number of records (lines) that have been read.

4. To print the names of the users who have the C shell as the initial shell:

```
awk  -F:  '/csh/{print  $1}'  /etc/passwd
```

Related Information

The following commands: "**lex**" on page 562, "**grep**" on page 501, and "**sed**" on page 887.

The **printf** subroutine in *AIX Operating System Technical Reference.*

The "Overview of International Character Support" in *Managing the AIX Operating System.*

The discussion of **awk** in *AIX Operating System Programming Tools and Interfaces.*

The discussion of Japanese Language Support in *Japanese Language Support User's Guide.*

back

Purpose

Plays backgammon.

Syntax

```
/usr/games/back ─┤
```

OL805186

Description

The **back** game provides you with a partner for backgammon. You select one of three skill levels: beginner, intermediate, or expert. You may also choose to roll your own dice during your turns, and you are asked if you want to move first.

The points are numbered such that:

- 0 is the bar for removed white pieces.
- 1 is white's extreme inner table.
- 24 is brown's extreme inner table.
- 25 is the bar for removed brown pieces.

For details on how to make your moves, enter `y` when **back** asks `Instructions` at the beginning of the game. When it first asks `Move?`, enter ? to see a list of choices other than entering a numerical move.

When the game is finished, **back** asks you if you want to save game information. A `y` response stores game data in the file **back.log** in your current directory.

The **back** game plays only the forward game, even at the expert level. It will object if you try to make too many moves in a turn, but not if you make too few. Doubling is not implemented.

To quit the game, press INTERRUPT (**Alt-Pause**).

Files

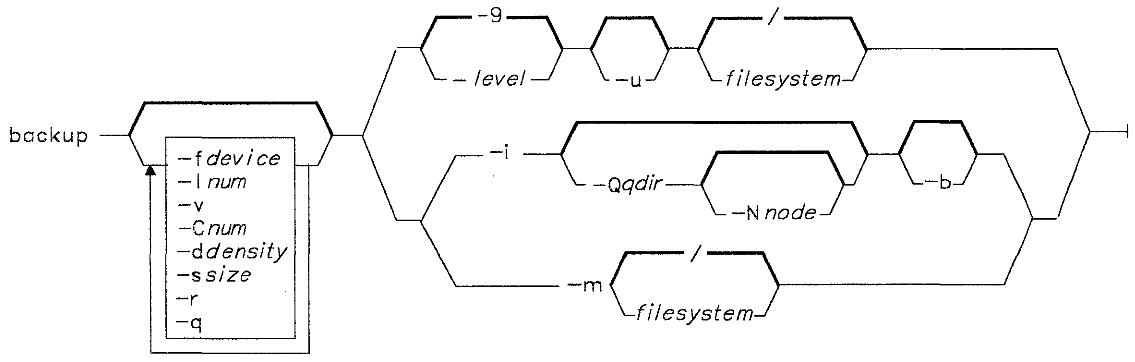
/usr/games/lib/backrules	Rules file.
/tmp/b*	Log temp file.
back.log	Log file.

backup

Purpose

Backs up files.

Syntax



OL805082

Description

The **backup** command copies files in **backup** format to a backup medium, such as a magnetic tape or diskette.

There are three ways to back up data:

- To back up specified files (***backup by name***) **-i**
- To back up an entire file system (***backup by file system or i-node***) *-level*
- To back up an entire minidisk (***backup by minidisk***) **-m**

To back up by name, use the **-i** flag. The **backup** command reads standard input for the names of the files to be backed up. You can specify files by using the **find** command to generate a list of path names and pipe the list into the **backup** command.

Backing up by name allows you to back up files to a backup medium on the local system or on a remote system.

When you specify **-Q**, **-Q** and **-N**, or when you use the **print -backup** command, the system writes a backup header to the backup medium. A header can contain the name of a qualifying directory and a target directory that subsequent **restore** commands can use to restore the files to the proper place. When a backup header is written if **-Q** is not specified, the system writes the path of the backup process's current directory to the header; if **-N** is not specified, the system writes the id of the node that requested the backup to the header.

To back up by file system (i-node), specify *-level* and *filesystem* to indicate the files you want to back up. You can use the level to back up either all files on the system (a ***full backup***) or only the files that have been modified since a specific full backup (an ***incremental backup***). The possible levels are 0-9. If you do not supply a level, the default level is 9. A level 0 backup includes all files on the file system. A level *n* backup includes all files modified since the last level *n*-1 backup. The levels, in conjunction with the **-u** flag, provide an easy way to maintain a hierarchy of incremental backups for each file system. For a discussion of backup strategy and the use of incremental backups, see *Managing the AIX Operating System.*

If you specify the name of a *filesystem,* it can be either the physical device name (the block or raw name) or the name of the directory on which the file system is normally mounted. When you specify a directory, **backup** reads **/etc/filesystems** for the physical device name. In this case, it also acquires values for other backup parameters from **/etc/filesystems**. If you do not specify a file system, the default is the root file system on the current minidisk.

To back up by minidisk, use the **-m** flag. This option copies an exact image of the entire minidisk. You can specify the file system name of the minidisk. The default is the root directory of the current minidisk. Because a backup by minidisk backs up an entire minidisk as an exact image, a large minidisk with a small or sparsely used file system may take longer and require more backup medium to back up this way, rather than by file system or by name.

When you do not specify a backup device, the **backup** command writes files to a default backup device. For backup by name, **backup -i**, the system writes to **/dev/rfd0** unless you specify a device with the **-f** flag. For a backup by file system (i-node), **backup** *-level*, or a backup by minidisk, **backup -m**, if **/etc/filesystems** contains a stanza that matches the name you specified and a stanza with a **backupdev** entry, then the system writes to the device specified by **backupdev**. Otherwise, the system writes to **/dev/rmt0** or the device specified with the **-f** flag.

The **backup** command recognizes a special syntax for the names of output files. If the argument is a range of names, such as **/dev/rfd0-3**, the **backup** command automatically goes from one drive in the range to the next. After exhausting all of the specified drives, it halts and requests that new volumes be mounted.

Notes:

1. During execution of remote backup operations, the file system is unmounted. The file system is remounted after the backup completes.
2. If you back up by either file system (i-node) or minidisk, the backup source and target must be on the local system. To back up to a remote system, back up by name with the **-i** flag. This flag allows users in a distributed services environment to back up files on a remote file system.
3. You should use the **-u** flag when you do an incremental backup to ensure that information regarding the last date, time, and level of each incremental backup is written to the file **/etc/budate**.
4. If the file system you are backing up is mounted and is not the root file system, **backup** unmounts the file system before it performs a file system (i-node) or minidisk backup and then remounts the file system before quitting. If the file systems you are backing up include the root file system, **backup** ensures that the other file systems are not in use. If one is, it warns you of this use and quits.

Warning: Be sure that the flags you specify match the backup medium. If the backup medium is not a disk or diskette, do not specify the **-l** flag. Similarly, if the backup medium is not a tape, do not specify the **-d** or **-s** flags. If you do specify flags that do not go with the medium, **backup** displays an appropriate error message and continues the backup.

Flags

-b	Enables users to back up files in unattended mode (user input is not permitted) to a backup medium on a remote system. If any user input (such as `Please insert volume 2`) is required, the command ends in an error. This enables users to set up a shell file that backs up files at night or at other times when the user is unavailable.
-C*num*	Specifies the number of blocks to write in a single output operation. If you do not specify *num*, **backup** uses a default value appropriate for the physical device selected. Larger values of *num* result in longer physical transfers to tape devices. The value of the **-C** flag is always ignored when **backup** writes to diskette. In this case, it always writes in clusters that occupy a complete track.
-d*density*	Specifies the amount of data a system can write to a tape medium in bytes per inch. The default density is 700 bytes per inch. **Note:** Tape drives vary in density capabilities. Use this flag with tape drives other than the IBM 6157 Streaming Tape Drive which has a density of 700.

-f*device*	Specifies the output device. Specify *device* as a file name (such as `/dev/rmt0`) to send output to the named device or specify **-** (minus) to send output to the standard output device. The **-** feature enables you to improve performance when backing up to streaming tape by piping the output of the **backup** command to the **dd** command (see example).
-i	Reads standard input for the names of files to back up.
-l*num*	Uses *num* as the limit of the total number of block to use on a diskette. The default value is the entire diskette (2400 blocks for 1.2M, 720 blocks for 360K diskette, and 2700 for rmt0 6157).
-m	Backs up the entire minidisk as an exact image.
-N *node*	Specifies the target node for subsequent **restore** commands. The *node* can be a node nickname or a node id (nicknames are translated to ids by **backup**). The **backup** command writes the id of node in the backup header. The default is the node id of the node where the **backup** command is running.
-q	Indicates that removable medium is ready to use. When you specify this flag, **backup** proceeds without prompting you to prepare the backup medium or waiting for you to press the **Enter** key to continue. Same as **-r** flag.
-Q *qdir*	Specifies the qualifying directory for subsequent **restore** commands. The **backup** command stores this name in the backup header. Then a subsequent **restore** command can use this information to place files with path names that are relative to a current directory in the qualifying directory. The *qdir* can be a relative or absolute directory. The default is the backup process's current working directory.
-r	Indicates that removable medium is ready to use. When you specify this flag, **backup** proceeds without prompting you to prepare the backup medium or waiting for you to press the **Enter** key to continue. Same as **-q** flag.
-s*length*	Specifies the *length* in feet of usable space on a tape medium. This is a combination of the physical length and the number of tracks on the tape. In the case of IBM RT Streaming Tape, you should multiply the physical length of the tape by 9 (the number of tracks) to determine the usable space available.
-u	Updates the time, date, and level of the backup in the **/etc/budate** file. This file provides the information needed for incremental backups.
-v	Reports on each phase of the backup as it is completed and gives regular progress reports during the longest phase.
-level	Specifies the backup *level* (0-9). The default *level* is 9.

Examples

1. To back up selected files:

   ```
   find  $HOME  -print  |  backup  -i  -v
   ```

   The -i flag tells the system to read from standard input the names of files to be backed up. The **find** command generates a list of files in the user's $HOME directory. This list is piped to the **backup** command as standard input. The -v displays a progress report as each file is copied. The files are backed up on the default backup device for the local system.

2. To back up an entire file system:

   ```
   backup  -0  -u  /
   ```

   The -0 level and the / file system tell the system to back up the entire root file system. The file system is backed up to the default device defined in the **backupdev** entry in **/etc/filesystems** if it exits. Otherwise, the files are backed up to **/dev/rfd0**. The **-u** tells the system to update the current backup level record in **/etc/budate**. Only the root file system is backed up, not mounted file systems.

3. To back up all files modified since the last level 0 backup:

   ```
   backup  -1  -u  /
   ```

4. To back up an entire minidisk:

   ```
   backup  -mf/dev/rmt1  /xyz
   ```

   This backs up the entire minidisk that contains the file system xyz. The -f tells the system to back up the minidisk to the streaming tape on /dev/rmt1 instead of the default device.

5. To back up files by name to the remote default device and specify the qualifying directory:

   ```
   find filelist -print | backup  -i -Q /tmp/darlene
   ```

   The system backs up the files in filelist and writes the qualifying directory /tmp/darlene to the header. Since a target node is not specified, the default node (the node where the **backup** command is running) is written to the header.

6. To back up files to a remote device and specify both the target node and the qualifying directory:

   ```
   find . -print | backup  -i -N darlene -Q /tmp/darlene
   ```

   This command backs up the current directory (.). The node nickname darlene is translated to a node id and written to the header with the qualifying directory /tmp/darlene. Note that when -N is specified, the -Q flag must also be present.

7. To improve performance on streaming tape, pipe the **backup** command to the **dd** command:

   ```
   backup -if- -C30 | dd of=/dev/rmt0 bs=30b
   ```

   The **backup** command backs up by name (`-i`), directs the output to the standard output device (`f-`), and specifies an output size as 30 blocks (`-C30`). The output is piped to **dd**. The **dd** command copies the files to an output file which is a streaming tape device (`of=/dev/rmt0`) and specifies a file size of 30 blocks (`bs=30b`). The file size in both commands should be the same. To restore these files, pipe the **dd** command to **restore**.

Files

/etc/filesystems	Read for default parameters.
/etc/budate	Log for most recent backup dates.
/dev/rfd0	Default backup device.
/dev/rhd0	Default file system.

Related Information

The following commands: "**find**" on page 422, "**dd**" on page 301, and "**restore**" on page 826.

The **budate** and **filesystems** files and the **tape** special file in *AIX Operating System Technical Reference.*

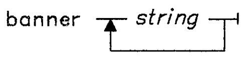
"Backing up Files and File Systems" in *Managing the AIX Operating System.*

banner

Purpose

Writes character strings in large letters to standard output.

Syntax



OL805080

Description

The **banner** command writes character *string*s to standard output in large letters. Each line in the output can be up to 10 uppercase or lowercase characters long. On output, all characters appear in uppercase, with the lowercase input characters appearing smaller than the uppercase input characters.

Examples

1. To display a banner at the work station:

   ```
   banner SMILE!
   ```

2. To display more than one word on a line, enclose the text in quotation marks:

   ```
   banner "Out to" Lunch
   ```

   This displays `Out to` on one line, and `Lunch` on the next.

3. To print a banner:

   ```
   banner We like Computers ¦ print
   ```

Related Information

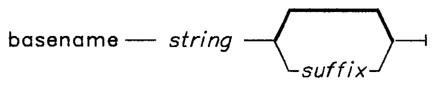
The following command: "**echo**" on page 369.

basename, dirname

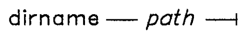
Purpose

Returns the base name of a string parameter.

Syntax



OL805085



OL805047

Description

The **basename** command reads the *string* specified on the command line, deletes any prefix that ends with a / (slash), as well as any specified *suffix*, if it is present, and writes the remaining base file name to standard output.

Note: A **basename** of / is null and is considered an error.

The **dirname** command writes to standard output all but the last part of the specified *path* name (all but the part following the last /).

The **basename** and **dirname** commands are generally used inside ***command substitutions*** within a shell procedure to specify an output file name that is some variation of a specified input file name. For more information, see "Command Substitution" on page 925.

Examples

1. To display the base name of a shell variable:

   ```
   basename  $WORKFILE
   ```

   This displays the base name of the value assigned to the shell variable `WORKFILE`. If `WORKFILE` is set to `/u/jim/program.c`, then `program.c` is displayed.

2. To construct a file name that is the same as another file name, except for its suffix:

   ```
   OFILE=`basename $1 .c`.o
   ```

   This assigns to OFILE the value of the first positional parameter ($1), but with its .c suffix changed to .o. If $1 is /u/jim/program.c, then OFILE becomes program.o. Because program.o is only a base file name, it identifies a file in the current directory.

   The ` ` (grave accents) perform command substitution.

3. To construct the name of a file located in the same directory as another:

   ```
   AOUTFILE=`dirname $TEXTFILE`/a.out
   ```

   This sets the shell variable AOUTFILE to the name of an **a.out** file that is in the same directory as TEXTFILE. If TEXTFILE is /u/fran/prog.c, then the value of dirname $TEXTFILE is /u/fran and AOUTFILE becomes /u/fran/a.out.

Related Information

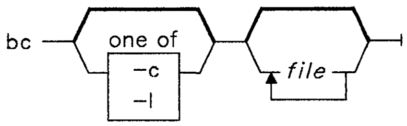
The following command: "**sh**" on page 913.

bc

Purpose

Provides an interpreter for arbitrary-precision arithmetic language.

Syntax



OL805081

Description

The **bc** command is an interactive process that provides unlimited precision arithmetic. It is a preprocessor for the **dc** command. **bc** invokes **dc** automatically, unless the **-c** (compile only) flag is specified. If the **-c** flag is specified, the output from **bc** goes to the standard output.

The **bc** command lets you specify an input and output base in decimal, octal, or hexadecimal (the default is decimal). The command also has a scaling provision for decimal point notation. The syntax for **bc** is similar to that of the C language.

The **bc** command takes input first from the specified *file*. When **bc** reaches the end of the input *file*, it reads standard input.

The following description of syntax for **bc** uses the following abbreviations: *L* means letters a-z; *E* means expressions; *S* means statements.

Names

Simple variables: *L*
Array elements: *L*[*E*]
The words **ibase**, **obase**, and **scale**.
Comments are enclosed in /* and */.

Other Operands

Arbitrarily long numbers with optional sign and decimal point.
(*E*)
sqrt (*E*)
length (*E*)   number of significant decimal digits
scale (*E*)   number of digits to the right of the decimal point
L (*E*, . . . ,*E*)

Operators

```
+ - * / % ^ (% is remainder; ^ is power)
++ -- (prefix and postfix; apply to names)
== <= >= != < >
= =+ =- =* =/ =% =^
```

Statements

```
E
{ S; . . . ;S }
if (E) S
while ( E ) S
for (E;E;E) S
(null statement)
break
quit
```

Function Definitions

```
define L ( L, . . . ,L ) {
    auto L, . . . ,L
    S; . . . S
    return ( E )
}
```

Functions in -l Math Library

s(x)	sine
c(x)	cosine
e(x)	exponential
l(x)	log
a(x)	arctangent
j(n,x)	Bessel function

All function parameters are passed by value.

The value of a statement that is an expression is displayed unless the main operator is an assignment. A semicolon or new-line character separates statements. Assignments to **scale** controls the number of decimal places printed on output and maintained during multiplication, division, and exponentiation. Assignments to **ibase** or **obase** set the input and output number radix respectively.

The same letter may refer to an array, a function, and a simple variable simultaneously. All variables are global to the program. "Auto" variables are pushed down during function calls. When you use arrays as function parameters, or define them as automatic variables, empty square brackets must follow the array name.

All **for** statements must have all three E's.

The **quit** statement is interpreted when read, not when executed.

Flags

-c Compiles *file*, but does not invoke **dc**.

-l Includes a library of math functions.

Examples

1. To use **bc** as a calculator:

```
    You: bc
         1/4
 System: 0
    You: scale = 1  /* Keep 1 decimal place  */
         1/4
 System: 0.2
    You: scale = 3  /* Keep 3 decimal places */
         1/4
 System: 0.250
    You: 16+63/5
 System: 28.600
```

```
    You: (16+63)/5
 System: 15.800
    You: 71/6
 System: 11.833
    You: 1/6
 System: 0.166
```

You may type the comments (enclosed in /* */), but they are provided only for your information. The **bc** command displays the value of each expression when you press the **Enter** key, except for assignments.

When you enter **bc** expressions directly from the keyboard, press **END OF FILE (Ctrl-D)** to end the **bc** session and return to the shell command line.

2. To convert numbers from one base to another:

```
    You: bc
         obase = 16    /* Display numbers in Hexadecimal */
         ibase = 8     /* Input numbers in Octal         */
         12
 System: A
    You: 123
 System: 53
    You: 123456
 System: A72E
```

When you enter **bc** expressions directly from the keyboard, press **END OF FILE (Ctrl-D)** to end the **bc** session and return to the shell command line.

3. To write and run C-like programs:

```
    You: bc -l prog.bc
         e(2)    /*  e squared   */
 System: 7.38905609893065022723
    You: f(5)    /*  5 factorial */
 System: 120
    You: f(10)   /* 10 factorial */
 System: 3628800
```

This interprets the **bc** program saved in `prog.bc`, then reads more **bc** statements from the work station keyboard. Starting **bc** with the **-l** flag makes the math library available. This example uses the **e** (exponential) function from the math library, and `f` is defined in the program file `prog.bc` as:

```
/* compute the factorial of n */

define f(n) {
   auto i, r;

   r = 1;
   for (i=2; i<=n; i++) r =* i;
   return (r);
}
```

The statement following a **for** or **while** statement must begin on the same line. When you enter **bc** expressions directly from the keyboard, press **END OF FILE (Ctrl-D)** to end the **bc** session and return to the shell command line.

4. To convert an infix expression to reverse polish notation (RPN):

```
    You: bc -c
         (a * b) % (3 + 4 * c)
 System: lalb* 3 4lc*+%ps.
```

This compiles the **bc** infix-notation expression into one that the **dc** command can interpret. **dc** evaluates extended RPN expressions. In the compiled output, the **l** (ell) before each variable name is the **dc** subcommand to load the value of the variable onto the stack. The **p** displays the value on top of the stack, and the **s.** discards the top value by storing it in register . (dot). You can save the RPN expression in a file for **dc** to evaluate later by redirecting the standard output of this command. For more details, see "Redirection of Input and Output" on page 926. When you enter **bc** expressions directly from the keyboard, press END OF FILE (**Ctrl-D**) to end the **bc** session and return to the shell command line.

Files

/usr/lib/lib.b	Mathematical library.
/usr/bin/dc	Desk calculator proper.

Related Information

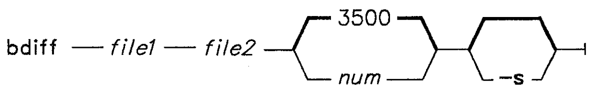
The following command: "**dc**" on page 295.

bdiff

Purpose

Uses **diff** to find differences in very large files.

Syntax



OL805083

Description

The **bdiff** command compares *file1* and *file2* and writes information about their differing lines to standard output. If either file name is - (minus), **bdiff** reads standard input. The **bdiff** command is used like **diff** to find lines that must be changed in two files to make them identical (see "**diff**" on page 320). Its primary purpose is to permit processing of files that are too large for **diff**.

The **bdiff** command ignores lines common to the beginning of both files, splits the remainder of each file into *num*-line segments, and calls **diff** to compare the corresponding segments. In some cases, the 3500 line default for *num* is too large for **diff**. If **diff** fails, specify a smaller value for *num* and try again.

The output of **bdiff** has the same format as that of **diff**. **bdiff** adjusts line numbers to account for the segmenting of the files. Note that because of the file segmenting, **bdiff** does not necessarily find the smallest possible set of file differences.

Flag

-s Suppresses error messages from **bdiff**. (Note that the **-s** flag does not suppress error messages from **diff**).

Example

To display the differences between `chapl` and `chapl.bak`:

```
bdiff  chapl  chapl.bak
```

Files

/tmp/bd*	Temporary files.

Related Information

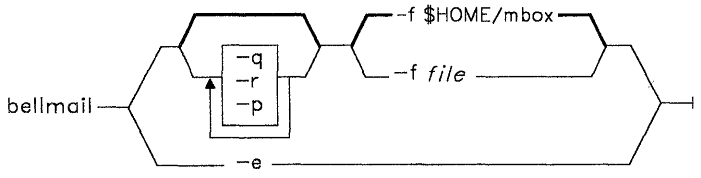
The following command: "**diff**" on page 320.

bellmail

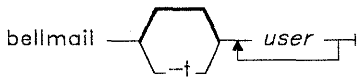
Purpose

Sends messages to system users and displays messages from system users.

Syntax



OL805347



OL805034

Description

The **bellmail** command with no flags writes to standard output, one message at a time, all stored mail addressed to the your login name. Following each message, **bellmail** prompts you with a ? (question mark). Press the **Enter** key to display the next mail message, or enter one of the subcommands that control the disposition of the message (see "Subcommands" on page 106).

When sending mail, you specify *user*s, and then **bellmail** reads a message from standard input until you press **END OF FILE (Ctrl-D)** or enter a line containing only a . (period). It prefixes this message with the sender's name and the date and time of the message (its ***postmark***) and adds this message to the file **/usr/mail/***user* for each *user* specified on the command line.

The action of **bellmail** can be modified in two ways by manipulating **/usr/mail/***user*:

- The default permission assignment for "others" is "read-only." If you change this permission assignment to "read/write" or to "all permissions denied," the system preserves the file, even when it is empty, in order to maintain the desired permissions.
- You can edit the file to contain as its first line:

  `Forward to` *person*

  This causes all messages sent to *user* to be sent to *person* instead. The **Forward to** feature is especially useful for sending all of a person's mail to a particular machine in a network environment.

To specify a recipient on a remote system, prefix the system name and an exclamation mark (!) to *user*. See "**uucp**" on page 1144 for a detailed discussion of how to address remote systems.

Flags

-e	Does not display any messages. This flag causes **bellmail** to return an exit value of 0 if the user has mail, an exit value of 1 if he has no mail.
-f *file*	Saves mail in the named *file* instead of in the default mailfile, **$HOME/mbox**.
-p	Displays mail without prompting for a disposition code. This flag does not delete, copy, or forward any messages. (For disposition codes, see "Subcommands" on page 106).
-q	Causes **bellmail** to exit when you press INTERRUPT (**Alt-Pause**). Normally, pressing INTERRUPT (**Alt-Pause**) stops only the message being displayed. (In this case, the next message sometimes does not display until you enter the **p** subcommand.)
-r	Displays mail in first-in, first-out order.
-t	Prefixes each message with the names of all recipients of the mail. (Normally, only the individual recipient's name appears as addressee.)

Usually, *user* is a name recognized by the **login** command. It can also be the ASCII synonym that is automatically defined for any name that contains NLS code points. If the system does not recognize one or more of the specified *user*s or if **bellmail** is interrupted during input, **bellmail** saves messages in the file **$HOME/dead.letter** to allow for editing and resending.

Subcommands

The following subcommands control message disposition:

+	Displays the next mail message (the same as pressing the **Enter** key).
-	Displays the previous message.
d	Deletes the current message and displays the next message.
p	Displays the current message again.
s [*file*]	Saves the message in the named *file* instead of in the default mailfile, **$HOME/mbox**.
w [*file*]	Saves the message, without its postmark, in the specified *file* instead of in the default mailfile **$HOME/mbox**.
m *user*	Forwards the message to the named *user*.
q	Writes any mail not yet deleted to **/usr/mail/***user* and exits. Pressing END OF FILE (**Ctrl-D**) has the same effect.
x	Writes all mail unchanged to **/usr/mail/***user* and exits.
!*AIX-cmd*	Runs the specified AIX command.
*	Displays a subcommand summary.

Examples

1. To display your mail:

   ```
   bellmail
   ```

   After the most recent message is displayed, a ? (question mark) indicates that **bellmail** is waiting for one of the subcommands explained previously (+, -, **d**, **p**, etc.). Enter `help` or `*` (asterisk) to list the subcommands available.

2. To send mail to other users:

   ```
   bellmail tom rachel
   Don't forget the
   meeting tomorrow at 9:30.
   ```

   Ctrl-D

   In this example the system mails the message `Don't forget the meeting tomorrow at 9:30.` to the users `tom` and `rachel`. The **Ctrl-D** indicates the end of the message but it is not sent with the text.

3. To send a file to another user:

   ```
   bellmail fran <proposal
   ```

   This command sends the contents of the file proposal to fran. You can create memo with an editor, which allows you to correct your mistakes before sending the message. You can also use this form of the **bellmail** command to send someone a copy of a data file.

4. To retrieve a file that was sent to you:

   ```
   bellmail
   ```

   This command displays the messages mailed to you one at a time. You need to look at them because the file you want was actually added to **/usr/mail/***user* as a message. You may see several other messages before the file that was sent to you. If so, press the **Enter** key after the ? prompt until the desired file appears. If you go too far, enter the - (minus) subcommand to go back a message. After the ? immediately following the file, enter:

   ```
   w mycopy
   ```

   This command creates a file named mycopy in the current directory that contains the text mailed to you. Actually, you can save a copy of any message this way.

Files

/etc/passwd	To identify sender and locate *user*.
/usr/mail/*user*	Incoming mail for *user*.
$HOME/mbox	Saved mail.
$HOME/dead.letter	Unmailable text.
/tmp/ma*	Temporary file.
/usr/mail/*.lock	Lock for mail directory.

Related Information

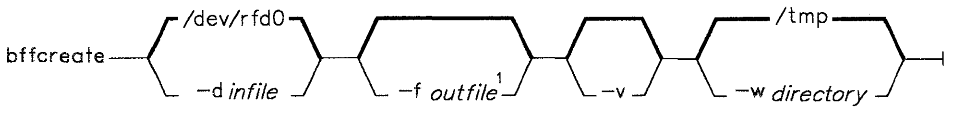
The following commands: "**login**" on page 584, "**uucp**" on page 1144, "**sendmail**" on page 897, and "**write**" on page 1225.

bffcreate

Purpose

Creates files in backup format for complete or subset programs in a code service environment.

Syntax



1 See flag description for special requirements, restrictions, and defaults

AJ2FL137

Description

The **bffcreate** command creates one or more files in backup format to support install and update by client systems in a code service environment. Input files must also be in backup format. You must be a member of the system group or operating with superuser authority to run this command. This command is also run when you specify the **-b** flag in either an **installp** or **updatep** command.

This command creates one or more of the following:

- A file that contains the files from an **installp** distribution media
- One file per program subset that contains the program subset file from an **installp** distribution media
- A file that contains the files from an **updatep** distribution media
- A file that contains the files that do not follow **installp** or **updatep** conventions.

When this command runs, the contents of the distribution media are restored in a temporary working directory. Then a copy is created in backup format and put into either the **/usr/lpp.install** or the **/usr/lpp.update** directory for use in a code service environment. Program subset files are created automatically for any program subset on the distribution media.

Flags

-d *infile*	Specifies the name of the distribution media. If given, it must already exist. The default is **/dev/rfd0**.
-f *outfile*	Specifies the name of the backup format file. This flag is required for non-standard distribution media and **installp** distribution media that contain multiple program names. This flag is not allowed for distribution media that contain only one program name; in this case, the system names the output file in the form *programname.vv.rr* where *programname* is the name of the program, *vv* is the version, and *rr* is the release. This flag is optional for **updatep** distribution media. If not specified, the system creates a name in the form **updt.***yyddd.nnn* where *yyddd* is the Julian date (for example, 88032 is February 1, 1988) and *nnn* represents a number in sequence for files created that day.
-v	Writes the name of the backup format file to standard output (verbose mode).
-w *directory*	Specifies the directory where a temporary working directory can be created to contain the restored files. The default is **/tmp**. If specified, the directory must already exist.

Files

/usr/lpp.install	Directory that contains files in backup format for use in installing complete or subset programs across a network.
/usr/lpp.update	Directory that contains files in backup format for use in updating complete or subset programs across a network.

Related Information

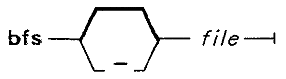
The following commands: "**installp**" on page 529 and "**updatep**" on page 1122.

bfs

Purpose

Scans files.

Syntax



OL805084

Description

The **bfs** command reads a *file* but does not do any processing of it, allowing you to scan but not edit it.

The **bfs** command is basically a read-only version of the **ed** command, except it can process much larger files and it has some additional subcommands. Input files can be up to 32K lines long, with up to 255 characters per line. **bfs** is usually more efficient than **ed** for scanning a file, because the file is not copied to a buffer. It is most useful for identifying sections of a large file where you can use the **csplit** command to divide it into more manageable pieces for editing.

If you enter the **P** subcommand, **bfs** prompts you with an * (asterisk). You can turn off prompting by entering a second **P**. **bfs** displays error messages when prompting is turned on.

Forward and Backward Searches

The **bfs** command supports all the address expressions described under "**ed**" on page 371. In addition, you can instruct **bfs** to search forward or backward through the file, with or without wrap-around. If you specify a forward search with wrap-around, **bfs** continues searching from the beginning of the file after it reaches the end of the file. If you specify a backward search with wrap-around, it continues searching backwards from the end of the file after it reaches the beginning. The symbols for specifying the four types of search are as follows:

/pattern/	Searches forward with wrap-around for the pattern.
?*pattern*?	Searches backward with wrap-around for the pattern.
>*pattern*>	Searches forward without wrap-around for the pattern.
<*pattern*<	Searches backward without wrap-around for the pattern.

The pattern matching routine of **bfs** differs somewhat from the one used by **ed** and includes additional features (see the **regcmp** subroutine in *AIX Operating System Technical Reference*). There is also a slight difference in ***mark names***: only lowercase letters **a** through **z** may be used, and all 26 marks are remembered.

Flags

-	Suppresses the display of file sizes. Normally, **bfs** displays the size in bytes of the file being scanned.

Subcommands

The **e**, **g**, **v**, **k**, **n**, **p**, **q**, **w**, **=** , **!** and null subcommands operate as explained under "**ed**" on page 371. Subcommands such as --, + + +-, + + + =, -12, and +4p are accepted. Note that `1,10p` and `1,10` both display the first ten lines. The **f** subcommand displays only the name of the file being scanned; there are no remembered file names. The **w** subcommand is independent of output diversion, truncation, or compression (see the **xo**, **xt**, and **xc** subcommands on page 111). ***Compressed output*** has strings of tabs and blanks reduced to one blank and blank lines suppressed.

The following additional subcommands are available:

xf *file*	Reads **bfs** subcommands from the *file*. When **bfs** reaches the end of file or receives an **INTERRUPT** signal or if an error occurs, **bfs** resumes scanning the file that contains the **xf** subcommand. These **xf** subcommands may be nested to a depth of 10.
xo [*file*]	Sends further output from the **p** and null subcommands to the named *file*, which is created with read and write permission granted to all users. If you do not specify a *file* parameter, **bfs** writes to standard output. Note that each redirection to a file creates the specified file, deleting an existing file if necessary.
:*label*	Positions a *label* in a subcommand file. The *label* is ended with a new-line character. Blanks between the : (colon) and the start of the *label* are ignored. This subcommand may be used to insert comments into a subcommand file, since labels need not be referenced.

[*addr1*[,*addr2*]]**xb**/*pattern*/*label*

Sets the current line to the line containing *pattern* and jumps to *label* in the current command file if *pattern* is matched within the designated range of lines. The jump fails under any of the following conditions:

- Either *addr1* or *addr2* is not between the first and last lines of the file.
- *addr2* is less than *addr1*.
- The pattern does not match at least one line in the specified range, including the first and last lines.

This subcommand is the only one that does not issue an error message on bad addresses, so it may be used to test whether addresses are bad before other subcommands are run. Note that the subcommand:

```
xb/^/label
```

is an ***unconditional jump***.

The **xb** subcommand is allowed only if it is read from some place other than a work station. If it is read from a pipe, only a ***downward jump*** is possible.

xt *number*
Truncates output from the **p** and null subcommands to *number* characters. The default *number* is 255.

xv[*digit*] [*value*]
Assigns the specified *value* to the variable named *digit* (0 through 9). You can put one or more spaces between *digit* and *value*. For example:

```
xv5  100
xv6  1,100p
```

assigns the value `100` to the variable `5` and the value `1,100p` to the variable `6`.

To reference a variable, put a % (percent sign) in front of the variable name. Given the preceding assignments for variables 5 and 6, the following three subcommands:

```
1,%5p
1,%5
%6
```

each display the first 100 lines of a file.

To escape the special meaning of %, precede it with a \ (backslash). For example:

```
g/".*\%[cds]/p
```

matches and lists lines containing **printf** variables (`%c`, `%d`, or `%s`).

You can also use the **xv** subcommand to assign the first line of command output as the *value* of a variable. To do this, make the first character of *value* an ! (exclamation point), followed by the command name. For example:

```
xv5 !cat junk
```

stores the first line of the file `junk` in the variable `5`.

To escape the special meaning of ! as the first character of *value*, precede it with a \ (backslash). For example:

```
xv7 \!date
```

stores the value `!date` in the variable 7.

xbz *label*
: Tests the last saved exit value from a shell command and jumps to *label* in the current command file if the value is zero.

xbn *label*
: Tests the last saved exit value from a shell command and jumps to *label* in the current command file if the value is not zero.

xc [*switch*]
: Turns compressed output mode on or off. (Compressed output mode suppresses blank lines and replaces multiple blanks and tabs with a single space.)

  If *switch* is 1, output from the **p** and null subcommands is compressed; if *switch* is 0 it is not. If you do not specify *switch*, the current value of *switch* reverses. Initially, *switch* is set to 0.

Related Information

The following commands: "**csplit**" on page 252 and "**ed**" on page 371.

The **regcmp** subroutine in *AIX Operating System Technical Reference.*

biod

Purpose

Starts NFS asynchronous block I/O daemons.

Syntax

```
biod ——— nservers ———┤
```

OL805479

Description

The **biod** command starts asynchronous block I/O daemons. This command is used on an NFS client to handle **read-ahead** and **write-behind** buffer cache. The *nservers* parameter specifies the number of asynchronous block I/O daemons started. Assign the number based on the load expected on the server. Four daemons can handle an average load.

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

File

/etc/rc.nfs

Related Information

The following command: "**nfsd**" on page 696.

biodd_cfg

Purpose

Configures the block I/O AIX device driver.

Syntax

```
biodd_cfg ——|
```

AJ2FL143

Description

The **biodd_cfg** command configures the block I/O kernel device driver so it can access specific adapter cards. The adapter cards which can be accessed by the block I/O kernel device driver are adapters whose VRM device drivers are written to interface with the VRM block I/O device manager. These include the token adapter, the baseband adapter, and multiprotocol/dual port (MPDP) adapter.

Before you run this command, you must edit the **/etc/biodd** file to add the device names for the adapter cards you want to access. After the file has been edited, run the **biodd_cfg** command. This command must be run again after each IPL of the system for the block I/O kernel device driver to be configured to run with the devices listed in the file. To run this command automatically at each IPL, edit the **/etc/rc.include** file to uncomment the line:

```
# /etc/biodd_cfg
```

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Files

/etc/biodd	Contains the device names of the adapters to be accessed by the block I/O kernel device driver.
/etc/rc.include	Contains startup routines.

Related Information

The discussion of the block I/O kernel device driver in *AIX Operating System Technical Reference.*

bj

Purpose

Plays blackjack.

Syntax

```
/usr/games/bj —┤
```

OL805187

Description

The **bj** game plays the role of the dealer in blackjack. The following rules apply.

The bet is $2 every hand. If you draw a ***natural*** (blackjack), you win $3. If the dealer draws a natural, you lose $2. If you and the dealer both have naturals, you exchange no money (a ***push***). If the dealer has an ace showing, you can make an ***insurance*** bet on the chance that the dealer has a natural, winning $2 if the dealer has a natural and lose $1 if not. If you are dealt two cards of the same value, you can ***double***, that is, play two hands, each of which begins with one of these cards, betting $2 on each hand. If the value of your original hand is 10 or 11, you can ***double down***, that is, double the bet to $4 and receive exactly one more card in that hand.

Under normal play, you can draw a card (***hit***) as long as your cards total 21 or less. If the cards total more than 21, you ***bust*** and the dealer wins the bet. When you ***stand*** (decide not to hit), the dealer hits until he has a total of 17 or more. If the dealer busts, you win. If both you and the dealer stand, the one with the higher total wins. A tie is a push.

The **bj** command deals, keeps score, and asks the following questions at appropriate times: ? (Do you want a hit?) `Insurance? Double? Double down?`. To answer yes, press `y`; to answer no, press the **Enter** key.

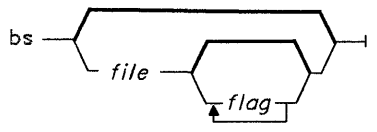
The dealer tells you whenever the deck is being shuffled and displays the ***action*** (total bet) and ***standing*** (total won or lost). To quit the game, press **INTERRUPT** (**Alt-Pause**); **bj** displays the final action and standing and exits.

bs

Purpose

Compiles and interprets modest-sized programs.

Syntax



OL805167

Description

This compiler/interpreter provides interactive program development and debugging. To simplify program testing, it minimizes formal data declaration and file manipulation, allows line-at-a-time debugging, and provides trace and dump facilities and run-time error messages.

The optional command line parameter *file* specifies a file of program statements that the compiler reads before it reads from the standard input. By default, statements read from this file are compiled for later execution. Likewise, statements entered from the standard input are normally executed immediately (see the **compile** keyword on page 119 and the **execute** keyword on page 119). Unless the final operation is assignment, the result of an immediate expression statement is displayed.

Additional command line *flags* can be passed to the program using the built-in functions **arg** and **narg** (explained in more detail on page 123).

Program lines must conform to one of the following formats:

statement
label statement

The interpreter accepts labeled statements only when it is compiling statements. A *label* is a *name* immediately followed by a colon. A label and a variable can have the same name. If the last character of a line is a \ (backslash), the statement continues on the following physical line.

A statement consists of either an expression or a keyword followed by zero or more expressions.

Statement Syntax

break	Exits the innermost **for** or **while** loop.
clear	Clears the symbol table and removes compiled statements from memory. A **clear** is always executed immediately.
compile [*expr*]	Causes succeeding statements to be compiled (overrides the immediate execution default). The optional expression is evaluated and used as a file name for further input. In this latter case, the symbol table and memory are cleared first. **compile** is always executed immediately.
continue	Transfers control to the loop-continuation test of the current **for** or **while** loop.
dump [*name*]	Displays the name and current value of every global variable or, optionally, of the named variable. After an error or interrupt, **dump** displays the number of the last statement and (possibly) the user-function trace.
exit [*expr*]	Returns to the system level. The expression is returned as process status.
execute	Changes to immediate execution mode (pressing INTERRUPT [**Alt-Pause**] has the same effect). This statement does not cause stored statements to execute (see **run** on page 121).

for *name*=*expr expr statement*

for *name*=*expr expr*
  statement . . .
next

for *expr, expr, expr statement*

for *expr, expr, expr*
  statement . . .
next

Repeatedly performs, under the control of a named variable, a statement (first format) or a group of statements (second format). The variable takes on the value of the first expression, then is increased by one on each loop until it exceeds the value of the second expression. The third and fourth formats require three expressions separated by commas. The first of these is the initialization, the second is the test (true to continue), and the third is the loop-continuation action.

fun *f* ([*a*, . . .]) [*v*, . . .]
  statement . . .
nuf

Defines the function name (*f*), parameters (*a*), and local variables (*v*) for a user-written function. Up to 10 parameters and local variables are allowed. Such names cannot be arrays, nor can they be I/O associated. Function definitions may not be nested.

freturn
Signals the failure of a user-written function. Without interrogation, **freturn** returns zero. (See the unary interrogation operator **?** discussed on page 122.) With interrogation, **freturn** transfers to the interrogated expression, possibly bypassing intermediate function returns.

goto *name*
Passes control to the compiled statement with the matching label.

ibase *n*
Sets the input base to *n*. The only supported values for *n* are **8**, **10** (the default), and **16**. Hexadecimal values 10-15 are entered as alphabetic characters **a-f**. A leading digit is required when a hexadecimal number begins with an alphabetic character (for example, `f0a` must be entered as `0f0a`). **ibase** is always executed immediately.

if *expr statement*

if *expr*
  statement . . .
[**else**
  statement . . .]
fi
Performs a statement (first format) or group of statements (second format) if the expression evaluates to nonzero. The strings 0 and "" (null) evaluate as zero. In the second format, an optional **else** allows a group of statements to be performed when the first group is not. The only statement permitted on the same line with an **else** is an **if**; only other **fi**s can be on the same line with a **fi**. You can combine **else** and **if** into **elif**. Only a single **fi** is required to close an **if** . . . **elif** . . . [**else** . . .] sequence.

include *expr*
The expression must evaluate to the name of a file containing program statements. Such statements become part of the program being compiled. **include** statements may not be nested, and are always executed immediately.

obase *n*
Sets the output base to *n*. The only supported values for *n* are **8, 10** (the default), and **16**. Hexadecimal values 10-15 are entered as alphabetic characters **a-f**. A leading digit is required when a hexadecimal number begins with an alphabetic character (that is, `f0a` must be entered as `0f0a`). Like **ibase**, **obase** is always executed immediately.

onintr *label*
onintr
Provides program control of interrupts. In the first format, control passes to the label given, just as if a **goto** had been performed when **onintr** was executed. The effect of the **onintr** statement is cleared after each interrupt. In the second format, pressing **INTERRUPT** (**Alt-Pause**) ends **bs**.

return [*expr*]
Evaluates the expression and passes the result back as the value of a function call. If you do not provide an expression, the function returns zero.

run	Passes control to the first compiled statement. The random number generator is reset. If a file contains a **run** statement, it should be the last statement; **run** is always executed immediately.
stop	Stops execution of compiled statements and returns to immediate mode.
trace [*expr*]	Controls function tracing. If you do not provide an expression or if it evaluates to zero, tracing is turned off. Otherwise, a record of user-function calls/returns will be written. Each **return** decreases by one the **trace** expression value.

while *expr statement*

while *expr*
  statement . . .
next

while is similar to **for** except that only the conditional expression for loop continuation is given.

! *AIXcmd*	Runs an AIX command, then returns control to **bs**.
#comment	Inserts a comment line.

Expression Syntax

name — Specifies a variable or, when followed immediately by a colon, a label. Names are composed of a letter (uppercase or lowercase) optionally followed by letters and digits. Only the first six characters of a name are significant. Except for names declared locally in **fun** statements, all names are global. Names can take on numeric (double float) values or string values or be associated with input/output (see the built-in function **open** on page 125).

name([*expr*[, *expr*] . . .)

Calls function *name* and passes to it the parameters in parentheses. Except for built-in functions (listed in the following text), *name* must be defined in a **fun** statement. Function parameters are passed by value.

name[*expr*[, *expr*] . . .]

References either arrays or tables (see built-in function **table** on page 126). For arrays, each expression is truncated to an integer and used as a specifier for the name. The resulting array reference is syntactically identical to a name; `a[1,2]` is the same as `a[1][2]`. The truncated expressions must be values between 0 and 32767.

number — Represents a constant numerical value. This number can be expressed in integer, decimal, or scientific notation (it can contain digits, an optional decimal point, and an optional **e** followed by a possibly signed exponent).

string Character string delimited by " " (double quotation marks). The \ (backslash) is an escape character that allows the double quotation mark (\"), new-line character (\n), carriage return (\r), backspace (\b), and tab (\t) characters to appear in a string. When not immediately followed by these special characters, \ stands for itself.

(*expr***)** Parentheses alter the normal order of evaluation.

(*expr*, *expr*[, *expr*] . . . **)** [*expr*]
The bracketed expression outside the parentheses functions as a subscript to the list of expressions within the parentheses. List elements are numbered from the left, starting at zero. The expression:

```
(False, True) [ a == b ]
```

has the value `True` if the comparison is true.

expr op expr Except for the assignment, concatenation, and relational operators, both operands are converted to numeric form before the operator is applied.

Unary Operators

?*expr* The interrogation operator (?) tests for the success of the expression rather than its value. It is useful for testing end of file, for testing the result of the **eval** built-in function, and for checking the return from user-written functions (see **freturn** on page 120). An interrogation ***trap*** (end of file, for example), causes an immediate transfer to the most recent interrogation, possibly skipping assignment statements or intervening function levels.

-*expr* Negates the expression.

+ +*name* Increases by one the value of the variable (or array reference).

--*name* Decreases by one the value of the variable.

!*expr* The logical negation of the expression.

Binary Operators (in increasing precedence)

= The assignment operator. The left operand must be a name or an array element. It acquires the value of the right operand. Assignment binds right to left; all other operators bind left to right.

_ The concatenation operator (the underline character).

& | Logical AND, logical OR. The result of:

expr & *expr*

is 1 (true) only if both of its parameters are nonzero (true); it is 0 (false) if one or both of its parameters are 0 (false).

The result of:

expr ¦ *expr*

is 1 (true) if one or both of its expressions are nonzero (true); it is 0 (false) only if both of its expressions are 0 (false). Both operators treat a null string as a zero.

< <= > >= == !=	The relational operators (< less than, . <= less than or equal to, > greater than, >= greater than or equal to, == equal to, != not equal to) return 1 if the specified relation is True. They return 0 (false) otherwise. Relational operators at the same level extend as follows: **a>b>c** is the same as **a>b & b>c**. A string comparison is made if both operands are strings. The comparison is based on the collating sequence specified in the environment variable **NLCTAB**.
+ -	Addition and subtraction.
* / %	Multiplication, division, and remainder.
^	Exponentiation.

Functions Dealing With Arguments

arg(*i*)	Returns the value of the *i*-th actual argument at the current function call level. At level zero, **arg** returns the *i*-th command-line argument. For example, **arg(0)** returns `bs`.
narg( )	Returns the number of arguments passed. At level zero, it returns the command line argument count.

Mathematical Functions

abs(*x*)	Returns the absolute value of *x*.
atan(*x*)	Returns the arctangent of *x*.
ceil(*x*)	Returns the smallest integer not less than *x*.
cos(*x*)	Returns the cosine of *x*.
exp(*x*)	Returns e raised to the power *x*.
floor(*x*)	Returns the largest integer not greater than *x*.
log(*x*)	Returns the natural logarithm of *x*.
rand( )	Returns a uniformly distributed random number between zero and one.
sin(*x*)	Returns the sine of *x*.
sqrt(*x*)	Returns the square root of *x*.

String Functions

size(*s*)	Returns the size (length in characters) of *s*.
bsize(*s*)	Returns the size (length in bytes) of *s*.
format(*f*, *a*)	Returns the formatted value of *a*, *f* being a format specification string in the style of the **printf** subroutine. Use only the **%...f**, **%...e**, and **%...s** formats.
index(*x*, *y*)	Returns a number that is the first position in *x* containing a character that any of the characters in *y* matches. If there is no match, **index** yields zero. For 2-byte extended characters, the index functions returns the location of the first byte.
trans(*s*, *f*, *t*)	Translates characters in the source string *s* which match characters in *f* into characters having the same position in *t*. Source characters that do not appear in *f* are copied unchanged into the translated string. If string *f* is longer than *t*, source characters that match characters found in the excess portion of *f* do not appear in the translated string.
substr(*s*, *start*, *length*)	Returns the substring of *s* defined by *start*ing position in characters and *length* in characters.
match(*string*, *pattern*) **mstring(*n*)**	This function returns the number of characters in *string* that match *pattern*. The characters ., *, ? [,], ^ (when inside square brackets), \(and \) have the following special meanings (see "**ed**" on page 371 for a more detailed discussion of this special notation):

.	Matches any character except the new-line character.
*	Matches zero or more occurrences of the pattern element that it follows (for example, .* matches zero or more occurrences of any character except the new-line character).
$	Specifies the end of the line.
[.-.] [. . .]	Matches any one character in the specified range ([-.]) or list ([. . .]), including the first and last characters.
[^.-.] [^ . . .]	Matches any character except the new-line character and the remaining characters in the range or list. A circumflex (^) has this special meaning only when it immediately follows the left bracket.

[].-.]
[] . . .] Matches] or any character in the list. The right square bracket does not terminate such a list when it is the first character within it (after an initial ^, if any).

\(. . . \) Marks a substring and matches it exactly.

To succeed, a pattern must match from the beginning of the string. It also matches the longest possible string. Consider, for example:

```
match('a123ab123',".*\([a-z]\)") == 6
```

In this instance, `.*` matches `a123a` (the longest string that precedes a character in the range a-z); `\([a-z]\)` matches `b`, giving a total of six characters matched in the string. In an expression such as **[a-z]**, the minus means "through," according to the current collating sequence.

A collating sequence may define ***equivalence classes*** for use in character ranges. See the "Overview of International Character Support" in *Managing the AIX Operating System* for more information on collating sequences and equivalence classes.

Japanese Language Support Information

Note: Japanese Language Support does not define equivalence classes for use in character ranges. To avoid unpredictable results when using a range expression, use a ***character class expression*** rather than a standard range expression. For information about character class expressions, see "File Name Substitution" on page 4.

The **mstring** function returns the *n*th substring in the last call to **match** (*n* must be between 1 and 10 inclusive).

File-Handling Functions

open(*name*, *file*, *mode***)**
close(*name***)** The *name* parameter must be a legal variable name (passed as a string). For **open**, the *file* parameter may be:

- A **0**, **1**, or **2** for standard input, output, or error output, respectively
- A string representing a file name
- A string beginning with an !, representing a command to be run (via `sh -c`).

The *mode* flag must be either **r** (read), **w** (write), **W** (write without new-line character), or **a** (append). After a **close**, the *name* becomes an ordinary variable. The initial associations are:

```
open("get", 0, "r")
open("put", 1, "w")
open("puterr", 2, "w")
```

access(*p*, *m*) Performs the **access** system call. Parameter *p* is the path name of a file; *m* is a bit pattern representing the requested mode of access. This function returns a 0 if the request is permitted, -1 if it is denied. (See *AIX Operating System Technical Reference* for a more extensive discussion of this system call.)

ftype(*s*) Returns a single character indicating file type: **f** for regular file, **p** for FIFO (named pipe), **d** for directory, **b** for block special, or **c** for character special.

Table Functions

table(*name*, *size*) A table in **bs** is an associatively accessed, one-dimensional array. ***Subscripts*** (called keys) are strings (numbers are converted). The *name* parameter must be a **bs** variable name (passed as a string). The *size* parameter sets the minimum number of elements to be allocated. On table overflow, **bs** writes an error message.

item(*name*, *i*)
key( ) The **item** function accesses table elements sequentially (in normal use, there is an orderly progression of key values). Where the **item** function accesses values, the **key** function accesses the subscript of the previous **item** call. The *name* parameter should not be quoted. Since exact table sizes are not defined, the interrogation operator should be used to detect end-of-table; for example:

```
table("t",100)
.
.
.
#If word contains "party", the following expression
#adds one to the count of that word:
++t[word]
.
.
.
# To display the key/value pairs:
for i=0, ?(s=item(t, i)), ++i if key() put=key()_":"_s
```

iskey(*name*, *word*)
Tests whether the key *word* exists in the table *name* and returns one for true, zero for false.

Miscellaneous Functions

eval(*string*) The string parameter is evaluated as an expression. The function is handy for converting numeric strings to numbers. **eval** can also be used as a crude form of indirection, as in:

```
name = "xyz"
eval("++"_name)
```

which increments the variable `xyz`.

In addition, **eval** preceded by the interrogation operator permits you to control **bs** error conditions. For example:

```
?eval("open(\"X\",\"XXX\", \"r\")")
```

returns the value zero if there is no file named `"XXX"` (instead of halting your program). The following performs a **goto** to the label `L:` (if it exists):

```
label="L:"
if!(?eval("goto"_label))puterr="no label"
```

plot(*request*, *args*)
The **plot** function produces output on devices recognized by the **tplot** command. Some requests do not apply to all plotters. All requests except 0 and 12 are implemented by piping characters to **tplot**. The *requests* are as follows:

Call	***Function***
plot(0, *term*)	Causes further **plot** output to be piped into **tplot** with a flag of **-T***term*.
plot(1)	Erases the plotter.
plot (2, *string*)	Labels the current point with *string*.
plot(3, *x1*, *y1*, *x2*, *y2*)	Draws the line between (x1, y1) and (x2, y2).
plot(4, *x*, *y*, *r*)	Draws a circle with center (x, y) and radius *r*.
plot(5, *x1*, *y1*, *x2*, *y2*, *x3*, *y3*)	Draws an arc (counterclockwise) with center (x1, y1) and endpoints (x2, y2) and (x3, y3).
plot(6)	Not implemented.
plot(7, *x*, *y*)	Makes the current point at (x, y).
plot(8, *xy*)	Draws a line from the current point to (x, y).
plot(9, *x*, *y*)	Draws a point at (x, y).

plot(10, *string***)**	Sets the line mode to *string*.
plot(11, *x1*, *y1*, *x2*, *y2***)**	Makes (x1, y1) the lower left corner of the plotting area and (x2, y2) the upper right corner of the plotting area.
plot(12, *x1*, *y1*, *x2*, *y2***)**	Causes subsequent x (y) coordinates to be multiplied by x1 (y1) and then added to x2 (y2) before they are plotted. The initial scaling is **plot(12, 1.0, 1.0, 0.0, 0.0)**.

last() In immediate mode, **last** returns the most recently computed value.

Related Information

The following commands: "**ed**" on page 371, "**sh**" on page 913, and "**tplot**" on page 1079.

The **access** system call, the **printf** subroutine, and the **plot** file in *AIX Operating System Technical Reference.*

"Overview of International Character Support" in *Managing the AIX Operating System.*

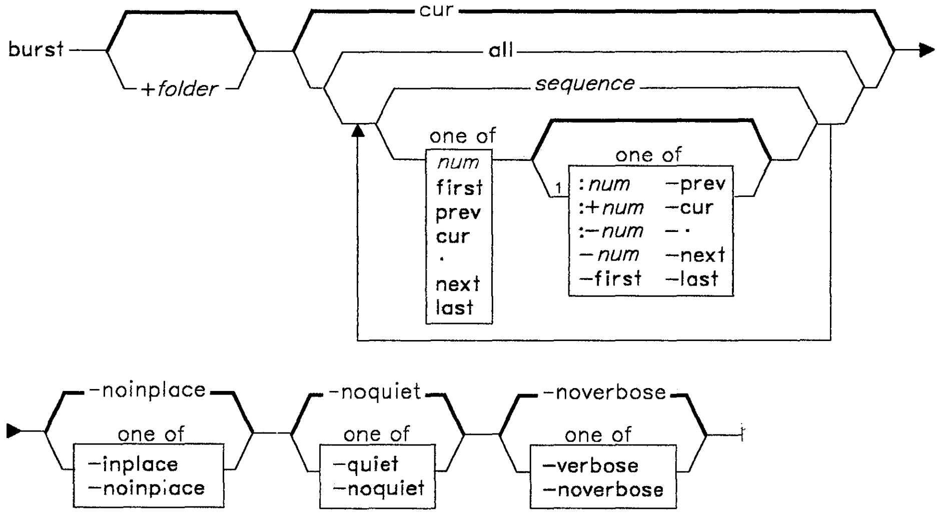
The discussion of Japanese Language Support in *Japanese Language Support User's Guide.*

burst

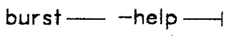
Purpose

Explodes digests into messages.

Syntax



AJ2FL220



AJ2FL160

[1] Do not put a blank between these items.

OL805308

Description

The **burst** command is used to explode digests, messages forwarded by the **forw** command, and blind carbon copies sent by the **forw** and **send** commands. **burst** is part of the MH (Message Handling) package and can be used with other MH and AIX commands.

The **burst** command cannot explode more than about 1,000 messages from a single message. **burst**, however, generally does not place a specific limit on the number of messages in a folder after bursting is complete.

The **burst** command uses encapsulation boundaries to determine where to separate the encapsulated messages. If an encapsulation boundary is located within a message, **burst** may split that message into two or more messages.

Flags

+*folder msgs* Specifies the messages that you want to burst. *msgs* can be several messages, a range of messages, or a single message. You can use the following message references when specifying *msgs*:

num	**first**	**prev**
cur	**.**	**next**
last	**all**	*sequence*

The default message is the current message in the current folder. If **-inplace** is also specified, the first message burst becomes the current message. Otherwise, the first message extracted from the first digest becomes the current message.

-help Displays help information for the command.

-inplace Replaces each digest by a table of contents for the digest, places the messages contained in each digest directly after the digest's table of contents, and renumbers all subsequent messages in the folder to make room for the messages in the exploded digest.

Warning: The **burst** command does not place text that appears after the last encapsulated message in a separate message. When you specify the **-inplace** flag, **burst** loses this trailing text. In digests, this text is usually an End-of-Digest string. However, if the sender appended remarks after the last encapsulated message, **burst** loses these remarks.

-noinplace Preserves each digest, does not produce a table of contents for each digest, and places the messages contained in each digest at the end of the folder. **burst** does not affect messages that are not part of digests. This flag is the default.

-noquiet Reports information about messages that are not in digest format. This flag is the default.

-noverbose	Does not report the general actions that **burst** performs while exploding the digests. This flag is the default.
-quiet	Does not report information about messages that are not in digest format.
-verbose	Reports the general actions that **burst** performs while exploding the digests.

Profile Entries

Current-Folder:	Sets your default current folder.
Msg-Protect:	Sets the protection level for your new message files.
Path:	Specifies your *user_mh_directory.*

Files

$HOME/.mh_profile	The MH user profile.

Related Information

Other MH commands: "**forw**" on page 438, "**inc**" on page 518, "**msh**" on page 677, "**packf**" on page 733, "**send**" on page 893, "**show**" on page 942.

The **mh-profile** file in *AIX Operating System Technical Reference.*

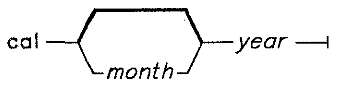
The "Overview of the Message Handling Package" in *Managing the AIX Operating System.*

cal

Purpose

Displays a calendar.

Syntax



OL805168

Description

The **cal** command writes to standard output a calendar for the specified year or month.

The *month* parameter names the month for which you want the calendar. It can be a number between 1 and 12 for January through December, respectively.

The *year* parameter names the year for which you want the calendar. Since **cal** can display a calendar for any year from 1 to 9999, enter the full *year* rather than just the last two digits.

Japanese Language Support Information

The name of the month is taken from the appropriate **NLLMONTH** string. The first two bytes of the **NLSDAY** environment variable string are used as the abbreviation of the day of the week.

Examples

1. To display a calendar for February 1984 at your work station:
   ```
   cal 2 1984
   ```
2. To print a calendar for 1984:
   ```
   cal 1984 ¦ print
   ```
3. To display a calendar for the year 84 A.D.:
   ```
   cal 84
   ```

Related Information

The "Overview of International Character Support" in *Managing the AIX Operating System.*

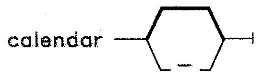
The discussion of Japanese Language Support in *Japanese Language Support User's Guide.*

calendar

Purpose

Writes reminder messages to standard output.

Syntax



OL805169

Description

The **calendar** command reads a file named **calendar**, which you create in your current (usually home) directory. It writes to standard output any line in the file that contains today's or tomorrow's date.

The **calendar** command recognizes date formats such as `Dec. 7` or `12/7`. It also recognizes the special character * (asterisk). It interprets `*/7`, for example, as signifying the seventh day of every month. **calendar** does not recognize formats such as `7 December`, `7/12`, or `DEC. 7`.

On Fridays, **calendar** writes all lines containing the dates for Friday, Saturday, Sunday, and Monday. It does not, however, recognize holidays, so "tomorrow" is the holiday rather than the next working day.

For you to get reminder service, your **calendar** should have read permission for others (see "**chmod**" on page 160).

Flag

- Calls **calendar** for everyone having a file **calendar** in his home directory and sends any reminders by **mail**.

Example

To display information in the **calendar** file that pertains to the next two business days:

```
calendar
```

A typical **calendar** file might look like this:

```
*/25 - Prepare monthly report
Aug. 12 - Fly to Denver
aug 23 - board meeting
Martha out of town - 8/23, 8/24, 8/25
8/24 - Mail car payment
sat aug/25 - beach trip
August 27 - Meet with Simmons
August 28 - Meet with Wilson
```

If today is Friday, August 24, then the **calendar** command displays:

```
*/25 - Prepare monthly report
Martha out of town - 8/23, 8/24, 8/25
8/24 - Mail car payment
sat aug/25 - beach trip
August 27 - Meet with Simmons
```

Files

$HOME/calendar	
/usr/lib/calprog	The program that determines dates.
/etc/passwd	Used to identify users.
/tmp/cal*	Temporary files.

Related Information

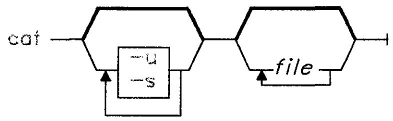
The following commands: "**chmod**" on page 160 and "**mail, Mail**" on page 608.

cat

Purpose

Concatenates or displays files.

Syntax



OL805086

Description

The **cat** command reads each *file* in sequence and writes it to standard output. If you do not specify *file* or specify - (minus) instead of a *file*, **cat** reads from standard input.

Warning: Do not redirect output to one of the input files using the > redirection symbol. If you do this, you will lose the original data in the input file because the shell truncates it before **cat** can read it (see "**sh**" on page 913).

Flags

-s	Does not display a message if **cat** cannot find an input file.
-u	Does not buffer output.

Examples

1. To display a file at the work station:

   ```
   cat  notes
   ```

   This displays the data in the file `notes`. If the file is more than about 23 lines long, some of it will scroll off the screen. To list a file one page at a time, use the **pg** command. (See "**pg**" on page 744 for details.)

2. To concatenate several files:

   ```
   cat  section1.1  section1.2  section1.3  >section1
   ```

   This creates a file named section1 that is a copy of section1.1 followed by section1.2 and section1.3.

3. To suppress error messages about files that do not exist:

   ```
   cat  -s  section2.1  section2.2  section2.3  >section2
   ```

   If section2.1 does not exist, this concatenates section2.2 and section2.3. The result is the same if you do not use the **-s**, except that **cat** displays the error message:

   ```
   cat: cannot open section2.1
   ```

   You may want to suppress this message with the **-s** flag when you use the **cat** command in shell procedures.

4. To append one file to the end of another:

   ```
   cat  section1.4  >>section1
   ```

   This appends a copy of section1.4 to the end of section1. The >> appends data to the end of section1. If you want to replace the file, use the >. For more details, see "Redirection of Input and Output" on page 926.

5. To add text to the end of a file:

   ```
   cat  >>notes
   Get milk on the way home
   ```
   Ctrl-D

   Get milk on the way home is added to the end of notes. The **cat** command does not prompt; it waits for you to enter text. Press **Ctrl-D** to indicate you are finished.

6. To concatenate several files with text entered from the keyboard:

   ```
   cat  section3.1  -  section3.3  >section3
   ```

   This concatenates section3.1, text from the keyboard, and section3.3.

7. To concatenate several files with output from another command:

   ```
   li  |  cat section4.1 -  >section4
   ```

   This copies section4.1, and then the output of the **li** command to the file section4.

Related Information

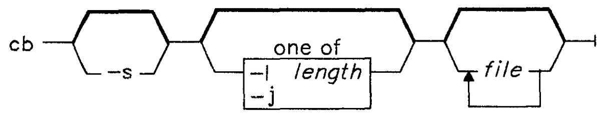
The following commands: "**cp**" on page 202, "**pr**" on page 761, and "**sh**" on page 913.

cb

Purpose

Puts C source code into a form that is easily read.

Syntax



OL805170

Description

The **cb** command reads C programs from standard input or from specified *file*s and writes them to standard output in a form that shows, through indentations and spacing, the structure of the code. When called without flags, **cb** does not split or join lines. Note that punctuation in preprocessor statements can cause indentation errors.

Flags

-j	Joins lines that are split.
-l *length*	Splits lines that are longer than *length*.
-s	Formats the source code according to the style of Kernighan and Ritchie in *The C Programming Language* (Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1978.).

Example

To create a version of `pgm.c` called `pgm.pretty.c` that is easy to read:

```
cb  pgm.c  > pgm.pretty.c
```

Related Information

The following command: "**cc**" on page 140.

The discussion of **cb** in *AIX Operating System Programming Tools and Interfaces.*

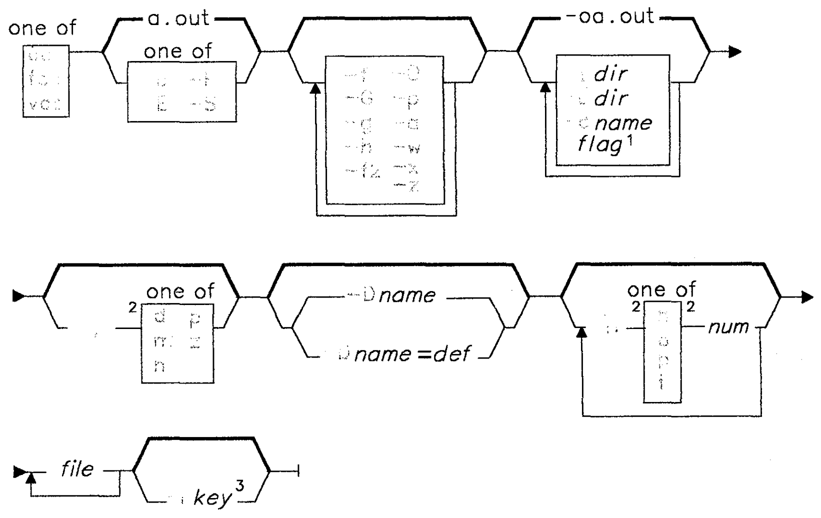
cc

Purpose

Compiles C programs.

Syntax

Ordinary Operation



OL805171

---

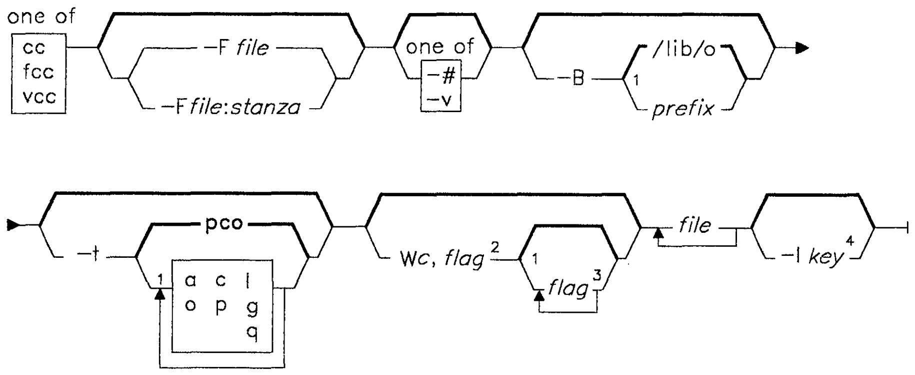
1 Use any flag belonging to **as**, **cpp**, or **ld** (except −l*key*).

2 Do not put a blank between these items.

3 Put this flag last if used (see the **ld** command).

OL805472

Extended Functions and Debugging



OL805389

1 Do not put a blank between these items.

2 Use any flag belonging to **as**, **cpp**, or **ld** (except –l *key*).

3 Use any flag from the first diagram (except –l*key*) or any flag belonging to **as**, **cpp** or **ld**.

4 Put this flag last if used (see the **ld** command).

OL805343

Description

The **cc** command runs the C compiler. It accepts files containing C source code, assembler source code, or object code and changes them into a form that the computer system can run. **cc** compiles and assembles source files and then links them with any specified object files, in the order listed on the command line. It puts the resulting executable program in a file named **a.out**.

The **fcc** command is a link to **cc** that compiles programs to run with the Floating-Point Accelerator. **fcc** should only be used on the 032 family of microprocessors. It automatically uses the **-f** flag as well as special versions of the standard libraries that have been compiled for direct floating-point applications. Note that programs compiled with **fcc** can run only on systems that have installed the Floating-Point Accelerator.

The **vcc** command is a link to **cc** that compiles modules to be installed in the VRM. Use the **vrmfmt** command to convert the **a.out** file produced by the **vcc** command to a VRM-compatible object module. The syntax of this command is as follows:

vrmfmt *infile* [*outfile*]

The default output file name is **a.vrm**.

The **cc** command runs the following programs. Each program processes the source file and then sends the results to the next program in the sequence:

cpp	The macro preprocessor.
ccom0	The first pass of the compiler.
ccomq	The intermediate code optimizer (if you specify the **-O** flag).

This program provides a variety of optimizations to the intermediate code, such as removing loop invariants, eliminating common subexpressions, and allocating registers. The following cannot be optimized:

- Functions that call **setjmp**
- Functions that contain **asm** statements

If you are compiling a large program and the flow optimizer runs out of space, the compiler stops the process and displays a message describing the problem.

ccom1	The second pass of the compiler.
copt	The optimizer (if you specify the **-O** flag).
as	The assembler.
ld	The linkage editor.

You can replace any or all of these passes with your own versions (see the **-B** and **-t** flags). Both **cc** and **fcc** use the **cc.cfg** configuration file, which specifies the standard run time, the link options, and the libraries to be used with each version of the compiler.

Input File Types

The **cc** command recognizes and accepts as input the following file types:

file.c

The name of a C language source file should end with **.c**. After **cc** compiles this source file, it gives the resulting object file the same name, except that it ends in **.o** rather than **.c**. If you use one command both to compile and to load a single C program, the compiler normally deletes the **.o** file when it loads the program. If you use the **-c** flag, the compiler does not delete the **.o** file.

file.i

The name of a file that contains preprocessed C source code ends in **.i**.

file.o

The name of an object file should end in **.o**. The **cc** command sends these files to the **ld** command.

file.s

The name of an assembly language source program should end with **.s**. After **cc** assembles this source file, it gives the resulting object file the same name, except that it ends in **.o** rather than **.s**.

Flags

The **cc** command recognizes several flags. In addition, flags intended to modify the action of the linkage editor (**ld**), the assembler (**as**), or the preprocessor (**cpp**) may also appear on the **cc** command line. **cc** sends any flags it does not recognize to these commands for processing. The following list includes the most commonly used **cpp** flags (**-D**, **-I**), and **ld** flags (**-l**, **-L**, **-o**). See "**as**" on page 61, "**cpp**" on page 210, and "**ld**" on page 557 for a complete list of additional flags.

Note: If you use the **-l** flag, it must be the last entry on the command line, following any file parameters.

Ordinary Operation

-a	Reserves a register for extended addressing. Use this flag if a compiled procedure creates a stack greater than 32,767 bytes. Because this flag causes the compiler to reserve a register for use by the assembler, it reduces the number of available registers by one.
-c	Does not send the completed object file to the **ld** command. With this flag, the output of **cc** is a **.o** file for each **.c** or **.s** file.
-D*name*[=*def*]	Defines *name* as in a **#define** directive. The default *def* is 1.
-E	Runs the named C source file through only the preprocessor and writes the result to standard output.
-f	Generates code that uses the Floating-Point Accelerator or Advanced Floating-Point Accelerator. Programs compiled with this flag will run correctly only on 032 microprocessors configured with either of the Floating-Point Accelerators.
-f2	Generates code that uses the Advanced Floating-Point Accelerator. Programs compiled with this flag will run correctly only on AIX processors configured with the Advanced Floating-Point Accelerator and an Advanced Processor Card.

-g	Produces additional information for use with the **sdb** command (the symbolic debugger).
-G	Indicates that global variables are volatile. The optimizer (**ccomq**) makes fewer transformations when you specify this flag. To make a particular variable volatile, add the "volatile" specification to its declaration.
-h	Treats files with the suffix **.h** in the same way as files with the suffix **.c**.
-I*dir*	Looks first in *dir*, then looks in the directories on the standard list for **#include** files with names that do not begin with / (slash).
-l[*key*]	Searches the specified library file, where *key* selects the file **lib***key***.a**. With no *key*, **-l** selects **libc.a**, the standard system library for C and assembly language programs. **ld** searches for this file in the directory specified by an **-L** flag, then in **/lib** and **/usr/lib**. The **ld** command searches library files in the order in which you list them on the command line.
-L*dir*	Looks in *dir* for files specified by **-l** keys. If it does not find the file in *dir*, **ld** searches the standard directories.
-N[ndpt]*num*	Changes the size of the symbol table (**n**), the dimension table (**d**), the constant pool (**p**), or the space for building the parse tree (**t**). Each table must be changed separately. The default size of the symbol table is 1500; the default size of the dimension table is 2000; the default size for the constant pool is 600; the default space for the parse tree is 1000.
-o*name*	Assigns *name* rather than **a.out** to the output file.
-O	Sends compiler output to the code optimizers.
-p	Prepares the program so that the **prof** command can generate an execution profile. The compiler produces code that counts the number of times each routine is called. If programs are sent to **ld**, the compiler replaces the startup routine with one that calls the **monitor** subroutine at the start (see *AIX Operating System Technical Reference* for a discussion of this subroutine), and writes a **mon.out** file when the program ends normally.
-P	Sends the specified C source file to the macro preprocessor and stores the output in a **.i** file.
-Q!	Controls inlining. The following may be used:

?	Shows the reason for not inlining in the output file.
-name,name . . .	Does not inline *name*.
+name,name . . .	Inlines *name*.

|*num*	Limits the size increase of the function in which inlining occurs to *num* intermediate operations. The default *num* is 100.
#*num*	Limits the expansion of an individual call to *num* intermediate operators. The default *num* is 100.
-@*file*	Reads a list of forbidden functions from *file*.
+@*file*	Reads a list of requested functions from *file*.

Requesting a function to be inlined overrides size constraints.

-S Compiles the specified C programs, storing assembly language output in a **.s** file.

-w Prevents printing of warning messages about functions that cannot be optimized.

-X Produces an assembler listing. This is stored in a file that has the same name as the assembler source file but with the extension **.lst** instead of **.s**.

-y[dmnpz] Specifies the rounding mode for floating-point constant folding. These modes are specified as follows:

d	Disables floating-point constant folding.
m	Rounds toward negative infinity.
n	Rounds to nearest whole number. This is the default action and applies to constant folding in all applicable passes of the compiler.
p	Rounds toward positive infinity.
z	Rounds toward 0.

-z Uses the **libm.a** version, or a version specified by the user, of the following transcendental functions:

acos	asin	atan	atan2	cos	exp
log	log10	sin	sqrt	tan	

If this flag is not used, the compiler generates calls to the AIX kernel, or the Advanced Floating Point Accelerator if possible. For more information on **libm.a**, see **math.h** in *AIX Operating System Technical Reference*. For more information on the Advanced Floating Point Accelerator, see **fpfp** in *AIX Operating System Technical Reference*.

Debugging

-F*file*[:*stanza*]	Uses an alternative *file* and/or *stanza* for **cc** configuration (see *AIX Operating System Technical Reference* for a discussion of the configuration file, **cc.cfg**). If used, this flag must be the first flag on the command line.
-v	Displays the trace as with **-#** and invokes the programs.
-#	Displays a trace of the actions to be taken (for example, invoking the preprocessor), without actually invoking any programs.

Extended Functions

-B*prefix*

Constructs path names for substitute preprocessor, compiler, optimizer, assembler, or linkage editor programs. *prefix* defines part of a path name to the new programs. To form the complete path name for each new program, **cc** adds *prefix* to the standard program names (see the discussion of the programs called by **cc** on page 142). For example, if you enter the command:

```
cc  testfile.c  -B/usr/jim/new
```

cc calls the following compiler programs:

1. **/usr/jim/newcpp**
2. **/usr/jim/newccom0**
3. **/usr/jim/newccom1**
4. **/usr/jim/newas**
5. **/usr/jim/newld**

Similarly, if you enter the command:

```
cc  testfile.c  -B/usr/jim/new/
```

cc calls the following compiler programs:

1. **/usr/jim/new/cpp**
2. **/usr/jim/new/ccom**
3. **/usr/jim/new/ccom1**
4. **/usr/jim/new/as**
5. **/usr/jim/new/ld**

The default *prefix* is **/lib/o**.

-t[pcqgoal]

Applies the **-B** flag instructions for constructing file names to only the designated preprocessor (**p**), compiler first (**c**), intermediate code optimizer (**q**), compiler second (**g**), optimizer (**o**), assembler (**a**), or linkage editor (l) passes. You can select any combination of **pcqgoal**.

The **-t** flag with no additional **p**, **c**, **q**, **g**, **o**, **a**, or **l** designates by default the preprocessor, compiler and optimizer programs (see the discussion of the programs called by **cc** on page 142).

If you do not specify the **-B** flag when you specify the **-t** flag, the default file name *prefix* is **/lib/n**.

Note: You can specify this *prefix* with the **-B** flag. However, depending on what combination of the **-B** and the **-t** flags you specify, *prefix* can have two possible default values. If you specify **-B** but no accompanying *prefix*, the default *prefix* is **/lib/o**. If you specify the **-t** flag without also specifying the **-B** flag, the default *prefix* is **/lib/n**.

-W*c*,*flag1*[,*flag2* . . .]
Gives the listed flags to the compiler program *c*; *c* can be any one of the values [**pcqgoal**] discussed with the **-t** flag. For example, since both **ld** and **as** recognize a **-o** flag, use **-W** to specify the program to which the flag is to be sent. That is, **-Wl,-o** sends it to **ld**. **-Wa,-o** sends it to **as**.

Examples

1. To compile and link a C program, creating an executable **a.out** file:

   ```
   cc pgm.c
   ```

2. To compile a program, producing an object file to be linked later:

   ```
   cc -c pgm.c
   ```

   This compiles `pgm.c` and produces an object file named `pgm`**.o**.

3. To compile a program to run on the Floating-Point Accelerator:

   ```
   fcc pgm.c
   ```

   This compiles `pgm.c` using the special libraries **libfc.a** and **libfm.a** instead of the standard libraries **libc.a** and **libm.a**.

4. To view the output of the macro preprocessor:

   ```
   cc -P -C pgm.c
   ```

   This creates a file named `pgm`**.i** that contains the preprocessed program text including comments. To view this file, use an editor or see "**pg**" on page 744. **cc** passes the **-P** and **-C** flags to the preprocessor. See "**cpp**" on page 210 for more details about them.

5. To predefine macro identifiers:

   ```
   cc -DBUFFERSIZE=512 -DDEBUG pgm.c
   ```

   This assigns BUFFERSIZE the value 512 and DEBUG the value 1 before preprocessing. **cc** passes the **-D** flag to the preprocessor.

6. To use **#include** files located in nonstandard directories:

   ```
   cc -I/u/jim/include pgm.c
   ```

   This looks in the directory that contains pgm.c for the **#include** files with names enclosed in double quotes (" "), then in /u/jim/include, and then in the standard directories. It looks in /u/jim/include for **#include** file names enclosed in angle brackets (< >), then in the standard directories. **cc** passes the **-I** flag to the preprocessor.

7. To optimize the object code and produce an assembler listing:

   ```
   cc -S -O pgm.c
   ```

   This uses the optimizing compiler (**-O** is minus, capital oh), and produces an assembler listing in a file named pgm**.s** (**-S**).

Files

file.c	C source file.
file.o	Object file.
file.s	Assembler file.
a.out	Linked output.
/etc/cc.cfg	**cc** configuration file.
/tmp/ctm*	Temporary.
/lib/cpp	C preprocessor.
/lib/ccom0	Compiler first pass.
/lib/ccomq	Intermediate code optimizer.
/lib/ccom1	Compiler second pass.
/lib/cgen	Compiler.
/lib/copt	Optimizer.
/bin/as	Assembler.
/bin/ld	Linkage editor.
/lib/crt0.o	Run-time startoff.
/lib/mcrt0.o	Run-time startoff for profiling.
/lib/libc.a	Standard library.
/lib/libfc.a	Standard library for use with Floating-Point Accelerator.

/lib/libm.a	Standard math library.
/lib/libfm.a	Standard math library for use with Floating-Point Accelerator.
/lib/librts.a	Runtime services.
/usr/include	Standard directory for #include files.
/usr/tmp/ctm*	Temporary.

Related Information

The following commands: "**as**" on page 61, "**ld**" on page 557, "**cpp**" on page 210, "**prof**" on page 773, and "**sdb**" on page 875.

The discussion of **cc** in *AIX Operating System Programming Tools and Interfaces*, in *C Language Guide and Reference* and in *Assembler Language Reference.*

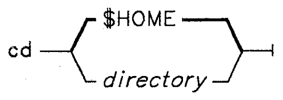
The **monitor** subroutine, the **a.out** and **cc.cfg** files, the discussion of the Advanced Floating Point Accelerator (**fpfp**), and **math.h** in *AIX Operating System Technical Reference.*

cd

Purpose

Changes the current directory.

Syntax



OL805087

Description

The **cd** command moves you from your present directory to another. You must have execute (search) permission in the specified *directory*.

If you do not specify a *directory*, **cd** moves you to your login directory (**$HOME**). If the specified *directory* name is a full path name, it becomes the current directory. A full path name begins with a / (slash—root directory), with a . (dot—current directory), or with a .. (dot dot—parent directory). If the directory name is not a full path name, **cd** searches for it relative to one of the paths specified by the **$CDPATH** shell variable. This variable has the same syntax as, and similar semantics to, the **$PATH** shell variable. (See "Shell Variables and Command-Line Substitutions" on page 917 for a discussion of these variables.)

Examples

1. To change to your home directory:

   ```
   cd
   ```

2. To change to an arbitrary directory:

   ```
   cd  /usr/include
   ```

   This changes the current directory to `/usr/include`. Now file path names that do not begin with / or ../ specify files located in `/usr/include`.

3. To go down one level of the directory tree:

   ```
   cd  sys
   ```

   If the current directory is `/usr/include` and if it contains a subdirectory named `sys`, then `/usr/include/sys` becomes the current directory.

4. To go up one level of the directory tree:

   ```
   cd  ..
   ```

   The special file name `..` (dot-dot) refers to the directory immediately above the current directory. However, under symbolic links, `..` (dot-dot) refers to the parent directory of the symbolic link, not to the directory above the current directory.

Related Information

The following commands: "**csh**" on page 225, "**pwd**" on page 800, and "**sh**" on page 913.

Note: The **csh** and **sh** commands each contain a built-in subcommand named **cd**. The description of **cd** given above applies only to **sh**. For more information see the **csh** command.

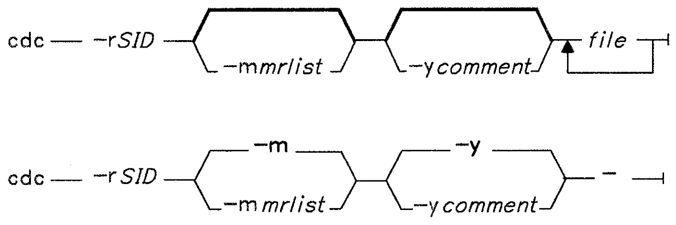
The **chdir** system call in *AIX Operating System Technical Reference.*

cdc

Purpose

Changes the comments in a Source Code Control System (SCCS) delta.

Syntax



OL805088

Description

The **cdc** command changes the ***Modification Requests*** (MRs) and comments for the *SID* specified by the **-r** flag for each named ***Source Code Control System*** (SCCS) *file*. If you specify a directory name, **cdc** performs the requested actions on all SCCS files in that directory (that is, all files with names that have the **s.** prefix). If you specify a **-** (minus) in place of *file*, **cdc** reads standard input and interprets each line as the name of an SCCS file. For more information on SCCS comments and Modification Requests, see *AIX Operating System Programming Tools and Interfaces.*

You can change the comments and MRs for an SID only if you made the SID or you own the file and the directory. For more information on the permissions needed to change SCCS files, see "SCCS Files" on page 478.

Flags

-m[*mrlist*]	Supplies a list of MR numbers for **cdc** to add or delete in the SID specified by the **-r** flag. You can only use this flag if the *file* has the **v** header flag set (see Figure 1 on page 44). A null MR list has no effect.

In the *mrlist*, MRs are separated by blanks, tab characters, or both. To delete an MR, precede the MR number with an ! (exclamation point). If the MR you want to delete is currently in the list of MRs, it is changed into a comment line. **cdc** places a list of all deleted MRs in the comment section of the delta and precedes them with a comment line indicating that the following MRs were deleted.

If you do not specify the **-m** flag, and the **v** header flag is set, MRs are read from standard input. If standard input is a work station, **cdc** prompts you for the MRs. The first new-line character not preceded by a backslash ends the list on the command line. **cdc** continues to take input until it reads an end-of-file character (**Ctrl-D**) or a blank line. MRs are always read before comments (see the **-y** flag).

If the **v** flag has a value, **cdc** interprets the value as the name of a program which validates the MR numbers. If the MR number validation program returns a nonzero exit value, **cdc** stops and does not change the MRs.

-r*SID*	Specifies the SCCS identification number of the delta for which **cdc** will change the comments or MRs.
-y[*comment*]	Specifies text to replace any *comment* already existing for the delta specified by the **-r** flag. **cdc** keeps the existing comments and precedes them by a comment line stating that they were changed. A null *comment* has no effect.

If you do not specify **-y**, **cdc** reads comments from standard input until it reads an end-of-file character. If the standard input is a work station, **cdc** prompts for the comments and also allows a blank line to end input. If the last character of a line is a \ (backslash), **cdc** ignores it and continues to read standard input.

Note: If **cdc** reads standard input for file names (that is, when you specify a file name of -), you must use the **-y** and **-m** flags.

Related Information

The following commands: "**admin**" on page 41, "**delta**" on page 310, "**get**" on page 477, "**help**" on page 513, and "**prs**" on page 781.

The **sccsfile** file in *AIX Operating System Technical Reference.*

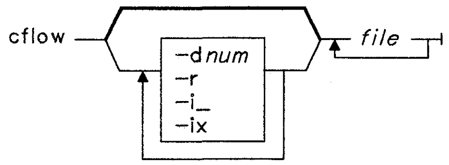
The discussion of SCCS in *AIX Operating System Programming Tools and Interfaces.*

cflow

Purpose

Generates a C flow graph of external references.

Syntax



OL805172

Description

The **cflow** command analyzes C, **yacc**, **lex**, assembler, and object *file*s and writes a chart of their external references to standard output.

It sends files with suffixes **.y**, **.l**, and **.c** to the **yacc**, **lex**, and **cpp** commands for the appropriate processing. This step is bypassed for **.i** files. It then runs the output of this processing through the first pass of **lint**. It assembles files which end in **.s**, extracting information from the symbol table (as it does with **.o** files). From this output, **cflow** produces a graph of external references, which it writes to standard output.

Each line of output begins with a line number followed by sufficient tabs to indicate the level of nesting. Then comes the name of the global, a colon, and its definition. This name is normally a function not defined as external and not beginning with an underline character; see the **-i_** inclusion flag on p. 155. For information extracted from C source files, the definition consists of an abstract type declaration (for example, `char*`), the name of the source file, surrounded by angle brackets, and the line number on which the definition was found. Definitions extracted from object files contain the file name and location counter under which the symbol appeared. **cflow** deletes leading underline characters in C-style external names.

Once **cflow** displays the definition of a name, later references to it contain only the **cflow** line number where the definition may be found. For undefined references, **cflow** displays only <> (redirection symbols).

If the nesting level becomes too deep to display in available space, pipe the output from **cflow** to the **pr** command, using the **-e** flag to compress the tab expansion to something less than every eight spaces.

Note: Files produced by **lex** and **yacc** cause the reordering of line number declarations which can confuse **cflow**. To get proper results, feed **cflow** the **yacc** or **lex** input.

Flags

In addition to the following, **cflow** recognizes the **-I**, **-D**, and **-U** flags of the **cpp** command.

-d*num*	Sets to decimal integer *num* the depth at which the flow graph is cut off. By default this is a very large number. Do not set the cutoff depth to a nonpositive integer.
-ix	Includes external and static data symbols. The default includes only functions.
-i_	Includes names that begin with an underline character. The default excludes these functions (and corresponding data if **-ix** is used).
-r	Produces an inverted listing which shows the callers of each function, sorted by called function.

Related Information

The following commands: "**as**" on page 61, "**cc**" on page 140, "**lex**" on page 562, "**lint**" on page 577, "**nm**" on page 705, "**pr**" on page 761, and "**yacc**" on page 1237.

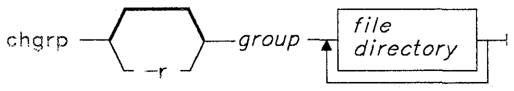
The discussion of **cflow** in *AIX Operating System Programming Tools and Interfaces.*

chgrp

Purpose

Changes the group ownership of a file or directory.

Syntax



OL805090

Description

The **chgrp** command changes the group associated with the specified *file* or *directory* to *groupname* or *groupID*. If you do not own the file, you must have superuser authority to change the group ID.

If the file or directory resides in a Distributed Services remote virtual file system, the translated group ID is used.

Flag

-r	Causes the untranslated group ID to be used. Applies only to files or directories that reside in a Distributed Services remote virtual file system.

Examples

To change the group ownership of the file or directory named `proposals` to `staff`:

```
chgrp  staff  proposals
```

The group access permissions for `proposals` now apply to the `staff` group.

Files

/etc/group	File that identifies all known groups.

Related Information

The following command: "**groups**" on page 506.

The **chown** and **chownx** system calls and the **group** file in *AIX Operating System Technical Reference.*

"Distributed Services **id** Translation" in *Managing the AIX Operating System.*

chkcomp

Purpose

Checks compatibility between a code server and an active-service client.

Syntax

```
chkcomp —— -n node —— -w directory ——|
```

AJ2FL261

Description

The **chkcomp** command checks compatibility between a code server and an active-service client before allowing active service. Generally, this command is run by the **chngstate** command. You must be a member of the system group or operating with superuser authority to run this command.

When the command runs, it compares one or more programs installed at the code server to a program or program subset installed at the client and identifies incompatibilities between version, release, or level. If the client and server are compatible, **chkcomp** ends. If the client and server are not compatible, **chkcomp** writes a comprehensive set of install and update orders to a **cs.compat** file.

There are two types of incompatibilities: incompatibilities that can be fixed by the system and incompatibilities that require manual intervention. These types of incompatibilities are handled in the following manner:

- If the incompatibilities can be fixed by the system and the upgrade mode is automatic, the system automatically initiates the required install and update orders.
- If the incompatibilities can be fixed by the system and the upgrade mode is manual, the system displays the list of required manual actions.
- If the incompatibilities cannot be fixed by the system, the client defaults to stand-alone mode and the user is responsible for initiating the required install and update actions (orders in the **cs.compat** file) to resolve the incompatibilities.

The upgrade mode is set by a value in the **serverattach** file or by **-m** flag to the **chngstate** command. For more information on the compatibility rules and which types of incompatibilities can be fixed by the system, see *Managing the AIX Operating System.*

Flags

-n *node*	Specifies the node ID or nickname of the code server. This flag and *node* are required and must be specified.
-w *directory*	Specifies the directory on the client where the code server's root file system is mounted. This mount must be done before **chkcomp** is run. If **chkcomp** is run from the **chngstate** command, the mount is done for you. If you want to run **chkcomp** directly, you must run the **mount** command before you run the **chkcomp** command. This flag and directory are required and must be specified.

Example

To create and mount a directory where a code server's root file system can reside on your system and to check compatibility between the code server and your system:

```
mkdir /tmp/nick
mount -n darlene / /tmp/nick
chkcomp -n darlene -w /tmp/nick
```

This makes the directory `/tmp/nick`, mounts the / (root) directory of the code server `darlene` on this directory, and then checks the compatibility of the code server and your system.

Files

/etc/codeserve/cs.compat	Contains information on client and server compatibility.
/etc/codeserve/serverattach	Contains code service attribute information.
/usr/lpp/*lpp_name*/lpp.hist	Contains history files for both the code server and the client
/vrm/lpp/*lpp_name*/lpp.hist	Contains history files for both the code server and the client systems.

Related Information

The following command: "**chngstate**" on page 164.

The **/etc/codeserve/servattach** file in *AIX Operating System Technical Reference.*

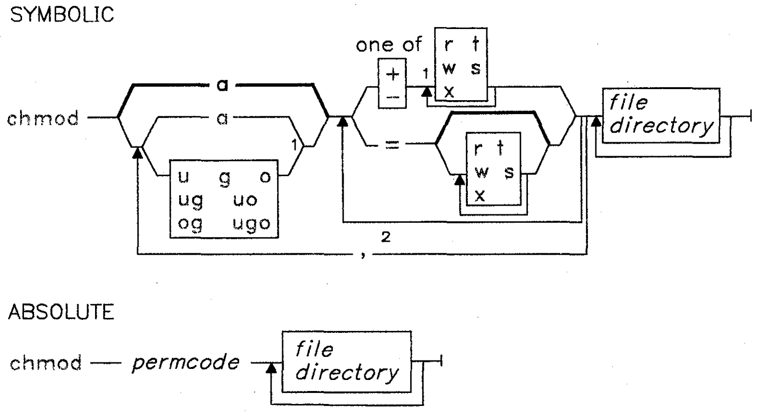
The discussion of code service and the **/etc/codeserve/cs.compat** file in *Managing the AIX Operating System.*

chmod

Purpose

Changes permission codes.

Syntax



[1] Do not put a blank between these items.

[2] Do not put a blank on either side of the comma.

OL805091

Description

The **chmod** command modifies the read, write, execute (*file*), or search (*directory*) permission codes of specified files or directories. You can use either symbolic or absolute mode to specify the desired permission settings.

You can change the permission code of a file or directory only if you own it or if you are operating with superuser authority.

Symbolic Mode

When you use the symbolic mode to specify permission codes, the first set of flags selects the permission field, as follows:

u	User (owner)
g	Group
o	All others
a	User, group, and all others (same effect as **ugo**). This is the default permission field.

The second set of flags selects whether permissions are to be taken away, added , or set exactly as specified:

-	Removes specified permissions
+	Adds specified permissions
=	Clears the selected permission field and sets it to the code specified. If you do not specify a permission code following =, **chmod** removes all permissions from the selected field.

The third set of flags of the **chmod** command selects the permissions as follows:

r	Read permission.
w	Write permission.
x	Execute permission for files; search permission for directories.
s	Set User-ID or Set Group-ID permission. This permission bit sets the effective user-ID or group-ID to that of the *file* whenever the *file* is run. Use this permission setting in combination with the **u** or **g** field to allow temporary or restricted access to files not normally accessible to other users. An **s** appears in the user or group execute position of a long listing (see "**ls**" on page 595 or "**li**" on page 567), to show that the file runs Set User-ID or Set Group-ID.
t	The save text permission. Setting this permission bit causes the text segment of a program to remain in virtual memory after its first use. The system thus avoids having to transfer the program code of frequently-accessed programs into the paging area. A character special file with this bit set is a multiplexed file. You can specify this permission only with the **u** field. A **t** appears in the execute position of the All Others field to indicate that the file has this bit (the ***sticky*** bit) set.

You can specify multiple symbolic modes, separated with commas. Do not separate items in this list with spaces. Operations are performed in the order they appear from left to right.

Absolute Mode

The **chmod** command also permits you to use octal notation to set each bit in the permission code. **chmod** sets the permissions to the *permcode* you provide. This *permcode* is constructed by combining (the logical OR of) the following values:

4000	Sets user-ID on execution
2000	Sets group-ID on execution
1000	Retains memory image after execution (executable file)
1000	Indicates multiplexed character special file
0400	Permits read by owner
0200	Permits write by owner
0100	Permits execute or search by owner
0040	Permits read by group
0020	Permits write by group
0010	Permits execute or search by group
0004	Permits read by others
0002	Permits write by others
0001	Permits execute or search by others

All permission bits not explicitly specified are cleared.

Examples

1. To add a type of permission to several files:

   ```
   chmod  g+w  chap1 chap2
   ```

   This adds write permission for group members to the files `chap1` and `chap2`.

2. To make several permission changes at once:

   ```
   chmod  go-w+x  mydir
   ```

   This denies group members and others the permission to create or delete files in `mydir` (`go-w`). It allows them to search `mydir` or use it in a path name (`go+x`). This is equivalent to the command sequence:

   ```
   chmod  g-w  mydir
   chmod  o-w  mydir
   chmod  g+x  mydir
   chmod  o+x  mydir
   ```

3. To permit only the owner to use a shell procedure as a command:

   ```
   chmod  u=rwx,go=  cmd
   ```

   This gives read, write, and execute permission to the user who owns the file (`u=rwx`). It also denies the group and others the permission to access `cmd` in any way (`go=`).

If you have permission to execute the shell command file cmd, then you can run it by entering:

```
cmd
```

This may not work in some cases, depending on the value of the shell variable **PATH**. See page 923 for more information about **PATH**.

4. To use Set-ID modes:

   ```
   chmod  ug+s  cmd
   ```

   When cmd is executed, this causes the effective user and group IDs to be set to those that own the file cmd. Only the effective IDs associated with the subprocess that runs cmd are changed. The effective IDs of the shell session remain unchanged.

   This feature allows you to permit restricted access to important files. Suppose that the file cmd has the Set-User-ID mode enabled and is owned by a user called dbms. dbms is not actually a person, but might be associated with a database management system. The user betty does not have permission to access any of dbms's data files. However, she does have permission to execute cmd. When she does so, her effective user ID is temporarily changed to dbms, so that the cmd program can access the data files owned by dbms.

   This way betty can use cmd to access the data files, but she cannot accidentally damage them with the standard shell commands.

5. To use the absolute mode form of the **chmod** command:

   ```
   chmod  644  text
   ```

   This sets read and write permission for the owner, and it sets read-only mode for the group and others.

Related Information

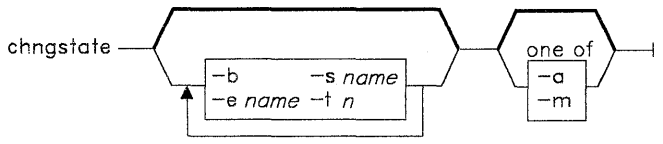
The following commands: "**ls**" on page 595, "**li**" on page 567, and "**umask**" on page 1110.

chngstate

Purpose

Changes the state of a code service client to either active-service or stand-alone.

Syntax



AJ2FL263

Description

Use the **chngstate** command to change the state of a code service client to active-service or stand-alone. You must be a member of the system group or operating with superuser authority to run this command. It is also run by the **rc** command during system initialization.

When the command runs, it validates and processes the code service attribute file **/etc/codeserve/serverattach**. The contents of this file tell **chngstate** whether the system should be an active-service system or a stand-alone system.

If the target state is stand-alone, **chngstate** runs **rc.standalone** and **rc.include.**

If the target state is active-service, **chngstate** checks the attribute file to determine:

- The server to use
- The time between attach attempts for the specified server
- The maximum time to attempt individual server attach
- The upgrade mode (automatic or manual).

Then **chngstate** runs the **chkcomp** command to check for client-server compatibility before attempting to attach to the code server in active-service mode.

If the client and server are compatible, **chngstate** runs **rc.actvsrve** and **rc.include** to attach the client to the code server in active-service mode. If the client and server are not compatible, **chkcomp** writes a comprehensive set of install and update orders to a **cs.compat** file.

There are two types of incompatibilities: incompatibilities that can be fixed by the system and incompatibilities that require manual intervention. These incompatibilities are handled in the following manner:

- If the incompatibilities can be fixed by the system and the upgrade mode is automatic, the system attempts to fix incompatibilities by calling the internal command **installc** to upgrade the install and update state of the client.
- If the incompatibilities can be fixed by the system and the upgrade mode is manual, the system displays the list of required manual actions.
- If the incompatibilities cannot be fixed by the system, the client defaults to stand-alone mode.

In the latter two cases, the user is responsible for initiating the required install and update actions (orders in the **cs.compat** file) to resolve the incompatibilities. The system will not attach a client as long as incompatibilities exist. To manually resolve incompatibilities:

- Run **installp** or **updatep** to install or update programs on the client.
- Rerun the **chngstate** command to verify compatibility and attach to the server as an active-service client.

For more information on the compatibility rules and which types of incompatibilities can be fixed by the system, see *Managing the AIX Operating System.*

Notes:

1. The upgrade mode parameter (automatic or manual) must be specified in the attribute file or an error will result. However, you can override the attribute in the file by specifying the **-a** or **-m** flag in the **chngstate** command.
2. If **chngstate** encounters a server timeout error while attempting to attach to a server, it reads the next system attribute file stanza and attempts to attach to that server according to the stanza attributes. Any other error causes the system to come up in stand-alone mode.
3. You can also run the **ckcomp** command from the command line to get a list of which programs are incompatible. However, **ckcomp** does not leave you attached to the server in active-service mode.

Flags

-a	Uses automatic upgrade mode when attempting to attach to any server with a target state of active-service. This flag overrides the mode set in the system attribute file stanzas.
-b	Prevents **chngstate** from running a **killall** command. This flag should be used only when **chngstate** is run from **/etc/rc** during a system boot.
-e *name*	Excludes the attribute file stanza specified by *name*. To exclude more than one stanza, enter the **-e** flag for each stanza.

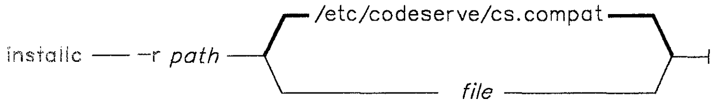
-m	Uses manual upgrade mode when attempting to attach to any server with a target state of active-service. This flag overrides the mode set in the system attribute file stanzas.
-s *name*	Starts processing with this system attribute file stanza.
-t *n*	Specifies the total time in seconds to attempt to attach to any server. The time, *n*, must be greater than or equal to 60.

Internal Commands

The **chngstate** command uses the following internal commands. Because they are internal commands, they do minimum validation of input parameters.

installc

The **installc** command installs a full program or subset program. It uses the following syntax:



AJ2FL140

The **installc** command attempts to upgrade the installation or update state of an active-service client to make it fully compatible with a specific server by installing a full program or a subset program on the active-service client. Generally, **installc** is run by the **chngstate** command when there is an incompatibility that can be automatically corrected by installing a complete or subset program. A user's path would not normally include this command. It is located at **/etc/codeserve/installc**. The **installc** command requires you to be a member of the system group or operating with superuser authority.

When run, **installc** processes install and update requests from an input file in the format defined in the file **cs.compat**. Install requests result in a call to the command **installp** and update requests result in a call to the internal command **updatec**.

The **chngstate** command runs this command when the **cs.compat** file contains only install, update, or install and update records. If **cs.compat** does not exist or contains dbos, uplevel, or unknown records, **installc** will not be run. In this case the user must manually upgrade the client by running the **installp** or **updatep** command for the appropriate programs.

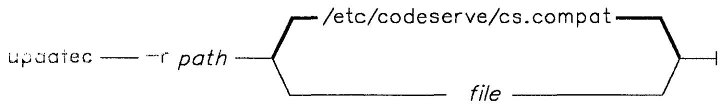
If install records exist, then **chngstate** mounts the server directory **/usr/lpp.install** prior to running **installc**. If update records exist, then **chngstate** mounts the server directory **/usr/lpp.update**.

The **-r** *path* parameter is used to pass the path that **installc** must use to gain access to the server's **/usr/lpp.install** directory. It represents a local path where **chngstate** has mounted the / (root) directory of the server.

The *file* is an input file in **cs.compat** format. This file contains specific install and update requests required to make the client fully compatible with a specific code server.

updatec

The **updatec** command controls the update process for complete or subset programs on active-service clients in a code service environment. It uses the following syntax:



AJ2FL141

Generally, this command is run from the **installc** internal command to upgrade the update state of an active service client and make it fully compatible with a specific server on an active service network. A user's path would not normally include this file. It is located at **/etc/codeserve/updatec**. You must be a member of the system group or be operating with superuser authority to run this command.

When run, this command processes update requests from an input file in the format defined in the file **cs.compat**. Update requests result in a call to the **updatep** internal command, **inuupdt**.

This command is run by **installc** only when the **cs.compat** file contains update records. If **cs.compat** does not exist or contains dbos, uplevel, or unknown records, **installc** will not be called.

The **-r** *path* parameter is used to pass the path that **updatec** must use to gain access to the server's **/usr/lpp.update** directory. It represents a local path where **chngstate** has mounted the / (root) directory of the server.

The *file* is an input file in **cs.compat** format. The file contains specific update requests required to make this client fully compatible with a specific code server.

Files

/etc/codeserve/cs.compat	Contains information on client and server compatibility.
/etc/codeserve/serverattach	Contains code service attribute information.
/etc/rc	Performs normal startup initialization.
/etc/rc.standalone	Initializes stand-alone system.

/etc/rc.actvsrvc	Initializes active-service client.
/usr/lpp.install	Server directory containing backup format files required to do installations.
/usr/lpp.update	Server directory containing backup format files required to do updates.

Related Information

The following commands: "**chkcomp**" on page 158, "**installp**" on page 529, "**rc**" on page 806, "**updatep**" on page 1122.

The **/etc/codeserve/serverattach** file in *AIX Operating System Technical Reference.*

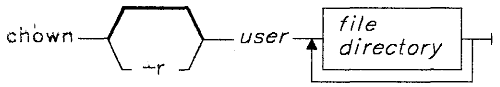
The discussion of code service and the **/etc/codeserve/cs.compat** file in *Managing the AIX Operating System.*

chown

Purpose

Changes the owner of files or directories.

Syntax



OL805095

Description

The **chown** command changes the owner of the specified *files* or *directories* to *username* or *userID*. The group associated with the file or directory is not affected.

Note: If you give ownership of a file or directory to another user, you cannot regain ownership unless you have superuser authority.

If the file or directory resides in a Distributed Services remote virtual file system, the translated user ID is used.

Flag

-r	Causes the untranslated user ID to be used. Applies only to files or directories that reside in a Distributed Services remote virtual file system.

Example

```
chown  jim  program.c
```

The user access permissions for `program.c` now apply to `jim`. As the owner, `jim` can use **chmod** to permit or deny the other users access to `program.c`. See "**chmod**" on page 160 for details.

Files

/etc/passwd	File that contains user IDs.

Related Information

The following command: "**passwd**" on page 735.

The **chown** and **chownx** system calls and the **passwd** file in *AIX Operating System Technical Reference.*

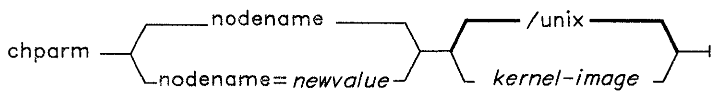
"Distributed Services **id** Translation" in *Managing the AIX Operating System.*

chparm

Purpose

Changes or examines system parameters.

Syntax



OL805093

Description

The **chparm** command lets you change a system parameter or look at its current setting. Currently, only the **nodename** parameter may be examined or changed. The name assigned cannot be longer than eight characters. If you do not assign a *newvalue*, **chparm** writes the current value of **nodename** to standard output. The default *kernel-image* is **/unix**.

Changes do not affect the running system. You must restart the system for the change to become effective.

Examples

1. To display the **nodename** of your system:

   ```
   chparm  nodename
   ```

   This displays the **nodename** of **/unix**, which is a file containing the kernel of the AIX Operating System. This file is loaded and run when you start up the computer.

2. To change the **nodename** of a system:

   ```
   chparm  nodename=COMP-CTR  /unix.compctr
   ```

   This changes the **nodename** of `/unix.compctr` to `COMP-CTR`. `/unix.compctr` is a file that contains an alternate version of the operating system kernel. The change does not affect the running system, even if you change the **/unix** kernel.

chroot

Purpose

Changes the root directory of a command.

Syntax

```
chroot — directory — command —|
```

OL805094

Description

Warning: If special files in the new root have different major and minor device numbers than they have in the real root, it is possible to overwrite the file system.

The **chroot** command can be used only by a user operating with superuser authority (see "**su**" on page 1026). If you have superuser authority, the **chroot** command changes the root directory to the specified *directory* when executing *command.* The first / (slash) in any path name changes to *directory* for the specified *command* and any of its children.

Notice that:

chroot *directory command* > *file*

creates the *file.* relative to the original root, not the new one.

The *directory* path name is always relative to the current root. Even if a **chroot** is in effect, *directory* is relative to the current root of the running process.

Several programs may not operate properly after **chroot** has been run. For example, the command **ls -l** will fail to give user and group names if the current root location makes **/etc/passwd** beyond reach. In addition, utilities that depend on description files produced by the **ctab** command (see page 257) may fail altogether if these files are also not in the new root file system. It is your responsibility to ensure that all vital data files are present in the new root file system and that the path names accessing such files are changed as necessary.

Examples

1. To run a subshell with another file system as the root:

   ```
   chroot  /diskette0  /bin/sh
   ```

   This makes the directory name / refer to **/diskette0** for the duration of the command **/bin/sh**. It also makes the original root file system inaccessible. The file system on **/diskette0** must contain the standard directories of a root file system. In particular, the shell will look for commands in **/bin** and **/usr/bin** on the **/diskette0** file system.

   Running the command **/bin/sh** creates a subshell, which runs as a separate process from your original shell. Press END OF FILE **(Ctrl-D)** to end the subshell 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 (/).

2. To run a command in another root file system and save the output:

   ```
   chroot  /diskette0  /bin/cc  -E  /u/bob/prog.c  >prep.out
   ```

   This runs the **/bin/cc** command with / referring to **/diskette0**. It saves the output in the file `prep.out`, which is in the original root file system.

   This runs the C language preprocessor (`/bin/cc -E`) on the file `/diskette0/u/bob/prog.c`, reading **#include** files from **/diskette0/usr/include**, and putting the preprocessed text in `prep.out` on the primary root file system.

Related Information

The following commands: "**cc**" on page 140, "**cpp**" on page 210, and "**sh**" on page 913.

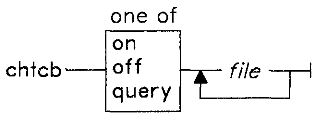
The **chdir** and **chroot** system calls in *AIX Operating System Technical Reference.*

chtcb

Purpose

Sets or queries the tcb attribute of a file.

Syntax



A5AC5021

Description

The **chtcb** command sets or queries the **tcb** attribute of the specified files. The **tcb** attribute of a file should be on in that file is in the trusted computing base; otherwise, is should be off.

A file must be in the trusted computing base if it is to be executed from the trusted shell (**tsh**).

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Related Information

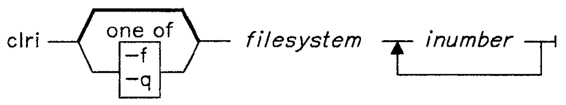
The following commands: "**sysck**" on page 1031 and "**tsh**" on page 1100.

clri

Purpose

Clears the specified i-node.

Syntax



OL805097

Description

Warning: Use this command only in emergencies and with extreme care.

The **clri** command is used to clear i-node entries for files that do not appear in a directory. In general, you do not need to use this program because **fsck** can deal with most file system inconsistencies.

Always run **fsck** on a file system after you have used **clri** on it, because it may create dangling directory references or missing blocks. These can be fixed if they are attended to promptly. Do not run the system when the file system has dangling directory references or a bad free list.

The **clri** command zeros over the flags′ word of the i-node, thus freeing it for reallocation. The *inumber* parameter specifies the i-node and *filesystem* specifies the file system it is on. *inumber* should be a decimal number, while *filesystem* can be either the name of the device on which the file system resides or the name by which it is normally mounted.

If you use **clri** to remove an i-node that does appear in a directory, you should track down and remove all of these entries. Otherwise, when the i-node is reallocated to some new file, the old entry will still point to that file. At that point removing the old entry destroys the new file and the new entry again points to an unallocated i-node.

By default, the **clri** command displays some information about the file and asks for confirmation before it destroys the file. If you enter a `y` or `yes`, the file is destroyed.

Since **clri** only zeros the flags′ word of the i-node, if you destroy the wrong file, you can recover the file by using the **fsdb** command to restore the flags′ word.

Note: If the file is open, **clri** is likely to be ineffective. For this reason, you should run **clri** only on an unmounted file system.

Flags

-f	Destroys the file without confirmation, but writes a description of the file.
-q	Destroys the file without confirming or writing a description of the file.

Example

To clear i-nodes 170 and 368 of the file system **/diskette0** and then clean up the file system:

```
clri  /diskette0  170  368
fsck  /diskette0
```

Related Information

The following commands: "**fsck**, **dfsck**" on page 445 and "**fsdb**" on page 450.

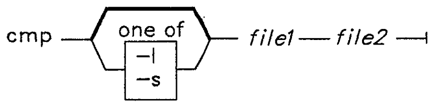
The **fs** file in *AIX Operating System Technical Reference.*

cmp

Purpose

Compares two files.

Syntax



OL805157

Description

The **cmp** command compares *file1* and *file2* and writes the results to standard output. If you specify a - (minus) for *file1*, **cmp** reads standard input. Under default conditions, **cmp** displays nothing if the files are the same. If they differ, **cmp** displays the byte and line number at which the first difference occurs. If one file is an initial subsequence of the other (that is, if **cmp** reads an end-of-file character in one file before finding any differences), **cmp** notes this. Normally, you use **cmp** to compare non-text files and the **diff** command to compare text files.

Flags

-l	Displays, for each difference, the byte number in decimal and the differing bytes in octal.
-s	Returns only an exit value. (0 indicates identical files; 1 indicates different files; 2 indicates inaccessible file or a missing argument)

Examples

1. To determine whether two files are identical:

   ```
   cmp  prog.o.bak  prog.o
   ```

   This compares `prog.o.bak` and `prog.o`. If the files are identical, then a message is not displayed. If the files differ, then the location of the first difference is displayed.

For instance:

```
prog.o.bak  prog.o  differ:  char  5,  line  1
```

If the message `cmp: EOF on prog.o.bak` is displayed, then the first part of `prog.o` is identical to `prog.o.bak`, but there is additional data in `prog.o`.

2. To display each pair of bytes that differ:

   ```
   cmp  -l  prog.o.bak  prog.o
   ```

   This compares the files, and then displays the byte number (in decimal) and the differing bytes (in octal) for each difference. For example, if the fifth byte is octal 101 in `prog.o.bak` and 141 in `prog.o`, then `cmp` displays:

   ```
        5  101  141
   ```

3. To compare two files without writing any messages:

   ```
   cmp  -s  prog.c.bak  prog.c
   ```

   This gives an exit value of 0 if the files are identical, 1 if different, or 2 if an error occurs. This form of the command is normally used in shell procedures. For example:

   ```
   if  cmp  -s  prog.c.bak  prog.c
   then
         echo  No  change
   fi
   ```

   This partial shell procedure displays `No change` if the two files are identical. See page 930 for details about the **if** command.

Related Information

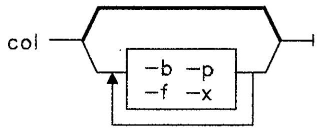
The following commands: "**comm**" on page 183, "**diff**" on page 320, and "**sh**" on page 913.

col

Purpose

Processes text having reverse linefeeds and forward/reverse half-linefeeds for output to standard output.

Syntax



OL805173

Description

The **col** command reads from standard input and writes to standard output. It performs the line overlays implied by reverse line feeds (ASCII ESC-7), and by forward and reverse half-line feeds (ASCII ESC-9 and ASCII ESC-8). **col** is particularly useful for filtering multicolumn output made by the **nroff .rt** command and output from the **tbl** command. The input format accepted by **col** matches the output format produced by **nroff -T37** or by **nroff -Tlp**. Use **-T37** and the **col -f** flag if the output is being sent to a device that can interpret half-line motions; use **-Tlp** otherwise.

The **col** command assumes that the ASCII control characters SO (\017) and SI (\016) begin and end text in an alternate character set. **col** remembers the character set each input character belongs to, and on output generates SI and SO characters as appropriate to ensure that each character is printed in the correct character set.

On input, **col** accepts only the control characters for Space, Backspace, Tab, Return, the new-line character, SI, SO, VT, and ESC-7, 8, or 9. VT (\013) is an alternate form of full reverse line feed included for compatibility with some earlier programs of this type. **col** ignores all other non-printing characters.

Notes:

1. The maximum number of lines that can be backed up is 128.
2. Up to 800 characters, including backspaces, are allowed on a line.
3. Local vertical motions that would result in backing up over the first line are ignored. As a result, the first line must not contain any superscripts.

Flags

-b Assumes that the output device in use is not capable of backspacing. In this case, if two or more characters are to appear in the same position, only the last one read appears in the output.

-f Suppresses the default treatment of half-line motions in the input. Normally, **col** does not emit half-line motions on output, although it does accept them in its input. With this flag, output may contain forward half-line feeds (ESC-9) but not reverse line feeds (ESC-7 or ESC-8).

-p Displays unknown escape sequences as characters, subject to overprinting from reverse line motions. Normally, **col** ignores them. You should be fully aware of the textual position of escape sequences before you use this flag.

-x Suppresses changing the white space to tabs. Without this flag, **col** converts white space to tabs wherever doing so might shorten printing time.

Related Information

The following commands: "**nroff, troff**" on page 709 and "**tbl**" on page 1053.

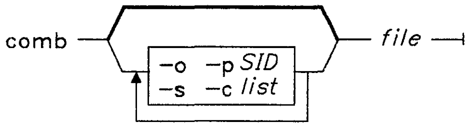
The discussion of **col** in *Text Formatting Guide.*

comb

Purpose

Combines SCCS deltas.

Syntax



OL805098

Description

The **comb** command writes to standard output a shell procedure that can combine the specified deltas (***SID***s) or all deltas into one delta. You may reduce the size of your SCCS file by running the resulting procedure on the file. You can see how much the file will be reduced by running **comb** with the **-s** flag. If you specify a directory in place of *file*, **comb** performs the requested actions on all SCCS files (that is, those with file names with the **s.** prefix). If you specify a **-** (minus) in place of *file*, **comb** reads standard input and interprets each line as the name of an SCCS file. **comb** continues to take input until it reads END OF FILE (**Ctrl-D**).

If you do not specify any flags, **comb** preserves only leaf deltas and the minimal number of ancestors needed to preserve the tree (see "**delta**" on page 310).

Note: The **comb** command may rearrange the shape of the tree deltas. It may not save any space; in fact, it is possible for the reconstructed file to actually be larger than the original.

Flags

Each flag or group of flags applies independently to each named file.

-c*list* Specifies a list of deltas (*SID*s) that the shell procedure will preserve (see **get -i** *list* for the SID list format on page 482). The procedure will combine all other deltas.

-o	Accesses the reconstructed file at the release of the delta to be created for each **get -e** generated; otherwise accesses the reconstructed file at the most recent ancestor. Using the **-o** flag may decrease the size of the reconstructed SCCS file. It may also alter the shape of the delta tree of the original file.
-p*SID*	Specifies the *SID* of the oldest delta for the resulting procedure to preserve. All older deltas are combined in the reconstructed file.
-s	Causes **comb** to generate a shell procedure that produces a report for each file giving: the file name, size (in blocks) after combining, original size (also in blocks), and percentage change computed by the formula: 100 * (original - combined) / original You should run **comb** using this flag and run its procedure before combining SCCS files in order to judge how much space will actually be saved by the combining process.

Files

s.COMB	The name of the reconstructed SCCS file.
comb*	Temporary files.

Related Information

The following commands: "**admin**" on page 41, "**delta**" on page 310, "**get**" on page 477, "**help**" on page 513, and "**prs**" on page 781.

The **sccsfile** file in *AIX Operating System Technical Reference.*

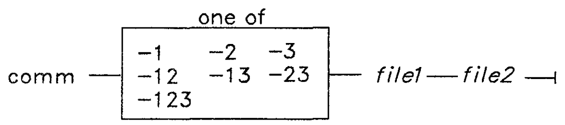
The discussion of SCCS in *AIX Operating System Programming Tools and Interfaces.*

comm

Purpose

Selects or rejects lines common to two sorted files.

Syntax



OL805099

Description

The **comm** command reads *file1* and *file2* and writes, by default, a three-column output to standard output. The columns consist of:

- Lines that are only in *file1*
- Lines that are only in *file2*
- Lines that are in both *file1* and *file2.*

If you specify - (minus) for one of the file names, **comm** reads standard input. Both *file1* and *file2* should be sorted according to the collating sequence specified by the environment variable **NLCTAB** (see "**ctab**" on page 257 and "**sort**" on page 958),

Flags

-1	Suppresses the display of the first column (lines in *file1*).
-2	Suppresses the display of the second column (lines in *file2*).
-3	Suppresses the display of the third column (lines common to *file1* and *file2*).

Note: Specifying **-123** does nothing (a noop).

Examples

1. To display the lines unique to each file and common to both:

```
comm  things.to.do  things.done
```

If the files `things.to.do` and `things.done` contain:

things.to.do	**things.done**
buy soap groceries luncheon meeting at 3 system update tech. review	2nd revision interview luncheon system update tech. review weekly report

then **comm** displays:

```
       2nd revision
buy soap
groceries
       interview
               luncheon
meeting at 3
               system update
               tech. review
       weekly report
```

The first column contains the lines found only in `things.to.do`. The second column, indented with a tab character, lists the lines found only in `things.done`. The third column, indented with two tabs, lists the lines common to both.

2. To display the lines that appear in only one file:

```
comm  -23  things.to.do things.done
```

This suppresses the second and third columns of the **comm** listing. If the files are the same as in Example 1, then the following is displayed:

```
buy soap
groceries
meeting at 3
```

Related Information

The following commands: "**cmp**" on page 177, "**ctab**" on page 257, "**diff**" on page 320, "**sdiff**" on page 883, "**sort**" on page 958, and "**uniq**" on page 1118.

The **environment** miscellaneous facility in *AIX Operating System Technical Reference.*

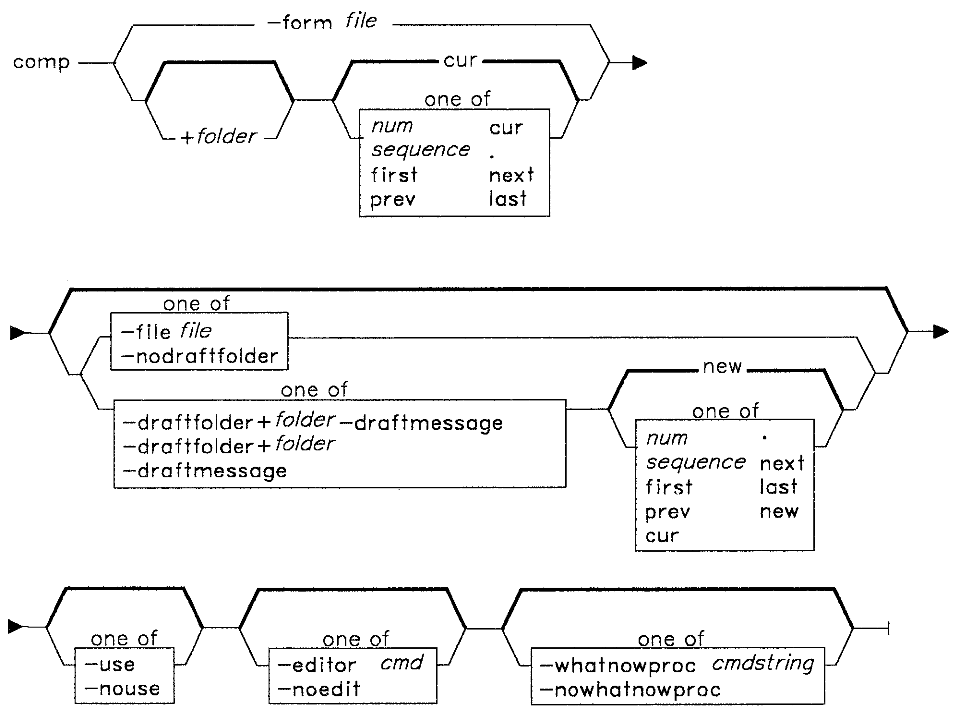
"Overview of International Character Support" in *Managing the AIX Operating System.*

comp

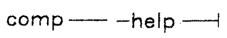
Purpose

Composes a message.

Syntax



AJ2FL222



AJ2FL167

Description

The **comp** command is used to create and modify messages. **comp** is part of the MH (Message Handling) package and can be used with other MH and AIX commands.

By default, **comp** copies a message form to a new draft message and invokes an editor. You can then fill in the message header fields **To:** and **Subject:**, fill in or delete the other header fields (such as **cc:** and **Bcc:**), and add the body of the message. When you exit the editor, the **comp** command invokes the MH command **whatnow**. You can specify any of the **whatnow** subcommands, or you can press **Enter** to see a list of the subcommands. These subcommands enable you to continue composing the message, direct the disposition of the message, or end the processing of the **comp** command. See "**whatnow**" on page 1215 for a description of the subcommands.

You can specify the ***form***, or format, of the message by using the **-form** flag or the +*folder* flag. If you do not specify one of these flags, **comp** uses your default message format located in the file *user_mh_directory*/**components**. If this file does not exist, **comp** uses the system default message format located in **/usr/lib/mh/components**.

You can compose a new message, or you can specify the **-use** flag and continue composing an existing message. The **-file**, **-draftfolder**, and **-draftmessage** flags enable you to specify the new or existing message that you want to compose.

Note: The line of dashes or a blank line must be left between the header and the body of the message for the message to be identified when it is sent.

Flags

-draftfolder +*folder*
: Places the draft message in the specified folder. If you do not specify this flag, **comp** selects a default draft folder according to the information supplied in the MH profiles. You can define a default draft folder in **$HOME/.mh_profile**. If **-draftfolder** +*folder* is followed by *msg*, *msg* represents the **-draftmessage** attribute.

-draftmessage *msg*
: Specifies the draft message. You can specify one of the following message references as *msg*:

num	*sequence*	**first**
prev	**cur**	**.**
next	**last**	**new**

If the **-use** flag is specified, the default draft message is **cur**. Otherwise, the default draft message is **new**.

-editor *cmd*	Specifies that *cmd* is the initial editor for composing the message. If you do not specify this flag, **comp** selects a default editor or suppresses the initial edit, according to the information supplied in the MH profiles. You can define a default initial editor in **$HOME/.mh_profile**.
-file *file*	Places the draft message in the specified file. If you do not specify the absolute path name for *file*, **comp** places *file* in *user_mh_directory*. If *file* exists, **comp** prompts your for the disposition of the draft.
+*folder msg*	Uses the form of the specified message in the specified folder. You can specify one of the following message references as *msg*:

num	*sequence*	**first**
prev	**cur**	
next	**last**	

The default message is the current message in the current folder.

-form *file*	Uses the form contained in the specified file. **comp** treats each line in *file* as a format string.
-help	Displays help information for the command.
-nodraftfolder	Places the draft in the file *user_mh_directory*/**draft**.
-noedit	Suppresses the initial edit.
-nouse	Creates a new message.
-nowhatnowproc	Does not invoke a program that guides you through the composing tasks. The **-nowhatnowproc** flag also prevents any edit from occurring.
-use	Continues composing an existing draft of a message.
-whatnowproc *cmdstring*	Invokes *cmdstring* as the program to guide you through the composing tasks. See "**whatnow**" on page 1215 for information about the default **whatnow** program and its subcommands.

Note: If you specify `whatnow` for *cmdstring*, **comp** invokes an internal **whatnow** procedure rather than a program with the file name **whatnow**.

Profile Entries

Draft-Folder:	Sets your default folder for drafts.
Editor:	Sets your default initial editor.
fileproc:	Specifies the program used to refile messages.
Msg-Protect:	Sets the protection level for your new message files.
Path:	Specifies your *user_mh_directory*.
whatnowproc:	Specifies the program used to prompt `What now?` questions.

Files

/usr/lib/mh/components	The system default message form.
user_mh_directory/components	The user's default message form. (If it exists, it overrides the system default message form.)
$HOME/.mh_profile	The MH user profile.
user_mh_directory/draft	The draft file.

Related Information

Other MH commands: "**ali**" on page 48, "**dist**" on page 336, "**forw**" on page 438, "**prompter**" on page 778, "**repl**" on page 821, "**refile**" on page 817, "**send**" on page 893, "**whatnow**" on page 1215, "**whom**" on page 1222.

The **mh-alias**, **mh-format**, and **mh-profile** files in *AIX Operating System Technical Reference.*

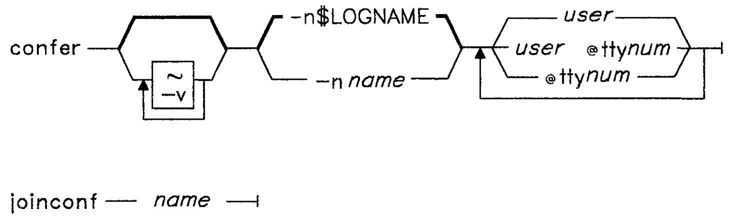
The "Overview of the Message Handling Package" in *Managing the AIX Operating System.*

confer

Purpose

Provides an online conferencing system.

Syntax



OL805174

Description

The **confer** command sets up an online, written conference among logged-in users on your local node. You start a conference by running the **confer** command, specifying the *user*s and/or work stations (@**tty***num*) that are part of the conference. If the users you specify are logged in and their work stations are writable, they are requested to join the conference by using the **joinconf** command. The other conferees are informed as each user joins the conference.

Once you join a conference, everything you enter at your work station displays at all other work stations that are part of the conference. This display continues until you press **Ctrl-D** to end your own active participation or until you **excuse** a conference participant, thus stopping the display of your contributions at his work station. (See page 190.)

To prevent the confusion that can be caused by several conferees typing at the same time, users should follow some agreed on protocol. The following is one recommended protocol:

- In order to take the floor, a user presses the **Enter** key before entering his contribution. This notifies other participants that he has the floor because his name displays in brackets at their respective work stations.
- A user is presumed to have the floor until he relinquishes it by entering a blank line.
- If two or more users try to claim the floor at the same time, the last person to do so (the one whose name appears last), is assumed to have the floor. The others should immediately relinquish the floor by typing single blank lines.

The **confer** command gives each conference a unique name, normally the name of the conference leader, with additional letters added to it, if necessary. The conference leader can override this default by specifying the **-n** flag.

A user who is logged in to more than one work station is normally written to on all of them, unless the conference leader specifies one of the work stations with the **@tty***num* flag when he invokes **confer**.

A conferee ends his active participation by pressing **Ctrl-D**. This action causes his name and the word BYE to display at the work stations of the other conference participants. However, the contributions of the other participants will continue to display at his work station until the other participants each **excuse** him.

You can run shell commands from within a conference by simply prefixing them with a ¦ (vertical bar) or an ! (exclamation point). Using the exclamation point causes the command to run in the normal fashion; the output displays only at the work station that runs it. Using the vertical bar, however, causes the command and all of its standard output and standard error output to become part of the conference, visible to all conferees.

Three subcommands are run directly by **confer** and **joinconf**. These are:

!excuse *name* . . .	Excuses the specified conferees from the conference. No further conference material displays at these work stations.
!~	Makes all contributions from the user who issues it off the record until he issues the **!~~** subcommand.
!~~	Cancels a preceding **!~**, placing the user's remarks back on the record.

Unless the conference leader makes a conference off the record by specifying the ~ flag, **confer** makes a transcript of all conference proceedings. When a participant leaves the conference, he is asked whether he wants a transcript. If he does, he is mailed a copy when the conference concludes. Any participant can make a comment off the record in a conference that is otherwise ***on the record*** by beginning the line with a ~ (tilde).

Conference contributions are normally transmitted one line at a time. If the conference leader specifies the **-v** flag, transmission occurs one character at a time. As this mode of transmission sends all user typing errors and hesitations and imposes a considerably larger load on the system, its use is strongly discouraged.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Flags

-n*name*	Assigns *name* to the conference transcript. The conference name is used by those joining the conference so that they get into the right one. The name of the user who starts the conference is the default conference name.
-v	Transmits conference messages one character at a time.
~	Sets up the conference ***off the record***, that is, no transcript of the proceedings is recorded.
@tty*num*	Specifies a particular work station for a conferee, if a *user* is also specified (for example, `tty1`). This is useful if a conferee is logged in to more than one work station. If no *user* is specified, this flag invites any user logged in to the specified work station to participate.

Examples

1. To start a conference with `steve` and `rachel`:

   ```
   confer  steve  rachel
   ```

   Running the **confer** command makes you the conference leader, so your login name is also the name of the conference. **confer** sends `steve` and `rachel` a message inviting them to join your conference and giving them the conference name.

2. To specify work stations that may join the conference:

   ```
   confer    steve@tty5    rachel    @tty10
   ```

   Suppose that `steve` is logged in at the work stations **tty3**, **tty4**, and **tty5**, and that `rachel` is logged in at **tty7** and **tty8**. This command invites `steve` to join the conference at work station **tty5** only, invites `rachel` to join at either work station she is using or at both, and invites whoever is logged in at **tty10** to join.

3. To join a conference named `paula`:

   ```
   joinconf  paula
   ```

   Now the text you type becomes part of the dialog: prefixed with your name, displayed at each participant's work station, and recorded in the transcript of the conference.

4. Suppose that you start a conference by entering the command given in Example 2, and the person using **tty10** decides not to join the conference. If you do nothing, this person also sees the dialog, even though not participating in it. To prevent this from happening, each person that has joined the conference must enter:

   ```
   !excuse  @tty10
   ```

Similarly, if `rachel` decides to join the conference from **tty7**, the discussion is also displayed at her other work station, **tty8**, unless everyone enters:

```
!excuse  rachel@tty8
```

rachel should enter this, too, but only at **tty7**, the work station she is using for the conference.

5. To make a single-line statement off the record:

```
~Coffee and donuts at my place.
```

confer displays lines beginning with ~ (tilde) at participants' work stations, but does not include them in the record of the conference.

To make a multiple-line statement:

```
!~
Everyone is invited
to my place after the conference
for coffee and donuts.
!~~
```

6. To run a shell command privately, without leaving the conference:

```
!li
```

This lists the current directory without including the **li** command or its output in the conference.

7. To include the output of a shell command in the discussion:

```
!cat  notes.conf
```

This lists the contents of the file `notes.conf` at each participant's work station, and includes it in the conference record.

8. To send command output to others, off the record:

```
!~
!cat notes.conf
!~~
```

9. To leave the conference, press **Ctrl-D**. If your user name is `paula`, then after you press **Ctrl-D**, the message: `[paula] BYE` is sent to the other participants. The rest of the discussion continues to appear at your work station until each of the other participants enters:

```
!excuse  paula
```

Files

/etc/utmp	List of logged-in users.
/dev/tty??	Work station names.
/tmp/*.cnf	User transcript files.
/tmp/*.ln?	Links to main conference file.
/tmp/*.mls	Transcript mailing list.

Related Information

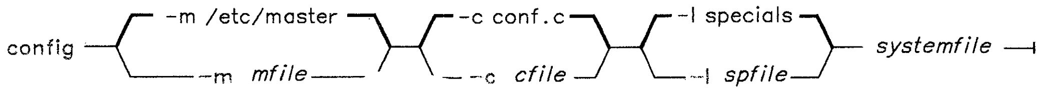
The following command: "**write**" on page 1225.

config

Purpose

Extracts configuration information from configuration files.

Syntax



OL805416

Description

The **config** program reads the AIX master and system configuration files (by default **/etc/master** and the specified *systemfile*). It writes a C Language configuration file and a special file list (by default **conf.c** and **specials**). The special file list is a list of the **mknod**, **chown**, and **chmod** commands that the shell runs to define the necessary special files. The return code is the number of errors encountered.

The C Language configuration file can then be compiled and linked with other kernel object files to produce a new kernel. Normally, when you want to reconfigure the kernel, you should run the **make** command with the **Makefile** supplied in the **/usr/sys** directory. This runs **config** and then builds a new kernel. For a discussion of reconfiguring the kernel, see *Managing the AIX Operating System.*

Flags

-c *cfile*	Writes the C configuration file to *cfile* instead of to **conf.c**.
-l *spfile*	Writes the special file list commands to *spfile* instead of to **specials**.
-m *mfile*	Reads *mfile* instead of **/etc/master**.

Files

/etc/master	Default master configuration file.
/etc/system	A system configuration file.
conf.c	Default C configuration file.
specials	Default special file list.

Related Information

The following commands: "**make**" on page 625 and "**vrmconfig**" on page 1206.

The **master** and **system** files in *AIX Operating System Technical Reference.*

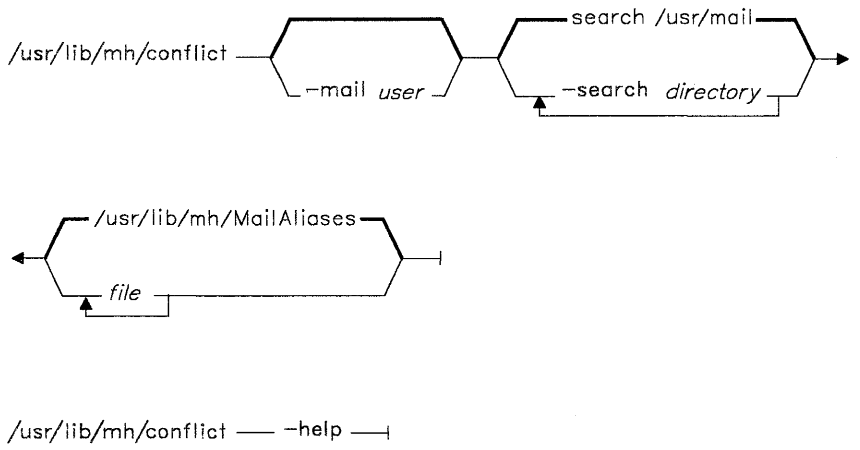
The discussion of **config** in *Managing the AIX Operating System.*

conflict

Purpose

Searches for alias and password conflicts.

Syntax



AJ2FL225

Description

The **conflict** command is used to find conflicts in aliases and to find invalid mail drops. **conflict** is not designed to be run directly by the user; it is designed to be called by **cron** and other programs used for system accounting. **conflict** is a system administrator command that is usually invoked by its full path name. The **conflict** command is part of the MH (Message Handling) package.

The **conflict** command searches all specified alias files for duplicate alias names that do not resolve to the same address. By default, **conflict** searches **/usr/lib/mh/MailAliases**. **conflict** also searches all specified mail drop directories for mailbox files with names that do not correspond to valid users defined in **/etc/passwd**.

The **conflict** lists its output on the display, unless you specify the **-mail** flag. **-mail** causes **conflict** to mail its output to the specified user.

Flags

-help	Displays help information for the command.
-mail *user*	Sends the results of the **conflict** command to the specified user.
-search *directory*	Searches the indicated directories for invalid mailboxes. You can specify any number of **-search** flags. The default mailbox directory is **/usr/mail**.

Files

/usr/lib/mh/mtstailor	The MH tailor file.
/etc/passwd	List of users.
/etc/group	List of groups.
/usr/mail/$USER	The location of the mail drop.

Related Information

Other MH commands: "**ali**" on page 48, "**whom**" on page 1222.

The **mh-alias**, **mh-mail**, and **mh-profile** files in *AIX Operating System Technical Reference.*

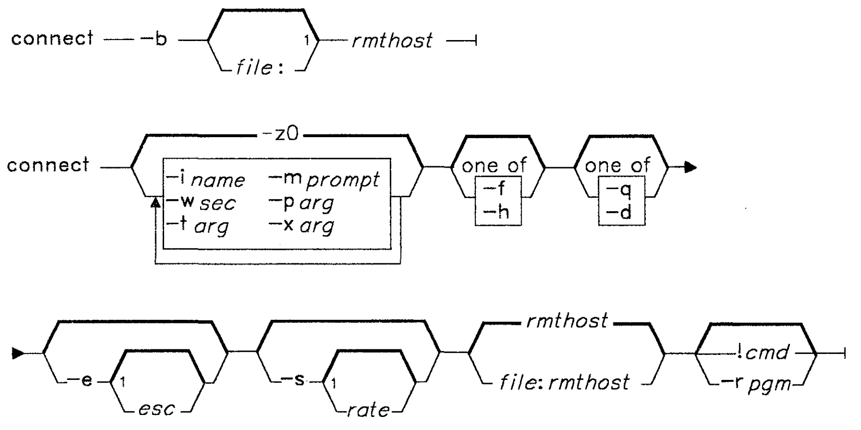
The "Overview of the Message Handling Package" in *Managing the AIX Operating System.*

connect

Purpose

Establishes a connection to a remote system.

Syntax



OL805388

Description

The **connect** command lets you establish a connection to a remote host. **connect** runs in two parts. The first part makes the connection with the remote system specified by *rmthost*. The second part is a program called the **talker**. It runs automatically and exchanges data with the *rmthost*. For more information about the **talker** program, see "**connect**" in *AIX Operating System Technical Reference*. Any flags that you specify are passed directly to the **talker** without interpretation. The default **talker** for asynchronous links is **atalk**.

The **connect** command uses a system-wide control file, **connect.con**, located in **/usr/lib/INnet**. You can specify an additional control file, *file:rmthost*. If you do not specify an additional file, **connect** searches **$HOME/bin** for a **connect.con** file. Information needed to complete the connection is found in one of these files.

Attributes needed to complete the connection are taken from the control file or from the command line assignment *var=val*. For a description of the parameters, see "**connect**" in *AIX Operating System Technical Reference*.

When **atalk** detects an escape sequence in the input, it places the work station in its former mode of operation and prompts you with the local prompt. You can then use the flags that follow. Once the flag has run, **atalk** returns to its former mode.

The **connect** command does not limit access to the phone system to control dialing based on the number to be called.

Warning: The **connect** command lets you set up and maintain connections through a wide variety of communications devices. It interacts with you through the file **connect.con** which is free-format. Problems with the format of this file may cause unpredictable results.

Flags

Note: There are no spaces between the flags and the associated parameters.

-b	Sends a break to the port. This is done by lowering the transmission speed to 75 bps and transmitting an ASCII NULL on the port. If the speed is too low, less than 100 bps, this may not work.
-d **-q**	Closes, quits (**q**) or disconnects (**d**) the port. Note that this does ***not*** end your job or session at the remote site. After closing the port, **connect** exits.
-e[*esc*]	Sets the escape sequence to the character string *esc*. If you do not specify *esc*, **connect** displays escape sequence. It takes the default escape sequence from the environment variable **CONESC**, if defined, or else sets it to: **Ctrl-V␣Ctrl-M**
-f **-h**	Enables (**-h**) or disables (**-f**) local echoing.
-i*name*	Writes file *name* to the port. **Warning:** If you are connected to the remote host by RS-232 lines, data from the file may be lost if the remote host cannot keep up with the input. Normally, this flag is used to transfer a small file from the local site to the remote site. File transmission must be ended manually by pressing **Ctrl-D**.

For example:

```
cat > newfile
[escape sequence]
LOCAL: ifred
.
.
.
Ctrl-D
```

-m*prompt* Set the local prompt to the *prompt* character string. **connect** displays this prompt when it recognizes the escape sequence. By default, it sets the prompt to the value of the environment variable **CONPMT**. If this variable is not set, it uses the string **LOCAL:**.

-p*arg* Sets parity as specified by *arg*, where *arg* is one of the following characters: **o** (odd), **e** (even), **7** (both even and odd), or **8** (eight data bits).

-r*pgm* Runs the network program *pgm*. Anything following *pgm* on the command line is passed to *pgm* as an argument, along with the additional arguments **-i3 -o3**. The port set up as file descriptor 3. The program is run as a child process.

-s*rate* Sets the transmission speed to *rate*, which is one of the following: 0, 50, 75, 110, 134, 150, 200, 300, 600, 1200, 1800, 2400, 4800, 9600, exta, extb (0 effectively turns off the port). If you do not specify *rate*, current transmission speed displays.

-t*arg* Enables or disables transcripts. If *arg* is any character string other than a minus or plus sign, the transcript function is enabled with the specified file *arg* as transcript. When you use an existing file as a transcript file, new data is added to its end. Use **t-** to disable the transcript function, and **t+** to enable the transcript to the previous transcript file (no default).

-w*sec* Sets the inter-line delay of the include function to cause a delay interval of the specified seconds between each line written to the port. The default value is 0.

-x*arg* Enables or disables input or output flow control. If the input flow control is enabled, **CTRL-S** and **CTRL-Q** are automatically sent to the remote host to control the rate at which it transmits data. If the output flow control is enabled, **CTRL-S** and **CTRL-Q** are automatically honored if received from the host. This is useful when using the **include** command. **xi+** enables input flow control. **xi−** disables input flow control. **xi** displays the current state. For control of output flow control, replace **xi** with **xo**. See the discussion of IXON and IXOFF in the **termio** file in *AIX Operating System Technical Reference*.

!*cmd* Runs the AIX command *cmd*. Anything that follows ! (exclamation point), including arguments to *cmd*, is passed to the local shell to be run by the **system** system call. In particular, all I/O redirection and piping works.

Files

/usr/lib/INnet/connect.con	System-wide connection control file.
$HOME/bin/connect.con	Private connection control file.
/usr/lib/INnet/dialers/*	System-wide dialer programs.
$HOME/bin/*	Private dialer programs.
/usr/lib/INnet/atalk	Default talker program, asynchronous lines.
/etc/sites	Network sites file.
/etc/locks	Directory for locks on ports (devices) used for logins and out-going connections.

Related Information

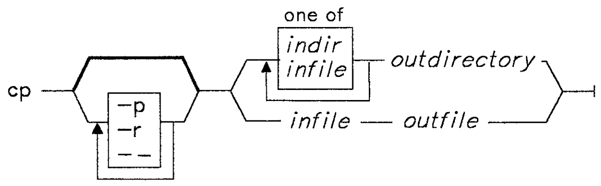
The **system** and **exec** system calls, the **connect** subroutine, and the **termio** special facility in *AIX Operating System Technical Reference.*

cp

Purpose

Copies files.

Syntax



OL805100

Description

The **cp** (copy) command copies a source file or the files in a source directory to a target file or directory. If your output is to a directory, then the files are copied to that directory with the same file name. If either *infile* or *outfile* is a symbolic link, the link is followed when **cp** is performed. An error message is displayed if the link cannot be followed.

You can also copy special device files. If the file is a named pipe, the data in the pipe is copied into a regular file. If the file is a device, the file is read until the end of file and that data is copied into a regular file.

Notes:

1. Do not name *outfile* as one of the input files.
2. If you specify a directory for the *outfile*, the directory must already exist.
3. If the *infile* contains subdirectories and the subdirectories do not exist, the system creates them.

Flags

-p Preserves the modification times and modes of the *infile* for the copy.

-r Copies each subtree rooted at the *infile* (recursive copy). If the *infile* is a directory, then the *outfile* must be a directory.

-- Indicates that the arguments following this flag are to be interpreted as file names. This null flag allows the specification of file names that start with a minus.

Examples

1. To make another copy of a file in the current directory:

   ```
   cp  prog.c  prog.bak
   ```

   This copies `prog.c` to `prog.bak`. If the file `prog.bak` does not already exist, then **cp** creates it. If it does exist, then **cp** replaces it with a copy of `prog.c`.

2. To copy a file to another directory:

   ```
   cp  jones  /u/nick/clients
   ```

   This copies `jones` to `/u/nick/clients/jones`.

3. To copy a file to a new file and preserve the modification date and time:

   ```
   cp -p smith smith.jr
   ```

   This copies `smith` to `smith.jr`. Instead of creating the file with the current date and time stamp, the system gives `smith.jr` the same date and time as `smith`.

4. To copy all the files in a directory to a new directory:

   ```
   cp /u/nick/clients/* /u/nick/customers
   ```

   This copies the files and directories in the directory `clients` to the directory `customers`.

5. To copy a directory, its files and its subdirectories to another directory:

   ```
   cp -r /u/nick/clients /u/nick/customers
   ```

   This copies the directory `clients`, its files, its subdirectories, and the files in the subdirectories to the directory `customers`.

6. To copy a specific set of files to another directory:

   ```
   cp  jones lewis smith  /u/nick/clients
   ```

   This copies `jones`, `lewis`, and `smith` to `/u/nick/clients`.

7. To use pattern-matching characters to copy files:

   ```
   cp  programs/*.c  .
   ```

   This copies the files in directory `programs` that end with `.c` to the current directory (.). You must type a space between the `c` and the final period.

Related Information

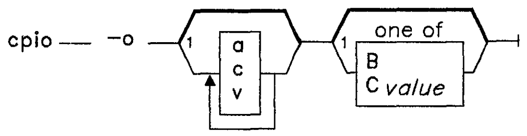
The following commands: "**cpio**" on page 205, "**link, unlink**" on page 575, "**ln**" on page 581, and "**mv**" on page 679.

cpio

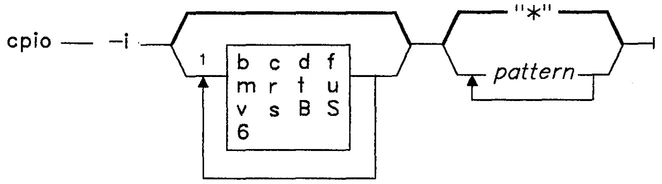
Purpose

Copies files into and out of archive storage and directories.

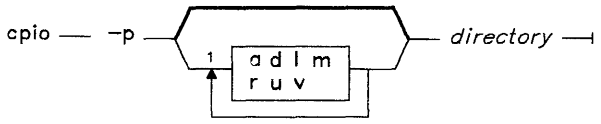
Syntax



OL805175



OL805350



[1] Do not put a blank between these items.

OL805351

Description

Warning: If you redirect the output from **cpio** to a special file (device), you should redirect it to the raw device and not the block device. Because writing to a block device is done asynchronously, there is no way to know if the end of the device has been reached.

cpio -o

This command reads file path names from standard input and copies these files to standard output along with path names and status information. Path names cannot exceed 128 characters. Avoid giving **cpio** path names made up of many unique linked files as it may not have enough memory to keep track of them and so would lose linking information.

cpio -i

This command reads from standard input an archive file created by the **cpio -o** command and copies from it the files with names that match *pattern*. These files are copied into the current directory tree. You may list more than one *pattern*, using the file name notation described under "**sh**" on page 913. Note, however, that in this application the special characters * (asterisk), ? (question mark), and [. . .] (ellipse) match the / (slash) in path names, in addition to their use as described under "**sh**" on page 913. The default *pattern* is * (select all files in the current directory). In an expression such as **[a-z]**, the minus means "through" according to the current collating sequence.

A collating sequence may define ***equivalence classes*** for use in character ranges. See the "Overview of International Character Support" in *Managing the AIX Operating System* for more information on collating sequences and equivalence classes.

Japanese Language Support Information

Note: A collating sequence in Japanese Language Support does not define equivalence classes for use in range expressions. To avoid unpredictable results when using a range expression to match a class of characters, use a ***character class expression*** rather than a standard range expression. For information about character class expressions, see the discussion of this topic in "**ed**" on page 371.

---

cpio -p

This command reads file path names from standard input and copies these files into the named *directory*. The specified *directory* must already exist. If these path names include directory names and if these directories do not already exist, you must use the **d** flag to cause the *directory* to be created.

Note: You can copy special files only if you have superuser authority.

Flags

All flags must be listed together, without any blanks between them. Not all of the following flags can be used with each of the **-o**, **-i**, and **-p** flags.

a	Resets the access times of copied files to the current time.
b	Swaps both bytes and halfwords. **Note:** If there are an odd number of bytes or halfwords in the file being processed, data can be lost.
B	Performs block input/output, 5120 bytes to a record.
c	Writes header information in ASCII character form.
C*value*	Performs block input/output, *value* * 512 bytes to a record. **Note:** The **C** flag and the **B** flag are mutually exclusive. If you list both, **cpio** uses the last one it encounters in the flag list.
d	Creates directories as needed.
f	Copies all files except those matching *pattern*.
l	Links files rather than copies them, whenever possible. This flag is usable only with **cpio -p**.
m	Retains previous file modification time. This flag does not work when copying directories.
r	Renames files interactively. If you do not want to change the file name, enter the current file name or press the **Enter** key only. In this last case, **cpio** does not copy the file.
s	Swaps bytes. This flag is usable only with **cpio -i**. **Note:** If there are an odd number of bytes in the file being processed, data can be lost.
S	Swaps halfwords. This flag is usable only with **cpio -i**. **Note:** If there are an odd number of halfwords in the file being processed, data can be lost.
t	Creates a table of contents. This does not copy any files.
u	Copies unconditionally. An older file now replaces a newer file with the same name.
v	Lists file names. If you use this with the **t** flag, the output looks similar to that of the **ls -l** command.
6	Processes an old file (one written in UNIX Sixth Edition format). This flag is usable only with **cpio -i**.

Examples

1. To copy files onto diskette:

   ```
   cpio  -ov  <filenames  >/dev/rfd0
   ```

   This copies the files with path names that are listed in the file `filenames` in a compact form onto the diskette (> `/dev/rfd0`). The **-v** flag causes **cpio** to display the name of each file as it is copied. This command is useful for making backup copies of files. The diskette must already be formatted, but it must not contain a file system or be mounted.

2. To copy files in the current directory onto diskette:

   ```
   ls  *.c  |  cpio  -ov  >/dev/rfd0
   ```

   This copies all the files in the current directory whose names end with `.c`.

3. To copy the current directory and all subdirectories onto diskette:

   ```
   find  .  -print  |  cpio  -ov  >/dev/rfd0
   ```

   This saves the directory tree that starts with the current directory (.) and includes all of its subdirectories and files. A faster way to do this is:

   ```
   find  .  -cpio  /dev/rfd0  -print
   ```

   The `-print` displays the name of each file as it is copied.

4. To list the files that have been saved onto a diskette with **cpio**:

   ```
   cpio  -itv  </dev/rfd0
   ```

   This displays the table of contents of the data previously saved onto **/dev/rfd0** in **cpio** format. The listing is similar to the long directory listing produced by **li -l**. To list only the file path names, use only the **-it** flags.

5. To copy the files previously saved with **cpio** from a diskette:

   ```
   cpio  -idmv  </dev/rfd0
   ```

   This copies the files previously saved onto **/dev/rfd0** by **cpio** back into (**-i**) the file system. The **-d** flag allows **cpio** to create the appropriate directories if a directory tree was saved. The **-m** flag maintains the last modification time that was in effect when the files were saved. The **-v** causes **cpio** to display the name of each file as it is copied.

6. To copy selected files from diskette:

   ```
   cpio  -i  "*.c"  "*.o"  </dev/rfd0
   ```

   This copies the files that end with `.c` or `.o` from diskette. Note that the patterns `"*.c"` and `"*.o"` must be enclosed in quotation marks to prevent the shell from treating the * as a pattern-matching character. This is a special case in which **cpio** itself decodes the pattern-matching characters.

7. To rename files as they are copied from diskette:

   ```
   cpio  -ir  </dev/rfd0
   ```

   The **-r** flag causes **cpio** to ask you whether or not to rename each file before copying it from diskette. For example, the message:

   ```
   Rename  <prog.c>
   ```

   asks whether to give the file saved as `prog.c` a new name as it is copied in. To rename the file, type the new name and press **Enter**. To keep the same name, you must enter the name again. To avoid copying the file at all, press the **Enter** key alone.

8. To copy a directory and all of its subdirectories:

   ```
   mkdir  /u/jim/newdir
   find  .  -print  |  cpio  -pdl  /u/jim/newdir
   ```

   This duplicates the current directory tree, including the current directory and all of its subdirectories and files. The duplicate is placed in the new directory `/u/jim/newdir`. The **-l** flag causes **cpio** to link files instead of copying them, when possible.

Related Information

The following commands: "**ar**" on page 55, "**find**" on page 422, and "**ln**" on page 581.

The **cpio** system call in *AIX Operating System Technical Reference.*

"Overview of International Character Support" in *Managing the AIX Operating System.*

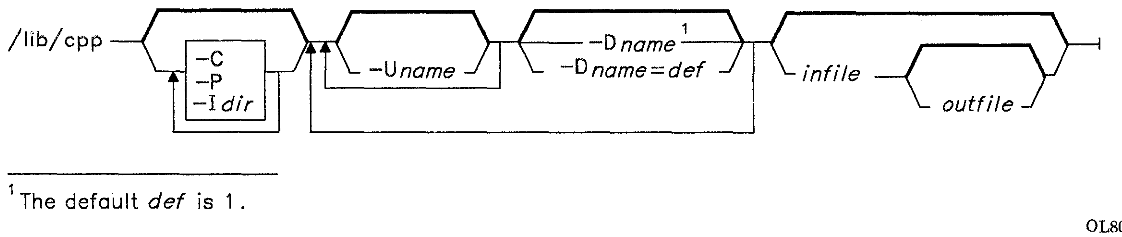
The discussion of Japanese Language Support in *Japanese Language Support User's Guide.*

cpp

Purpose

Performs file inclusion and macro substitution on C Language source files.

Syntax



OL805378

Description

The **cpp** program is the C Language preprocessor. It reads *infile* and writes to *outfile* (standard input and standard output by default). Although you can use this preprocessor by itself, it is best to use it through the **cc** command, which by default sends a C Language source file to **cpp** as the first pass in compilation.

The **cpp** program recognizes two special names, **__LINE__** (the current line number) and **__FILE__** (current file name). These names can be used anywhere just as any other defined name.

All **cpp** directive lines must begin with a hash sign (#). These directives are:

#define *name token-string*
Replaces subsequent instances of *name* with *token-string.*

#define *name*(*arg*, . . . ,*arg*) *token-string*
Replaces subsequent instances of the sequence *name* (*arg*, . . . ,*arg*) with *token-string*, where each occurrence of an *arg* in *token-string* is replaced by the corresponding token in the comma-separated list. Note that there must not be any space between *name* and the left parenthesis.

#undef *name* Ignores the definition of *name* from this point on.

#include *"file"* **#include** *<file>*	Includes at this point the contents of *file*, which **cpp** then processes.
	If you enclose *file* in " ", (double quotation marks) **cpp** searches first in the directory of *infile*, second in directories named with the **-I** flag, and last in directories on a standard list .
	If you use the *<file>* notation, **cpp** searches for *file* only in the standard places. It does not search the directory in which *infile* resides.
#line *num* [*"file"*]	Includes line control information for the next pass of the C compiler. *num* is the line number of the next line and *file* is the file from which it comes. If you omit *"file"*, the current file name remains unchanged.
#endif	Ends a section of lines begun by a test directive (**#if**, **#ifdef**, or **#ifndef**). Each test directive must have a matching **#endif**.
#ifdef *name*	Places the subsequent lines in the output only if *name* has been defined by a previous **#define** and has not been undefined by an intervening **#undef**.
#ifndef *name*	Places the subsequent lines in the output only if *name* has not been defined by a previous **#define** or has been undefined by an intervening **#undef**.
#if *expr*	Places subsequent lines in the output only if *expr* evaluates to nonzero. All the binary nonassignment C operators, the **?:** operator, and the unary **-**, **!**, and **~** operators are legal in *expr*. The precedence of the operators is the same as that defined in the C Language. There is also a unary operator **defined**, which can be used in *expr* in these two forms:
	defined (*name*) **defined** *name*
	This allows the utility of **#ifdef** and **#ifndef** in a **#if** directive. Only these operators, integer constants, and names which are known by **cpp** should be used in *expr*. The **sizeof** operator is not available.
#else	Places subsequent lines in the output only if the expression in the preceding **#if** directive evaluates to False (and hence the lines following the **#if** and preceding the **#else** have been ignored).

You can nest the test directives and the possible **#else** directives.

Flags

-C	Copies source file comments to the output file. If you omit this flag, **cpp** removes all comments (except those found on **cpp** directive lines).
-D*name*[=*def*]	Defines *name* as in a **#define** directive. The default *def* is **1**.
-I*dir*	Looks first in *dir*, then looks in the directories on the standard list for **#include** files with names that do not begin with a / (slash). See the previous discussion of **#include**.
-P	Preprocesses input without producing line control information for the next pass of the C compiler.
-U*name*	Removes any initial definition of *name*, where *name* is a reserved symbol predefined by the preprocessor.

Examples

1. To display the text that the preprocessor sends to the C compiler:

   ```
   /lib/cpp  pgm.c
   ```

   This preprocesses `pgm.c` and displays the resulting text at the work station. You may want to see the preprocessor output when looking for errors in your macro definitions.

2. To create a file containing more readable preprocessed text:

   ```
   /lib/cpp  -P  -C  pgm.c  pgm.i
   ```

   This preprocesses `pgm.c` and stores the result in `pgm.i`. It omits line numbering information intended for the C compiler (**-P**), and includes program comments (**-C**).

3. To predefine macro identifiers:

   ```
   /lib/cpp  -DBUFFERSIZE=512  -DDEBUG  pgm.c  pgm.i
   ```

   This defines `BUFFERSIZE` with the value `512` and `DEBUG` with the value 1 before preprocessing.

4. To use **#include** files located in nonstandard directories:

   ```
   /lib/cpp  -I/u/jim/include  pgm.c
   ```

   This looks in the current directory for quoted **#include** files, then in `/u/jim/include`, and then in the standard directories. It looks in `/u/jim/include` for angle-bracketed **#include** files (< >) and then in the standard directories.

Files

/usr/include	Standard directory for **#include** files.

Related Information

The following commands: "**cc**" on page 140 and "**m4**" on page 603.

craps

Purpose

Plays craps.

Syntax

```
/usr/games/craps —┤
```

OL805188

Description

The **craps** game plays a form of the game of craps that is played in Las Vegas. It simulates the ***roller*** while you place bets. Bet with the roller by making a positive bet or with the ***House*** by making a negative bet.

You start with a $2000 bankroll. When the program prompts with `bet?`, you may bet all or part of your bankroll. If you bet more than your bankroll, the program repeats the prompt until you make a legal bet. Then the roller throws the dice. The payoff odds are one to one. The player wins depending on whether the bet is placed with the roller or with the House. The first roll is the roll immediately following a bet.

The following rules apply. On the first roll, 7 or 11 wins for the roller; 2, 3, or 12 wins for the House; and any other number becomes the ***point*** and you roll again (the next rule then applies). On subsequent rolls, the point wins for the roller; 7 wins for the House; and any other number rolls again.

If you lose your bankroll, the House prompts `marker?`, offering to lend you an additional $2000. Accept the loan by responding `y` or `yes`. Any other response ends the game. When you hold markers, the House reminds you before a bet how many markers are outstanding. When you have markers and your bankroll exceeds $2000, **craps** asks `Repay marker?` If you want to repay part or all of your loan, respond with `y` (or `yes`). If you have more than one marker, **craps** asks you `How many?` If you respond with a number greater than the number of markers you hold, it repeats the prompt until you enter a valid number. If you accumulate 10 markers (a total loan of $20,000), **craps** tells you so and exits. If you accumulate a bankroll of more than $50,000 while holding markers, the money owed is repaid automatically.

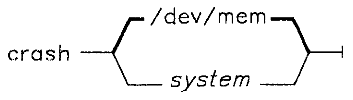
A bankroll of more than $100,000 breaks the bank, and **craps** will prompt `New game?` To quit the game, press INTERRUPT (**Alt-Pause**); **craps** displays whether you have won, lost, or broken even and exits.

crash

Purpose

Examines system images.

Syntax



OL805101

Description

The **crash** command is an interactive utility for examining an operating system image (a core image or the running kernel). It has facilities for interpreting and formatting control structures in the system and certain miscellaneous functions useful for examining a dump.

The *system* parameter specifies the file that contains the system image and the kernel symbol definitions. You can run **crash** with no arguments to examine an active system. The default value is **/dev/mem**. If you specify a system-image file, **crash** assumes it is a system dump file and sets the default process to the process running at the time of the crash.

Notes:

1. When using **crash** to identify the flags it uses, a source listing of system header files may be helpful.
2. Stack tracing of the current process on a running system does not work.

The **crash** command recognizes the following aliases in subcommand *format* specifications.

Format	**Aliases**	**Format**	**Aliases**	**Format**	**Aliases**
byte	b	hexadecimal	hexadec, hex, h, x	octal	oct, o
character	char, c	inode	ino, i	write	w
decimal	dec, e	longdec	ld, D		
directory	direct, dir, d	longoct	lo, O		

Subcommands

The **crash** command presents a prompt (>) when it is ready to interpret subcommands entered at the work station. The general subcommand format for **crash** is:

subcommand [*flags*] [*structures to be displayed*]

When allowed, *flags* modify the format of the data displayed. If you do not specify which structure elements you want to examine, all valid entries are displayed. In general, those subcommands that perform I/O with addresses assume hexadecimal notation.

Most of the subcommands recognized by **crash** have ***aliases*** (abbreviated forms that give the same result). **crash** recognizes the following subcommands:

buf [*buffer-header*] . . .
Displays the system buffer headers.

buffer [*format*] [*buffer*] . . .
Displays the data in a system buffer according to *format.* If you do not provide a *format* parameter, the previous *format* is used. Valid formats include **decimal, octal, hex, character, byte, directory, i-node** and **write**. The **write** format creates a file in the current directory containing the buffer data.

callout Aliases: **calls, call, c, timeout, time, tout**
Displays all entries in the callout table.

cm [*slot-number segment-number*]
If you specify the process slot-number and segment number, this subcommand changes the map of **crash** internal pointers for any segment of a process not swapped out. This allows the **od** subcommand to display data relative to the beginning of the segment desired. If you enter **cm** without any parameters, **cm** resets the map (equivalent of a **reset** subcommand). Use only when analyzing the currently running system.

ds [*data-address*] . . .
Finds the data symbols closest to the given addresses.

du [*slot-number*]
Uses the specified process slot number to display a combined hex and ASCII dump of the user block for any process that has not been swapped out. The default is the current process.

file [*file-table-entry*] . . . Aliases: **files, f**
Displays the file table. Unless specific file entries are requested, only those with a nonzero reference are displayed.

fs [*slot-number*]
Traces a kernel stack for the process specified by the process slot number for any process that has not been swapped out. Displays the called subroutines with a hex dump of the stack frame for the subroutine which contains the parameters passed to the subroutine. The default process is the currently running process.

inode [-] [*i-node-table-entry*] . . . Aliases: **ino, i**
Displays the i-node table. The - flag also displays the i-node data block addresses. Unless specific i-node entries are requested, only those with a nonzero reference are displayed.

map [*map-name*] . . .
Displays the named system map structures.

mount [*mount-table-entry*] . . . Aliases: **mnt, m**
Displays the mount table. Unless specific mount table entries are requested, only those in use are displayed.

nm [*symbol*] . . .
Displays symbol value and type as found in the *kernel-image* file.

od [*symbol name or address*] [*count*] [*format*]
Dumps *count* data values starting at the symbol value or address given according to *format*. Allowable formats are **octal, longoct, decimal, longdec, character, hex** or **byte**.

proc [-] [**-r**] [*process-table-entry*] . . . Aliases: **ps, p**
Displays the process table. (See the **/usr/include/sys/proc.h** file for this structure definition.) The **-r** flag causes only runable processes to be displayed. The - (minus) alone displays a longer listing.

q
Exits from **crash**.

reset Aliases: **r**
Reinitializes the **crash** data, takes another slice from **/dev/mem**, and updates the process table. Any new processes created can be displayed. Use only when analysing the currently running system.

stack [*process-table-entry*] . . . Aliases: **stk, s, kernel, k**
Displays a dump of the kernel stack of a process. The addresses shown are virtual data addresses rather than true physical locations. If you do not specify an entry, information about the last running process is displayed. You can not trace the stack of the current process on a running system.

stat
Displays statistics found in the dump. These include the panic message (if a panic occurred), time of crash, and system name.

text [*text-table-entry*] . . . Aliases: **txt, x**
Displays the text table. Unless specific text entries are requested, only those with a nonzero i-node pointer are displayed.

trace [*process-table-entry*] . . . Aliases: **t**
Displays a kernel stack trace of the current process. The trace starts at the bottom of the stack and attempts to find valid stack frames deeper in the stack. If you do not specify a process table entry, information about the last running process is displayed.

ts [*text-address*] . . .
Finds the text symbols closest to the given addresses.

tty [*type*] [-] [*tty-entry*] . . . Aliases: **term, dz, dh**
Displays the tty structures. The *type* parameter specifies which structure is used (such as **ksr**, or **rs**). The last *type* entered with the **tty** command becomes the default. The - flag displays the **stty** parameters for the given line.

user [*process-table-entry*] . . . Aliases: **uarea, u_area, u**
Displays the user structure of the named process as determined by the information contained in the process table entry. (See the **/usr/include/sys/user.h** file for this structure definition.) If you do not specify the entry, the information about the last running process is displayed. Attempting to display a paged process produces an error message.

var Aliases: **tunables, tunable, tune, v**
Displays the tunable system parameters.

vfs [-] [*vfs slot-number*]
Uses the specified vfs slot number to display an entry in the vfs table. The - flag displays the vnodes associated with the vfs. The default is to display the entire vfs table.

vnode [*vnode slot-number*]
Uses the specified vnode slot number to display an entry in the vnode table. The default is to display the entire vnode structure.

!
: Runs shell commands.

?
: Displays summary of **crash** commands.

Files

/usr/include/sys/*.h	Header files for table and structure information.
/dev/mem	Default system-image file.
/unix	Default kernel-image file.
buf.#	Files containing buffer data.

Related Information

The following commands: "**mount**" on page 669, "**nm**" on page 705, "**ps**" on page 786, "**sh**" on page 913, and "**stty**" on page 1018.

cron

Purpose

Runs commands automatically.

Syntax

```
cron ─┤ 1
```

[1] Not usually run from the command line, but included in **/etc/rc**.

OL805184

Description

The **cron** command runs shell commands at specified dates and times. Regularly scheduled commands can be specified according to instructions contained in **crontab** files. You can submit your **crontab** file via the **crontab** command (see page 222). Use the **at** command (see page 63) to submit commands that are to be run only once. Because **cron** never exits, it should be run only once. This is best done by running **cron** from the initialization process through the **/etc/rc** command file (see page 806).

The **cron** command examines **crontab** files and **at** command files only during process initialization and when a file changes. This reduces the overhead of checking for new or changed files at regularly scheduled intervals.

The **cron** command also executes a **sync** system call approximately once a minute to assure that all information in memory that should be on disk (buffered output) is written out. These periodic updates minimize the possibility of file system damage in the event of a crash. In addition, **cron** keeps a number of frequently used system directories open to keep their i-nodes in kernel memory for faster access.

The **cron** command creates a log of its activities in **/usr/lib/cron/log**.

For a discussion of how to schedule commands, see "**crontab**" on page 222.

Files

/usr/lib/cron	Main cron directory.
/usr/lib/cron/log	Accounting information.
/usr/spool/cron	Spool area.
/bin	Directory kept open.
/lib	Directory kept open.

/usr	Directory kept open.
/usr/bin	Directory kept open.
/usr/lib	Directory kept open.
/etc	Directory kept open.
/tmp	Directory kept open.

Related Information

The following commands: "**at**, **batch**" on page 63, "**crontab**" on page 222, and "**rc**" on page 806.

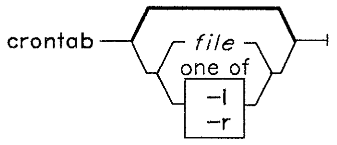
The **sync** system call and the **crontab** file in *AIX Operating System Technical Reference.*

crontab

Purpose

Submits a schedule of commands to **cron**.

Syntax



OL805003

Description

The **crontab** command copies the specified *file*, or standard input if you do not specify a *file*, into a directory that holds all users' **crontab** files. The **cron** command runs commands according to the instructions in these **crontab** files. It then mails you the output from standard output and standard error for these commands, unless you redirect standard output or standard error. When entries are made to a **crontab** file, all previous entries are erased.

You may use **crontab** if your logname appears in the file **/usr/lib/cron/cron.allow**. If that file does not exist, **crontab** checks the file **/usr/lib/cron/cron.deny** to determine if you should be denied access to **crontab**. If neither file exists, you can submit a job only if you are operating with superuser authority. The allow/deny files contain one user name per line.

Notes:

1. If your login ID is associated with more than one login name, **crontab** uses the first login name that appears in the **/etc/passwd** file, regardless of which login name you might actually be using.
2. If **cron.allow** exists, the superuser's log name must appear there for the superuser to be able to use the command.

Each **crontab** file entry consists of a line with six fields, separated by spaces and tabs, that contain, respectively:

1. The minute (0-59)
2. The hour (0-23)
3. The day of the month (1-31)
4. The month of the year (1-12)
5. The day of the week (0-6 for Sunday-Saturday)
6. The shell command.

Each of these fields can contain:

- A number in the specified range
- Two numbers separated by a minus to indicate an inclusive range
- A list of numbers separated by commas, which selects all numbers in the list
- An asterisk, meaning all legal values.

Note that the specification of days may be made by two fields (day of the month and day of the week). If you specify both as a list of elements, both are adhered to. For example, the following entry:

`0 0 1,15 * 1` *command*

would run *command* on the first and fifteenth days of each month, as well as every Monday. To specify days by only one field, the other field should contain an *.

The **cron** command runs the command named in the sixth field at the selected date and time. If you include a % (percent sign) in the sixth field, **cron** treats everything that precedes it as the command invocation and makes all that follows it available to standard input, unless you escape or quote the percent sign (\% or "%").

Note: The shell runs only the first line of the command field (up to a % or end of line). All other lines are made available to the command as standard input.

The **cron** command invokes a subshell from your **$HOME** directory. This means that it will not run your **.profile** file. If you schedule a command to run when you are not logged in and you want to have commands in your **.profile** run, you must explicitly do so in the **crontab** file. (For a more detailed discussion of how **sh** can be invoked, see "**sh**" on page 913).

cron supplies a default environment for every shell, defining **HOME**, **LOGNAME**, **SHELL** (=`/bin/sh`), and **PATH** (=`:/bin:/usr/bin`).

Flags

-l	Lists your **crontab** file.
-r	Removes your **crontab** file from the **crontab** directory.

Examples

The following examples show valid **crontab** file entries.

1. To write the time to the console every hour on the hour:

   ```
   0 * * * * echo The hour is `date`. >/dev/console
   ```

   This example uses ***command substitution***. For more information, see "Command Substitution" on page 925.

2. To run **calendar** at 6:30 a.m. every Monday, Wednesday, and Friday:

   ```
   30 6 * * 1,3,5 /usr/bin/calendar -
   ```

3. To define text for the standard input to a command:

   ```
   0 16 10-31 12 5 /etc/wall%HAPPY HOLIDAYS!%Remember to turn in your time c
   ```

   This writes a message to all users logged in at 4:00 p.m. each Friday between December 10th and 31st.

   The text following the % (percent sign) defines the standard input to the **wall** command as:

   ```
   HAPPY HOLIDAYS!
   Remember to turn in your time card.
   ```

Files

/usr/lib/cron	Main **cron** directory.
/usr/spool/cron/crontabs	Spool area.
/usr/lib/cron/cron.allow	List of allowed users.
/usr/lib/cron/cron.deny	List of denied users.

Related Information

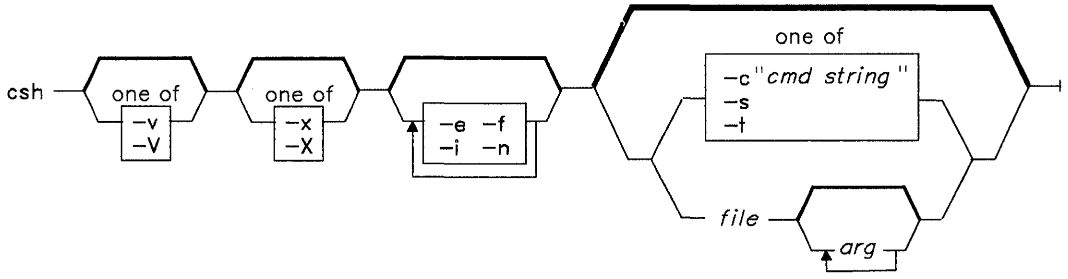
The following commands: "**cron**" on page 220 and "**sh**" on page 913.

csh

Purpose

Interprets commands read from a file or entered from the keyboard.

Syntax



OL805447

Description

The **csh** command is a system command interpreter and programming language that incorporates a history mechanism and a C-like syntax. Like the **sh** command, it is an ordinary user program that reads commands typed at the keyboard and arranges for their execution. In addition, it can read commands from a file, usually called a ***shell procedure*** or a ***command file***.

When you run **csh**, it begins by executing commands from the file **.cshrc** in your home directory, if it exists. If, on the other hand, **csh** runs as a login shell, it executes commands from your **.cshrc** file and your **.login** file.

Commands

A ***simple command*** is a sequence of ***words*** separated by single blanks or tabs.

Japanese Language Support Information

Words can also be separated by double blanks.

A word is a sequence of characters and/or numerals that does not contain blanks without quotation marks. In addition, the following characters and doubled characters also form single words when used as command separators or terminators:

```
&    |    ;    <    >    (    )
&&   ||   <<   >>
```

These special characters may be parts of other words. Preceding them with a \ (backslash), however, prevents the shell from interpreting them as special characters. When the shell is not reading input from a work station, it treats any word that begins with a # (number sign) as a comment and ignores that word and all characters following up to the next new-line character. Strings enclosed in ' ' or " " (matched pairs of quotation characters) or ` ` (grave accents) can also form parts of words. (Blanks, tab characters, and special characters do not form separate words when they are found within these quotation marks.) In addition, within ' ' or " " (pairs of single or double quotation marks), you may include the new-line character by preceding it with \ (backslash).

The first word in the simple-command sequence (numbered 0), usually specifies the name of a command. Any remaining words, with a few exceptions, are passed to that command. If the command specifies an executable file that is a compiled program, the shell immediately runs that program. If the file is marked executable but is not a compiled program, the shell assumes that it is a shell procedure. In this case it spawns another instance of itself (a ***subshell***), to read the file and execute the commands included in it.

A ***pipeline*** is a sequence of one or more commands separated by a | (vertical bar). The output of each command in a pipeline provides the input to the next command.

A ***list*** is a sequence of one or more pipelines separated by a ; (semicolon), & (ampersand), && (two ampersands), or || (two vertical bars) and optionally ended by a ; (semicolon) or an & (ampersand). These separators and terminators have the following effects:

;	Causes ***sequential execution*** of the preceding pipeline (the shell waits for the pipeline to finish).
&	Causes ***asynchronous execution*** of the preceding pipeline (the shell does *not* wait for the pipeline to finish).
&&	Causes the list following it to be executed *only* if the preceding pipeline returns a zero exit value.
||	Causes the list following it to be executed *only* if the preceding pipeline returns a nonzero exit value.

Note: The **cd** command is an exception. If it returns a nonzero exit value, no subsequent commands in a list are executed, regardless of the separators.

The ; and & separators have equal precedence, as do && and ||. The single-character separators have lower precedence than the double-character separators. A new-line character without quotation marks following a pipeline functions the same as a ; (semicolon). Place any of the above in parentheses to form a simple command.

The shell associates a ***job*** with each pipeline. It keeps a table of current jobs and assigns them small integer numbers. When you start a job asynchronously by terminating the command with a &, the shell displays a line that looks like the following:

```
[1]  1234
```

This line indicates that the job number is 1 and that the job is composed of one process with a process-ID of 1234. Use the built-in **jobs** command (page 243) to see what jobs are currently running.

A job running in the background competes for input if it tries to read from the work station. Background jobs can also produce output that competes for the work station and is interleaved there with the output of other jobs.

There are several ways to refer to jobs in the shell. Use the % (percent) character to introduce a job name. This name can be either the job number or the command name that started the job, if this name is unique. So, for example, if a **make** process is running as job 1, you can refer to it as %1. You can also refer to it as %make, if there is only one suspended job with a name that begins with the string make. You can also use

%?:*string*

to specify a job whose name contains *string*, if there is only one such job.

The shell detects immediately whenever a process changes state. Whenever a job becomes blocked so that further progress is not possible, a message is sent to the work station, but not until just before the shell prompt. If, however, the **notify** shell variable is set (see page 237), the shell issues a message that indicates changes in status of background jobs immediately. Use the **notify** built-in command (page 244) to mark a single process so that its status changes are immediately reported. By default, **notify** marks the current process.

History Substitution

History substitution lets you modify individual words from previous commands to create new commands, thus making it easy to repeat commands, repeat the arguments of a previous command in the current command, or fix spelling mistakes in the previous command with little typing.

History substitutions begin with the ! (exclamation) character and may appear anywhere on the command line, provided they do not nest (in other words, a history substitution cannot contain another history substitution). You can precede the ! with a \ to prevent its special meaning. In addition, if you place the ! before a blank, tab, new-line character, = (equal sign), or ((left parenthesis), it is passed unchanged. History substitutions also occur when you begin an input line with a ^ (circumflex). (This special abbreviation is discussed on page 230.) The shell echoes any input line containing history substitutions at the work station before it executes that line.

The history list saves commands that the shell reads from the work station and that consist of one or more words. History substitution reintroduces sequences of words from these saved commands into the input stream.

The **history** shell variable (page 237) controls the size of the history list. You must set the **history** shell variable either in the **.cshrc** file or on the command line with the built-in **set** command (page 245). The previous command is always retained, however, regardless of the value of **history**. Commands in the history list are numbered sequentially starting from 1. The built-in **history** command (page 242) produces output of the type:

```
 9  write michael
10  ed write.c
11  cat oldwrite.c
12  diff *write.c
```

The command strings are shown with their event numbers. It is not usually necessary to use event numbers to refer to events, but you can have the current event number displayed as part of your system prompt by placing an ! in the prompt string assigned to the **prompt** environmental variable (page 238).

A full history reference contains an event specification, a word designator, and one or more modifiers in the following general format:

event[:]*word*:*modifier*[:*modifier*] . . .

Note: Only one word can be modified. A string that contains blanks is not allowed.

In the previous sample of **history** command output, the current event number is 13. Using this example, the following refer to previous events:

Event Specification

`!10`	Refers to event number 10
`!-2`	Refers to event number 11 (the current event minus 2)
`!d`	Refers to a command word beginning with `d` (in this case event number 12)
`!?mic?`	Refers to a command word that contains the string `mic` (in this case, event number 9).

These forms, without further modification, simply reintroduce the words of the specified events, each separated by a single blank. As a special case, `!!` refers to the previous command; the command `!!` alone on an input line reruns the previous command.

To select words from an event, follow the event specification with a : (colon) and one of the following word designators (the words of an input line are numbered sequentially starting from 0):

Word Designator	
0	The first word (the command name)
n	The *n*th argument
^	The first argument
$	The last argument
%	The word matched by an immediately preceding ?*string*? search
x-y	A range of words from the *x*th word to the *y*th word
-y	A range of words from the first word (0) to the *y*th word
*	The first through the last argument, or nothing if there is only one word (the command name) in the event
*x**	The *x*th through the last argument
x-	Like *x** but omitting the last word.

You may omit the colon that separates the event specification from the word designator if the word designator begins with a ^, $, *, -, or %. You can also place a sequence of the following modifiers after the optional word designator, each preceded by a colon:

Modifier

h Remove a trailing path name extension, leaving the head.

r Remove a trailing ".xxx" component, leaving the root name.

e Remove all but the trailing extension ".xxx."

s/*l*/*r*/ Substitute *l* for *r*. With substitutions, it is an error for no word to be applicable.

The *l* (left) side of a substitution is not a pattern in the sense of a string recognized by an editor; rather, it is a word, a single unit without blanks. Normally, a / (slash) delimits the word (*l*) and its replacement (*r*). However, you can use any character as the delimiter if you precede that character with a \ (backslash). Thus, in the following example:

```
s\%/usr/myfile\%/usr/yourfile\%
```

the % becomes the delimiter allowing you to include the / in your word. If you include an & in the replacement, it is replaced by the text from the left-hand side (*l*). A null *l* side is replaced by either the last substitution or by the last string used in the contextual scan !?*string*?. You may omit the trailing delimiter (/) if a new-line character follows immediately.

t	Remove all leading path name components, leaving the tail.
&	Repeat the previous substitution.
g	Apply the change globally, that is, **g&**.
p	Display the new command, but do not run it.
q	Quote the substituted words, thus preventing further substitutions.
x	Act like **q**, but break into words at blanks, tabs, and new-line characters.

Unless the modifier is preceded by a **g**, the change applies only to the first modifiable word.

If you give a history reference without an event specification, for example, `!$`, the shell uses the previous command as the event, unless a previous history reference occurs on the same line, in which case it repeats the previous reference. Thus, the following sequence:

```
!?foo?^ !$
```

gives the first and last arguments of the command that matches `?foo?`.

A special abbreviation of a history reference occurs when the first non-blank character of an input line is a ^ (circumflex). This is equivalent to `!:s^`, thus providing a convenient shorthand for substitutions on the text of the previous line. The command `^lb^lib` corrects the spelling of lib in the previous command.

You can enclose a history substitution in {} (braces), if necessary, to insulate it from the characters that follow. For example, if you want to use a reference to the command:

```
ls -ld ~paul
```

to perform the command:

```
ls -ld ~paula
```

use the following:

```
!{l}a
```

whereas `!la` would look for a command starting with `la`.

Quoting with Single and Double Quotes

Enclose strings in single and double quotation marks to prevent all or some of the substitutions that remain. Enclosing strings in ' ' (single quotation marks) prevents any further interpretation. Enclosing strings in " " (double quotation marks) allows further expansion. In both cases, the text that results becomes (all or part of) a single word. Only in one special case does a string quoted by " " yield parts of more than one word; strings quoted by ' ' never do (see “Command Substitution” on page 231).

Command and File-Name Substitution

The shell performs command and file-name substitutions selectively on the arguments of built-in commands. This means that it does not expand those parts of expressions that are not evaluated. For commands that are not built-in, the shell substitutes the command name separately from the argument list. This occurs very late, after it performs input/output redirection and in a child of the main shell.

Command Substitution

The shell performs command substitution on a command string enclosed in ` ` (grave accents). The shell normally breaks the output from such a command into separate words at blanks, tabs, and new-line characters; this text then replaces the original command string. Within strings surrounded by " " (double quotation marks), the shell treats only the new-line character as a word separator, thus preserving blanks and tabs within the word.

In any case, the single final new-line character does not force a new word. Note that it is therefore possible for command substitution to yield only part of a word, even if the command outputs a complete line.

File-name Substitution

If a word contains any of the characters *, ?, [, or {, or begins with the ~ character, that word is a candidate for file-name substitution, also known as ***globbing***. The word is then regarded as a pattern and replaced with an alphabetically sorted list of file names which match the pattern.

The current collating sequence is used, which may be specified by the environment variables **NLCTAB** or **NLFILE**. In a list of words specifying file-name substitution, it is an error for no patterns to match an existing file name, but it is not required that each pattern match. Only the character-matching symbols *, ?, and [imply pattern matching; the characters ~ and { being more related to abbreviations.

In matching file names, the character . (dot) at the beginning of a file name or immediately following a /, and 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 [*abcd*] matches any one of the enclosed characters. Within [], a lexical range of characters may be indicated by [*a-z*]. The characters that match this pattern are defined by the current collating sequence (see "**ctab**" on page 257).

Japanese Language Support Information

Note: For information about matching file names, see "File-name Substitution in Japanese Language Support" on page 233.

The ~ (tilde) character at the beginning of a file name is used to see home directories. Standing alone, ~ expands to your home directory as reflected in the value of the home shell variable. When followed by a name that consists of letters, digits, and - (dash) 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 anywhere except at the beginning of a word, it is left undisturbed.

The pattern `a{b,c,d}e` is a shorthand for `abe ace ade`. The shell preserves the left-to-right order, with results of matches being stored separately at a low level to preserve this order. This construct may be nested. Thus:

```
~source/s1/{oldls,ls}.c
```

expands to:

```
/usr/source/s1/oldls.c  /usr/source/s1/ls.c
```

if the home directory for `source` is `/usr/source`. Similarly:

```
../{memo,*box}
```

might expand to:

```
../memo  ../box  ../mbox
```

(Note that `memo` is not sorted with the results of matching `*box`.) As a special case, {, }, and {} are passed undisturbed.

File-name Substitution in Japanese Language Support

You can also use the following notation to match file names within a range indication:

[:*charclass*:]

This format instructs the system to match any single character belonging to *class*; the defined classes correspond to **ctype** subroutines. Following are the names of these classes:

alnum	jalpha
alpha	jdigit
digit	jhira
lower	jkanji
print	jkata
punct	jparen
space	jpunct
upper	jspace
xdigit	jxdigit

For example, the expression that matches any single kanji character would be the following:

```
[[:jkanji:]]
```

For additional information about character class expressions, see the discussion of this topic in "**ed**" on page 371.

Alias Substitution

The shell maintains a list of aliases that the **alias** and **unalias** built-in commands (page 240) can establish, display, and modify. After the shell scans the command line, it divides it into distinct commands and checks the first word of each command, left to right, to see if it has an alias. If it does, the shell uses the history mechanism available (see "History Substitution" on page 227), to replace the text of the alias with the text of the command it stands for. The words that result replace the command and argument list. If reference is not made to the history list, the argument list is left unchanged. Thus, if the alias for the **ls** command is `ls -l`, the shell replaces the command `ls /usr` with `ls -l /usr`. The argument list is undisturbed because there is no reference to the history list in the command with an alias. Similarly, if the alias for **lookup** is:

```
grep \!^ /etc/passwd
```

then the shell replaces `lookup bill` with:

```
grep bill /etc/passwd
```

Here, !^ refers to the history list and the shell replaces it with the first argument in the input line, in this case `bill`. Note that you can use special pattern-matching characters in an alias. Thus the command:

```
alias lprint 'pr \!* >> print'
```

makes a command which formats its arguments to the line printer. The ! is protected from the shell in the alias so that it is not expanded until **pr** runs.

If an alias is found, the word transformation of the input text is performed and the alias process begins again on the reformed input line. If the first word of the next text is the same as the old, looping is prevented by flagging it to terminate the alias process. Other loops are detected and cause an error.

Variable Substitution

The shell maintains a set of variables, each of which has as its value a list of zero or more words. Some of these variables are set by the shell or referred to by it. For instance, the **argv** variable is an image of the shell variable list, and words which comprise the value of this variable are referred to in special ways.

You can change and display the values of variables with the **set** and **unset** commands. Of the variables referred to by the shell, a number are toggles (variables that turn something on and off); the shell does not care what their value is, only whether they are set or unset. For instance, the **verbose** variable is a toggle which causes command input to be echoed. The setting of this variable results from the **-v** flag on the command line.

Other operations treat variables numerically. The @ command performs numeric calculations and the result is assigned to a variable. Variable values are, however, always represented as (zero or more) strings. For numeric operations, the null string is considered to be zero, and the second and subsequent words of multiword values are ignored.

After an input line is parsed and alias substitution is performed, and before each command is run, variable substitution is performed, keyed by $ characters. You can prevent this expansion by preceding the $ with a \, except within " " (double quotation marks, where it always occurs, and within ' ' (single quotation marks), where it never occurs. Strings quoted by ' ' are interpreted later (see "Command Substitution" on page 231), so $ substitution does not occur there until later, if at all. A $ is passed unchanged if it is followed by a blank, tab, or new-line character.

Input/output redirections are recognized before variable expansion and are expanded separately. Otherwise, the command name and complete argument list expands together. It is therefore 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 parameters.

Unless enclosed in " " or given the **:q** modifier, the results of variable substitution may themselves eventually be command and file name substituted. Within pairs of double quotation marks, a variable with a value that consists of multiple words expands to a (portion of a) single word, with the words of the variable's value separated by blanks. When you apply the **:q** modifier to a substitution, the variable expands to multiple words. Each word is separated by a blank and quoted to prevent later command or file name substitution.

The following notation allows you to introduce variable values into the shell input. Except as noted, it is an error to reference a variable that is not set.

$*name* ${*name*}	Replaced by the words assigned to *name*, each separated by a blank. Braces insulate *name* from any following characters that would otherwise be part of it. Shell variable names start with a letter and consist of up to 20 letters and digits, including the _ (underline) character. If *name* is not a shell variable but is set in the environment, then that value is returned. The : modifiers and the other forms given below are not available in this case.
$*name*[*selector*] ${*name*[*selector*]}	Used to select only some of the 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 -. The first word of a variable's string 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 * symbol selects all words. It is not an error for a range to be empty if the second argument is omitted or is in range.
$#*name* ${#*name*}	Gives the number of words in the variable. This is useful for later use in a [*selector*].
$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[*]`.

You can apply the modifiers **:gh, :gt**, **:gr**, **:h, :r, :q**, and **:x** to the substitutions above. If {} (braces) appear in the command form, then the modifiers must appear within the braces. The current implementation allows only one : modifier on each $ expansion.

The following substitutions may not be changed with : modifiers.

$?*name* ${?*name*}	Substitutes the string 1 if name is set; 0 if it is not set.
$?0	Substitutes 1 if the current input file name is known; 0 if it is not known.
$$	Substitutes the (decimal) process number of the (parent) shell.
$<	Substitutes a line from the standard input, without further interpretation. Use it to read from the keyboard in a shell procedure.

Predefined and Environmental Variables

The following variables have special meaning to the shell. Of these, **argv**, **cwd**, **home**, **path**, **prompt**, **shell**, and **status** are always set by the shell. Except for **cwd** and **status**, this setting occurs only at initialization. These variables maintain their settings unless you explicitly reset them.

The **csh** command copies the environment variables **USER**, **TERM**, **HOME,** and **PATH** into the **csh** variables **user**, **term**, **home,** and **path**, respectively. The values are copied back into the environment whenever the normal shell variables reset. It is not necessary to worry about the setting of the **path** variable other than in the **.cshrc** file, since **csh** subprocesses import the definition of path from the environment and re-export it if it is changed.

argv	Set to the arguments to the shell; it is from this variable that positional parameters are substituted.
cdpath	Can be given a list of alternate directories to be searched by the **chdir** commands to find subdirectories.
cwd	The full path name of the current directory.
echo	Set when the **-x** command line flag is used; when set, causes each command and its arguments to echo just before it is run. For non-built-in commands, all expansions occur before echoing. Built-in commands are echoed before command and file name substitution, since these substitutions are then done selectively.
histchars	Can be given a string value to change the characters used in history substitution. Use the first character of its value as the history substitution character, this replaces the default character !. The second character of its value replaces the ^ (circumflex) character in quick substitutions.

history	Can be given a numeric value to control the size of the history list. Any command that is referenced in this many events is not discarded. Very large values of **history** may run the shell out of memory. Saves the last command that ran on the history list, regardless of whether **history** is set.
home	Your home directory, initialized from the environment. The file name expansion of ~ refers to this variable.
ignoreeof	If set, the shell ignores an end-of-file character from input devices that are work stations. This prevents shells from accidentally being killed when it reads an end-of-file character (**Ctrl-D**).
mail	The files where the shell checks for mail. This is done after each command completion, which results in a prompt if a specified interval has elapsed. The shell displays the message, `"You have new mail"` if the file exists with an access time not greater than its change time. If the first word of the value of **mail** is numeric, it specifies a different mail checking interval (in seconds); the default is 10 minutes. If you specify multiple mail files, the shell displays the message, "`New mail in` *file*", when there is mail in *file*.
noclobber	If set, places restrictions on output redirection to insure that files are not accidentally destroyed, and that > > redirections see existing files. (See "Redirecting Input and Output" on page 238).
noglob	If set, inhibits file-name expansion. This is most useful in shell procedures that are not dealing with file names, or after a list of file names has been obtained and further expansions are not desirable.
nonomatch	If set, it is not an error for a file-name expansion to not match any existing files; rather, the primitive pattern returns. It is still an error for the primitive pattern to be malformed.
notify	If set, the shell notifies asynchronously of changes in job status. The default presents status changes just before displaying the shell prompt.

path	Each word of the **path** variable specifies a directory in which commands are to be sought for execution. A null word specifies the current directory. If there is no **path** variable set, then only full path names run. The usual search path is the current directory , **/bin**, and **/usr/bin**. For the superuser, the default search path is **/etc**, **/bin**, and **/usr/bin**. A shell which is given neither the **-c** nor the **-t** flags normally hashes the contents of the directories in the **path** variable after reading **.cshrc** and each time the **path** variable is reset. If new commands are added to these directories while the shell is active, it may be necessary to give the **rehash** command (page 245), or the commands may not be found.
prompt	The string which is displayed before each command is read from an interactive work station input. If a ! appears in the string, it is replaced by the current event number. If the ! is in a quoted string, it must be preceded by a \ (backslash). The default prompt is %, # for the superuser.
savehist	Given a numeric value to control the number of entries of the history list that are saved in ~**/.history** when you log off. Any command which is referenced in this many events is saved. During startup, the shell reads ~**/.history** into the history list, enabling history to be saved across logins. Very large values of **savehist** slow down the shell startup.
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 "Nonbuilt-in Command Execution" on page 249). This is initialized to the (system-dependent) home of the shell.
status	The status returned by the last command. If it ended abnormally, then 0200 is added to the status. Built-in commands that fail return exit status 1; all other built-in commands set status 0.
time	Controls automatic timing of commands. If set, any command that takes more than the specified number of CPU seconds causes a line giving user, system, and real times and a utilization percentage, that is the ratio of user-plus-system-times to real time, displays when it ends.
verbose	Set by the **-v** command line flag; causes the words of each command to display after history substitution.

Redirecting Input and Output

You can redirect the standard input and standard output of a command with the following syntax:

< *name*	Opens file *name* (which is first variable, command, and file name expanded) as the standard input.

<< *word* Reads the shell input up to a line which is the same as *word*. *word* is not subjected to variable, file name, or command substitution, and each input line is compared to *word* before any substitutions are done on this input line. Unless a quoting character (\, ", ', or `) appears in *word*, the shell performs variable and command substitution on the intervening lines, allowing \ to quote $, \, and `. Commands which are substituted have all blanks, tabs, and new-line characters preserved, except for the final new-line character, 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* Uses the file *name* as standard output. If the file does not exist, it is made. If the file exists, it is truncated, its previous contents being lost. If the **noclobber** shell variable is set, the file must not exist or be a character special file, or an error results. This helps prevent accidental destruction of files. In this case, use the ! forms to suppress this check. The forms involving & route the diagnostic output into the specified file as well as the standard output. *name* expands in the same way as < input file names.

>> *name*
>>& *name*
>>! *name*
>>&! *name* Uses file *name* as standard output like >, but places output at the end of the file. If the **noclobber** shell variable is set, it is an error for the file not to exist, unless one of the ! forms is given. Otherwise, it is similar to >.

A command receives the environment in which the shell was invoked, as changed by the input/output parameters and the presence of the command as a pipeline. Thus, unlike some previous shells, commands that run from a file of shell commands do not have any access to the text of the commands by default. Rather, they receive the original standard input of the shell. Use the << mechanism to present in-line data. This lets shell command files function as components of pipelines and lets the shell block read its input. Note that the default standard input for a command run detached is not changed to be the empty file **/dev/null**. Rather, the standard input remains as the original standard input of the shell.

To redirect the diagnostics output through a pipe with the standard output, use the form |& (vertical bar ampersand) rather than just | (vertical bar).

Control Flow

The shell contains some commands that can be used to regulate the flow of control in command files (shell procedures) and (in limited but useful ways) from work station input. These commands all operate by forcing the shell to reread or skip in its input and, because of the implementation, restrict the placement of some of the commands.

The **foreach, switch**, and **while** statements, and 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 input is not searchable, the shell buffers input whenever a loop is being read and searches the internal buffer to do the rereading implied by the loop. To the extent that this allows, backward **goto**s succeed on inputs that you cannot search.

Built-in Commands

Built-in commands are run within the shell. If a built-in command occurs as any component of a pipeline except the last, it runs in a subshell.

Notes:

1. If you enter a command from **csh** at the prompt, the system searches for a **csh** built-in command first. If a built-in command does not exist, then the system searches for an AIX command. Some **csh** built-in commands and AIX commands have the same name. However, these commands do not necessarily work the same way. Check the appropriate command description for information on how the command works.
2. If you run a shell procedure from **csh** and the first characters of the shell procedure are #!/*shell_pathname*, **csh** runs the shell specified in the comment to process the procedure. Otherwise, **csh** runs the standard shell (**sh**). If run by **sh**, **csh** built-in commands are not recognized. To get the system to run **csh** commands, the first line of the procedure should be: `#!/bin/csh`.

alias **alias** *name* **alias** *name wordlist*	Displays all aliases (first form). The second form displays the alias for *name*. The final form assigns the specified *wordlist* as the alias of *name*. *wordlist* is command and file name substituted. *name* is not allowed to be **alias** or **unalias**.
break	Resumes running after the end of the nearest enclosing **foreach** or **while**. Runs the remaining commands on the current line. Multilevel breaks are therefore possible by writing them all on one line.
breaksw	Breaks from a **switch**; resumes after the **endsw**.
case *label*:	Defines a label in a **switch** statement, as discussed in the following.

cd
cd *name*
chdir
chdir *name*

Changes the current directory to *name*. If no argument is given, then changes to your home directory.

If *name* is not found as a subdirectory of the current directory and does not begin with /, ./, or ../, then each component of the **cdpath** shell variable is checked to see if it has a subdirectory *name*. Finally, if all else fails, but *name* is a shell variable with a value that begins with /, then this is tried to see if it is a directory.

continue

Continues execution of the nearest enclosing **while** or **foreach**. The rest of the commands on the current line run.

default:

Labels the default case in a **switch** statement. The default should come after all **case** labels.

dirs

Displays the directory stack, the top of the stack is at the left, the first directory in the stack being the current directory.

dirstyle +
dirstyle -

Interprets directories in specified format so you can read their contents in System V format. The + flag converts path names from remote file systems to System V format. The - flag leaves the directory contents in raw form instead of converting the path names of remote file systems to the System V format.

echo *string*
echo -n *string* . . .

Writes the listed *string*s to the shell's standard output, separated by spaces and ending with a new-line character unless you specify the **-n** flag.

else
end
endif
endsw

See the description of the **foreach**, **if**, **switch**, and **while** statements.

eval *arg* . . .

Reads *arg* as input to the shell and runs the resulting command(s) in the context of the current shell. Use this to run commands generated as the result of command or variable substitution, since parsing occurs before these substitutions.

exec *cmd*

Runs the specified command in place of the current shell.

exit

exit (*expr*)	Exits the shell with either the value of the **status** shell variable (first form) or with the value of the specified expression (second form).

foreach *name* (*list*)
. . .
end
	Successively sets *name* to each member of *list* and runs the sequence of commands between the **foreach** and the matching **end**. Both **foreach** and **end** must appear alone on separate lines.
	
	Use the **continue** statement to continue the loop and the **break** statement to end the loop prematurely. When this command is read from the work station, the loop is read once, prompts with ? before any statement in the loop runs. If a mistake is made in entering a loop, it can be corrected before you run the loop. Commands within loops, prompted for by ?, are not placed in the history list.

glob *list*	Functions like **echo**, but does not recognize backslash (\) escapes, and delimits words by null characters in the output. Useful if you want to use the shell to perform file-name substitution to expand a list of words.
goto *word*	Continues to run after the line specified by *word*. The specified *word* is file-name and command expanded to yield a string of the form *label*. The shell rewinds its input as much as possible and searches for a line of the form *label*:, possibly preceded by blanks or tabs.
history **history** *num* **history -r** *num* **history -h** *num*	Displays the history event list. If you specify a number, only the *n* most recent events are displayed. The **-r** flag reverses the order of display to the most recent first rather than the oldest first. The **-h** flag causes the history list to be displayed without leading numbers. Use this to produce files suitable for used with the **-h** flag of the **source** command.
if (*expr*) *cmd*	Runs the single command (with arguments) if the specified expression evaluates true. Variable substitution on *cmd* happens early, at the same time it does for the rest of the **if** statement. *cmd* must be a simple command, not a pipeline, command list, or parenthesized command list. **Note:** Input and output redirection occurs even if *expr* is false (and the command is not executed).
if (*expr*) **then**	

. . .
else if (*expr2*) then
. . .
else
. . .
endif

If *expr* is true, runs the commands that follow the first **then**; **else if** *expr2* is true, runs the commands that follow the second **then**; **else** runs the commands that follow the second **else**. 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**.

jobs
jobs -l

Lists the active jobs. With the **-l** flag, lists process-IDs in addition to the job number and process-ID.

kill *%job*
kill *-signal %job* . . .
kill *pid*
kill *-signal pid* . . .
kill -l

Sends to the jobs or process that you specify either the **TERM** (terminate) signal or *signal*. Specify *signal*s either by number or by names (as given in **/usr/include/signal.h**, stripped of the SIG prefix). Signal names are listed by **kill -l**.

limit
limit *resource*
limit *resource max-use*

Limits the usage by the current process and each process it creates to not individually exceed *max-use* on the specified *resource*. If a *max-use* is not given, the current limit displays; if a *resource* is not given, all limitations are given. Controllable resources are limited to **filesize**, **stacksize**, and **datasize**. You can specify *max-use* as a (floating-point or integer) number followed by a scale factor: **k** or **kilobytes** (1024 bytes), **m** or **megabytes**, or **b** or **blocks** (the units used by the **ulimit** system call). For both *resource* names and scale factors, unambiguous prefixes of the names suffice. The **filesize** may be lowered by an instance of **csh**, but may only be raised by an instance whose effective user-ID is `root`. (See the **ulimit** system call in *AIX Operating System Technical Reference.*)

login

Ends a login shell, and replaces it with an instance of **/bin/login**. This is one way to log off (included for compatibility with the **sh** command).

logout

Ends a login shell. Especially useful if **ignoreeof** is set.

newgrp

Executes the **newgrp** command in the current shell process. See "**newgrp**" on page 689 for a discussion of command options.

nice
nice +*num*
nice *cmd*
nice +*num cmd*

Sets the priority of commands run in this shell to 24 (first form). The second form sets the priority to the specified number. The final two forms run the specified command at priority 24 and the specified number, respectively. If you have superuser authority, you can specify **nice** with a negative number. The command always runs in a subshell, and the restrictions placed on commands in simple **if** statements apply.

nohup
nohup *cmd*

Causes hangups to be ignored for the remainder of the procedure (first form). The second form causes the specified command to be run with hangups ignored. To run a pipeline or list of commands with this form, put the pipeline or list in a shell procedure, give the procedure execute permission, and use the shell procedure as the *cmd*. All processes run in the background with & are effectively protected from being sent a hangup signal when you log off, but will still be subject to explicitly sent hangups unless **nohup** is used.

notify
notify %*job* . . .

Causes the shell to notify you asynchronously when the status of the current or specified jobs changes. Normally, notification is presented just before the shell prompt. This is automatic if the **notify** shell variable is set.

onintr
onintr -
onintr *label*

Controls the action of the shell on interrupts. The first form restores the default action of the shell on interrupts, which is to end shell procedures or to return to the work station command input level. The second form causes all interrupts to be ignored. The third form causes the shell to run a **goto** *label* when it receives an interrupt or a child process ends due to an interruption. In any case, if the shell is running detached and interrupts are being ignored, all forms of **onintr** have no meaning, and interrupts continue to be ignored by the shell and all invoked commands.

popd
popd +*n*

Pops the directory stack, returns to the new top directory. With a +*n*, discards the *n*th entry in the stack. The elements of the directory stack are numbered from the top starting at 0.

pushd **pushd** *name* **pushd** +*n*	With no arguments, exchanges the top two elements of the directory stack. With *name*, changes to the new directory and pushes the old current directory (as given in the **cwd** shell variable) onto the directory stack. With a numeric argument, rotates the *n*th argument of the directory stack around to be the top element and changes to it. The members of the directory stack are numbered from the top starting at 0.
rehash	Causes the internal hash table of the contents of the directories in the **path** shell 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 commands are added to one of the user's own directories, or if someone changes the contents of one of the system directories.
repeat *count cmd*	Runs the specified command, which is subject to the same restrictions as the **if** statement, *count* times. **Note:** I/O redirections occur exactly once, even if *count* is 0.
set **set** *name* **set** *name*=*word* **set** *name*[*index*]=*word* **set** *name*=**(***list***)**	Shows the value of all shell variables (first form). Variables that have more than a single word as their value are displayed as a parenthesized word list. The second form sets *name* to the null string. The third form sets the *index*th component of *name* to *word*; this component must already exist. The final form sets *name* to the list of words in *list*. In all cases, the value is command and file-name expanded. These arguments may be repeated to set multiple values in a single **set** command. However, variable expansion happens for all arguments before any setting occurs.
setenv *name value*	Sets the value of environment variable *name* to be *value*, a single string. The most commonly used environment variables, **USER, TERM,** and **PATH**, are automatically imported to and exported from the **csh** variables **user, term,** and **path**; there is no need to use **setenv** for these. If you modify the environment variables **NLFILE** or **NLCTAB**, the current international character support environment and collating sequence are changed as specified for subsequent commands executed from the shell.
shift	

shift *variable*

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 its value. The second form does the same function on the specified *variable.*

source *name*
source -h *name*

Reads commands from *name.* You can nest the **source** commands. However, if they are nested too deeply, the shell may run out of file descriptors. An error in a **source** command at any level ends all nested **source** commands. Normally, input during **source** commands is not placed on the history list. The **-h** flag causes the commands to be placed in the history list without running.

switch (*string***)**
case *str1***:**
. . .
breaksw
default:
. . .
breaksw
endsw

Successively matches each case label against *string.* The *string* is command and file-name expanded first. Use the pattern-matching characters *, ?, and [. . .] 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 **breaksw** command causes execution to continue after the **endsw**. Otherwise, control may fall through case labels and the **default** labels, as in C. If no label matches and there is no **default**, execution continues after the **endsw**.

time
time *cmd*

With no argument, displays a summary of time used by this shell and its children. If arguments are given, the specified command is timed, and a time summary as described under the **time** shell variable is displayed. If necessary, an extra shell is created to display the time statistic when the command completes.

umask
umask *value*

Displays the file creation mask (first form) or sets it to the specified *value* (second form). The mask is given as an octal value. Common values for the mask are 002, giving all access to owner and group and read and execute access to others, or 022, giving all access to the owner and all access except write access for users in the group or others.

unalias *pattern*

Discards all aliases with names that match *pattern.* Thus, all aliases are removed by **unalias ***. The absence of aliases does not cause an error.

unhash	Disables the use of the internal hash table to locate running programs.
unlimit **unlimit** *resource*	Removes the limitation on *resource*. If you do not specify *resource*, then all *resource* limitations are removed. The only removable limitation is that on **filesize**, and only the superuser can remove it.
unset *pattern*	Removes all variables with names that match the *pattern*. Use **unset *** to remove all variables. It is not an error for nothing to be unset.
unsetenv *pattern*	Removes all variables from the environment whose names match the specified *pattern*. (See the **setenv** built-in command on page 245.)
wait	Waits for all background jobs. If the shell is interactive, an INTERRUPT (**Alt-Pause**) can disrupt the wait, when the shell displays the names and job numbers of all jobs known to be outstanding.

while (*expr*)
. . .
end
	Evaluates the commands between the **while** and the matching **end** while *expr* evaluates nonzero. You can use **break** to end and **continue** to continue the loop prematurely. The **while** and **end** must appear alone on their input lines. If the input is a work station, prompts occur the first time through the loop, as for the **foreach** statement.

@ @ *name* = *expr* @ *name*[*index*] = *expr*	Displays the values of all the shell variables (first form). The second form sets the specified *name* to the value of *expr*. If the expression contains <, >, &, or ¦, then at least this part of the expression must be placed within parentheses. The third form assigns the value of *expr* to the *index*th argument of *name*. Both *name* and its *index*th component must already exist. C operators, such as *= and += are available. The space separating *name* from the assignment operator is optional. Spaces are, however, required in separating components of *expr*, which would otherwise be single words. Special postfix + + and -- operators increase and decrease *name*.

Expressions

The @ built-in command and the **exit**, **if**, and **while** statements accept expressions which include operators similar to those of C, with the same precedence. The following operators are available:

```
*    /    %
+    -
<<   >>
<=   >=   >
==   !=   =~   !~
```

In the preceding list, operators of equal precedence appear on the same line, below those lines containing operators (if any) that have greater precedence and above those lines containing operators having lesser precedence. The ==, !=, =~, and !~ operators compare their arguments as strings; all others operate on numbers. The =~ and !~ operators are similar to != and ==, except that the right-most side is a *pattern* against which the left-hand operand is matched. This reduces the need for use of the **switch** statement in shell procedures when all that is really needed is pattern matching.

Strings which begin with 0 are considered octal numbers. Null or missing arguments are considered 0. The result of all expressions are strings, which represent decimal numbers. It is important to note that now two components of an expression can appear in the same word; except when next to components of expressions which are syntactically significant to the parser (& | < > ( )), expression components should be surrounded by spaces.

Also available in expressions as primitive operands are command executions enclosed in { and } and file inquiries of the form *-l name* where *l* is one of:

r	Read access
w	Write access
x	Execute access
e	Existence
o	Ownership
z	Zero size
f	Plain file
d	Directory

The specified name is command and file-name expanded and then tested to see if it has the specified relationship to the real user. If the file does not exist or is inaccessible, then all inquiries return false, that is, 0. (Command runs succeed, returning true (1), if the command exits with status 0; otherwise they fail, returning false (0).) If more detailed status information is required, run the command outside an expression and the examine **status** shell variable.

Nonbuilt-in Command Execution

When a command to run is found not to be a built-in command, the shell attempts to run the command with *execve*. (See the **exec** system call in *AIX Operating System Technical Reference*.) Each word in the **path** shell variable names a directory from which the shell attempts to run the command. If it is given neither a **-c** nor a **-t** flag, the shell will hash the names in these directories into an internal table so it only tries an **exec** in a directory if there is a possibility that the command resides there. If this mechanism has been turned off with **unhash**, or if the shell is given a **-c** or **-t** (and in any case for each directory component of **path** that does not begin with a `/`), the shell concatenates with the given command name to form a path name of a file, which it then attempts to run.

Parenthesized commands always run in a subshell. Thus, `(cd ; pwd) ; pwd` displays the **home** directory without changing the current directory location, whereas `cd ; pwd` changes the current directory location to the **home** directory. Parenthesized commands are most often used to prevent **chdir** from affecting the current shell.

If the file has execute permissions, but is not an executable binary to the system, then it is assumed to be a file containing shell commands and a new shell runs to read it.

If there is an alias for shell, then the words of the alias will be prefixed to the argument list to form the shell command. The first word of the alias should be the full path name of the shell. Note that this is a special, late-occurring case of alias substitution and only allows words to be prefixed to the argument list without modification.

Signal Handling

The shell normally ignores **QUIT** signals. Jobs running detached are immune to signals generated from the keyboard (**INTERRUPT**, **QUIT**, and **HANGUP**). Other signals have the values the shell inherited from its parent. You can control the shell's handling of **INTERRUPT** and **TERMINATE** signals in shell procedures with **onintr**. Login shells catch the **TERMINATE** signal; otherwise, this signal is passed on to children from the state in the shell's parent. In no case are **INTERRUPT**s allowed when a login shell is reading the **.logout** file.

Limitations

The following are **csh** limitations:

- Words can be no longer than 1024 characters.
- Argument lists are limited to 5120 characters.
- The number of arguments to a command that involves file-name expansion is limited to 1/6th the number of characters allowed in an argument list.

- Command substitutions can 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.

Flags

If the first argument to the shell is - (minus), this is a login shell. The flags are interpreted as follows:

-c	Reads commands from the (single) following argument, which must be present. Any remaining arguments are placed in **argv**.
-e	Exits if any invoked command ends abnormally or yields a nonzero exit status.
-f	Starts without searching for or running commands from the **.cshrc** file in the your home directory.
-i	Prompts for its top-level input (an interactive shell), even if input does not appear to be coming from a work station. Shells are interactive without this flag if their input and output are attached to work stations.
-n	Parses commands but does not run them. This aids in syntactic checking of shell procedures.
-s	Takes command input from the standard input.
-t	Reads and processes a single line of input. You can use a \ to escape the new-line character at the end of the current line to continue onto another line.
-v	Sets the **verbose** shell variable, with the effect that command input is echoed after history substitution.
-V	Sets the **verbose** shell variable even before **.cshrc** runs.
-x	Sets the **echo** shell variable, so that commands are echoed immediately before they run.
-X	Sets the **echo** shell variable even before **.cshrc** runs.

After processing of flag arguments, if arguments remain but none of the **-c**, **-i**, **-s**, or **-t** flags were given, the first parameter is taken as the name of a file of commands (shell procedure). The system opens this file and saves its name for possible resubstitution by **$0**. If the first characters of the shell procedure are *#!/shell_pathname*, **csh** runs the specified shell to process the procedure. Otherwise, **csh** runs the standard shell (**sh**). Remaining parameters initialize the **argv** variable. For more information on the *#!/shell_pathname* comment, see the **exec** system call in *AIX Operating System Technical Reference.*

Files

$HOME/.cshrc	Read at beginning of execution by each shell.
$HOME/.login	Read by login shell, after **.cshrc** at login.
$HOME/.logout	Read by login shell, at logoff.
/bin/sh	Standard shell.
/tmp/sh*	Temporary file for < <.
/etc/passwd	Source of home directories for ~*name*.

Related Information

The following commands: "**cd**" on page 150, "**make**" on page 625, "**pr**" on page 761, and "**sh**" on page 913.

The **access**, **exec**, **fork**, **pipe**, **umask**, and **wait** system calls, the **a.out** and **environ** files, and the **environment** miscellaneous facility in *AIX Operating System Technical Reference.*

"Overview of International Character Support" in *Managing the AIX Operating System.*

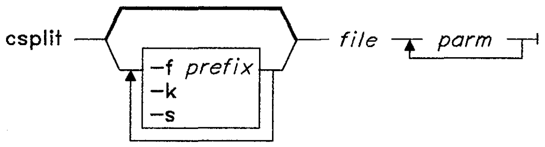
The discussion of Japanese Language Support in *Japanese Language Support User's Guide.*

csplit

Purpose

Splits files by context.

Syntax



OL805177

Description

The **csplit** command reads a *file* and separates it into segments defined by the specified parameters (*parm* . . .). By default, **csplit** writes these segments to files **xx00**+ . . . **xx***n*, where *n* is the number of *parm*s listed on the command line (*n* may not be greater than 99). These new files get the following pieces of *file*:

00: From the start of *file* up to, but not including, the line referenced by the first *parm*.
01: From the line referenced by the first *parm* up to the line referenced by the second *parm*.

.
.
.

n+**1:** From the line referenced by the last *parm* to the end of *file*.

Note that **csplit** does not alter the original *file*.

The specified *parm*s can be a combination of the following:

/pattern/	Creates a file that contains the segment from the current line up to (but not including) the line containing *pattern*, which becomes the current line.
%pattern%	Makes the line containing *pattern* the current line, but does not create a file for the segment.
+*num* -*num*	Moves forward or backward the specified number of lines from the line matched by an immediately preceding *pattern* parameter (for example, `/Page/-5`).

linenum	Creates a file containing the segment from the current line up to (but not including) *linenum*, which becomes the current line.
{*number*}	Repeats the preceding argument the specified *number* of times. This number can follow any of the *pattern* or *linenum* parameters. If it follows a *pattern* parameter, **csplit** reuses that *pattern* the specified number of times. If it follows a *linenum* parameter, **csplit** splits the file from that point every *linenum* of lines for the specified number of times.

Quote all *pattern* parameters that contain blanks or other characters special to the shell. Patterns may not contain embedded new-line characters. In an expression such as **[a-z]**, the minus means "through" according to the current collating sequence. A collating sequence may define ***equivalence classes*** for use in character ranges. See "Overview of International Character Support" in *Managing the AIX Operating System* for more information on collating sequences and equivalence classes.

Flags

-f *prefix*	Specifies the *prefix* name for the created file segments. **xx** is the default *prefix.*
-k	Leaves created file segments intact in the event of an error.
-s	Suppresses the display of character counts.

Examples

1. To split the text of a book into a separate file for each chapter:

   ```
   csplit book "/^ Chapter *[0-9]/" {9}
   ```

   This creates files named **xx00**, **xx01**, **xx02**, . . . ,**xx9**, which contain individual chapters of the file `book`. Each chapter begins with a line that contains only the word `Chapter` and the chapter number. The file **xx00** contains the front matter that comes before the first chapter. The `{9}` after the *pattern* allows up to nine chapters.

2. To specify the prefix for the created file names:

   ```
   csplit -f chap book "/^ Chapter *[0-9]/" {9}
   ```

   This splits `book` into files named `chap`**00**, `chap`**01**, `chap`**02**, . . . ,`chap`**9**.

Related Information

The following commands: "**ed**" on page 371, "**sh**" on page 913, and "**regcmp**" on page 820.

The **regxp** file in *AIX Operating System Technical Reference.*

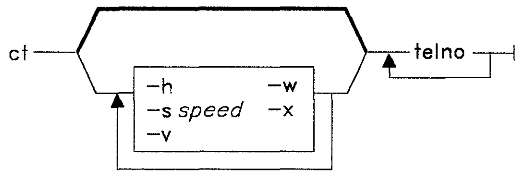
"Overview of International Character Support" in *Managing the AIX Operating System.*

ct

Purpose

Dials an attached terminal and issues a login process.

Syntax



AJ2FL101

Description

The Basic Networking Utilities (BNU) command **ct** enables a user on a remote ASCII terminal, such as an IBM 3161 or a DEC VT100, to communicate with an RT work station over a telephone line attached to a modem at each end of the connection. The user on the remote terminal can then log in and work on the RT work station.

A user on the local system issues **ct** with the appropriate telephone number to call the modem attached to the remote terminal. When the connection is established, **ct** issues an AIX login prompt that is displayed on the remote terminal. The user on the remote terminal enters an AIX login name at the prompt, and AIX opens a new shell. The user at the remote terminal then proceeds to work on the RT PC just like a local user.

Note: In order to establish a **ct** connection, the remote user generally contacts a local user (with a regular phone call) and asks the local user to issue the command.

The **ct** command is useful in the following situations:

- When a user working off site needs to communicate with a local system under strictly supervised conditions. Because the local system contacts the remote terminal, the remote user does not need to know the phone number of the local system.
- When the cost of the connection should be charged either to the local site, or to a specific account on the calling RT. If the remote user has the appropriate access permission and can make outgoing calls on the attached modem, that user can make the equivalent of a collect call. The remote user calls the specified local system, logs in, and issues the phone number of the remote terminal *without* the **-h** flag. The local

system hangs up the initial link so that the remote terminal is free for an incoming call, and then calls back to the modem attached to the remote terminal.

Note: Before issuing the **ct** command, be certain that the remote terminal is attached to a modem that can answer the telephone.

The **ct** command is not as flexible as the BNU command **cu**. For example, the user can not issue AIX commands on the local system while connected to a remote system via **ct**. However, the **ct** command does have two features not available with **cu**:

- The user can instruct **ct** to continue dialing the specified number until the connection is established or a set amount of time has elapsed.
- The user can specify more than one telephone number at a time to instruct **ct** to continue dialing each modem until a connection is established over one of the lines.

If the user specifies alternate dialing paths by entering more than one number on the command line, **ct** tries each line listed in the file **/usr/adm/uucp/Devices** until it finds an available line with appropriate attributes, or runs out of entries. If there are no free lines, **ct** asks if it should wait for one, and if so, for how many minutes. The **ct** command continues to try to open the dialers at 1-minute intervals until the specified time is exceeded. The user can override this prompt by specifying a time with the **-w***n* flag when entering the command.

After the user logs off, **ct** prompts the user on the remote terminal with a reconnect option; the system can either display a new login prompt or drop the line.

Flags

-w*n*	Allows the dialogue to be overridden by specifying *n* as the maximum number of minutes that **ct** is to wait for a line. The command then dials the remote modem at 1-minute intervals until the connection is established or the specified time has elapsed.
-x*n*	Used for debugging. Produces detailed information about the command's execution on standard error output on the local system. The debugging level, *n*, is a single digit between 0 and 9. The recommended default is 9.
-h	Prevents **ct** from hanging up the current line to answer an incoming call.
-v	Allows **ct** to send a running narrative to standard error output.
-s*speed*	Sets the data rate where *speed* is expressed in baud. The default is 1200.
telno	Specifies the telephone number of the modem attached to the remote terminal. The *telno* may include the digits 0 - 9, minus signs (-) representing delays, equal signs (=) representing secondary dial tones, asterisks (*), and pound/number signs (#). The phone number may contain a maximum of 31 characters.

Examples

1. To connect to a modem with an internal number 4-1589 (the - is optional):

   ```
   ct 41589
   ```

   The system responds:

   ```
   Allocated dialer at 1200 baud
   Confirm hang_up? (y to hang_up)
   ```

2. To dial a modem connected to a local telephone number (dialing 9 for an outside line and specifying a 3-minute wait time):

   ```
   ct -w3 9=2453017
   ```

3. To dial a long-distance number (specifying an outside line and a 5-minute wait):

   ```
   ct -w5 9=15026647003
   ```

Files

/usr/adm/uucp/Devices	Information about available devices.
/usr/adm/uucp/Dialcodes	Dialing code abbreviations.
/usr/adm/uucp/Dialers	Initial handshaking on a link.
/usr/adm/uucp/Permissions	Access permission codes.
/usr/adm/uucp/Systems	Accessible remote systems.

Related Information

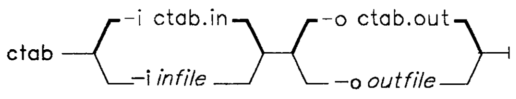
The following commands: "**cu**" on page 263 and "**login**" on page 584.

ctab

Purpose

Produces a collating table.

Syntax



OL805451

Description

The **ctab** command takes an input file (by default a file named **ctab.in** found in the current directory) and produces a binary file (by default named **ctab.out**) containing a collating table. These output files should be stored in a special directory such as **/usr/lib/nls**.

Programs that need the current collating information use the **NLCTAB** environment variable to access that information.

The following conventions are used to make it easier to set up a table file:

- One line of information is present for each character explicitly named.
- A line beginning with the word **option** serves to change one or more of the default conditions or metacharacters built into **ctab**. An **option** line contains a set of name/value pairs, with each half of each pair delimited by tab or space characters. The following is a list of recognized names:

eclass	Turns the use of equivalence classes on or off globally. The assigned value must be **on** (the default) or **off**. (This name is ignored when Japanese Language Support is installed.)
sep	Uses the assigned value as the field separator character. The default value is **:** (colon).
trans	Uses the assigned value of the "translate" indicator in subject character fields. The default character is ¦ (vertical bar).
repeat	Uses the assigned value as the "same as last line" indicator in subject character field. The default value is ^ (circumflex).
comment	Uses the assigned value as the comment character. The default value is the # character.

- The order of the per-character input lines specifies the collating sequence.
- By default, fields on a line are separated by colons. Tabs or spaces may surround fields or separators. You can change the separator character with an option line.
- Use an octal escape sequence in the ASCII range to name a nonprintable character. A backslash character that does not form part of a valid escape sequence serves to strip the following character, including a second backslash, of any special meaning it otherwise would have. For example, to include the colon character in the collating sequence, use the following line:

  ```
  \::
  ```

  The input file format includes a comment convention, namely that the remainder of the line following a # character is ignored. The comment character can be changed with an **option** line.

Input File Specification

Use the following rules to build *infile*, entering field information for each line:

1. The first field on a line contains the ***subject character***, a character to be inserted into the collating sequence at that point.
   - This subject character definition can include a ***translation mechanism***:
     - Instead of a single character, this field may contain two or more characters that are to be collated as a single unit, or
     - The single subject character may be followed by a vertical bar (¦) and a single- or multiple-character string. The vertical bar indicates that the first character will be translated to the second string before being collated.

       For example, to treat an “é” (e acute) as equivalent to the character “e,” use the following line:

       ```
       é¦e
       ```
     - One restriction is placed on the translation mechanism: the subject character cannot be contained in the translated string of characters. For example, the following line is illegal:

       ```
       o¦oe
       ```
   - Any form of the first field may contain a trailing circumflex (^) to indicated that the current character is to collate to the same value as the preceding one. However, a circumflex following a translation string is illegal because the subject character to be translated has no inherent collating value.

- If the subject field contains a string of multiple characters (to collate as a unit), its first character must be declared elsewhere to establish the default collating sequence of that character.
- The translate and collating no-change characters can be changed with **option** lines.

2. The second and third fields specify whether or not a character is alphabetic and what its lower- and upper-case equivalents are:

   - If a subject character is to be treated as a lowercase alphabetic, the second field on its line is its uppercase equivalent, and the third field must be **l** or **L**.
   - If a subject character is to be treated as a uppercase alphabetic, the second field on its line is its lowercase equivalent, and the third field must be **u** or **U**.
   - If a subject character is to be treated as a control character or a space character, the third field must be **c**, **C**, **s**, or **S**.
   - Each character explicitly named whose line contains a non-null second field will be considered alphabetic (that is, matched by **NCisalpha**). Characters that do not have an uppercase or lowercase equivalent (that is, that have a null second field) but that you wish to be considered alphabetic should simply contain a third field that is **l**, **L**, **u**, or **u**.

3. The fourth field on a line is used explicitly to specify the first character in the ***equivalence class*** of the subject character. The members of one equivalence class must be consecutively listed in the input file.

   - There cannot be any gaps within a particular equivalence class. For example, the following lines will put the characters **a**, **b**, and **c** in the same equivalence class:

     ```
     a:A:l:a
     b:B:l:a
     c:C:l:a
     ```

   - As a convenience, if the fourth field is not specified, then the group of consecutive characters with blank fourth fields, provided that they are all based on the same Roman alphabetic character, will be placed in the same equivalence class. To reiterate, only characters with the same base will be placed into the same equivalence class by default. If you wish to have many characters from different bases belong to one equivalence class, as in the preceding example, the first character of the equivalence class has to be specified in the fourth field for every character specified.

- It is illegal to specify an equivalence character that comes later in the collating sequence. The fourth field can refer only to characters that have already been mentioned.
- All ***international character support*** characters not based on Roman alphabetic characters by default are the sole members of their equivalence class.

Japanese Language Support Information

When Japanese Language Support is installed on your system, the information about the second, third, and fourth fields is irrelevant.

Characters not named in the table file that have an ordinal value (that is, a value as an **NLchar**) below the ordinal value of the lowest-valued character named are put into the collating sequence below the first character in the table file. All other characters not named in the table file are put into the collating sequence above the last character in the table file.

The standard characters for decimal and hexadecimal digits are always marked as digits (to be matched by **NCisdigit** and **NCisxdigit**). All other printable characters not marked as alphabetic are marked as punctuation.

Flags

-i *infile*	Specifies the name of the input file (**ctab.in** by default).
-o *outfile*	Specifies the name of the output file (**ctab.out** by default).

Files

/usr/lib/nls/ascii.ctab	Input file listing the ASCII range of characters.
/usr/lib/nls/example.ctab	Input file listing a sample Collating Sequence.

Related Information

The **NCisalpha**, **NCisdigit**, **NCisxdigit**, **nls**, and **NLgetenv** subroutines in *AIX Operating System Technical Reference.*

"Overview of International Character Support" in *IBM RT Managing the AIX Operating System.*

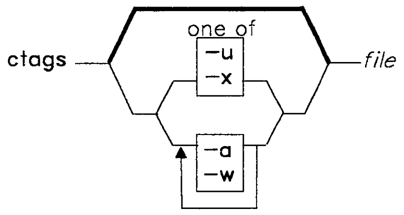
The discussion of Japanese Language Support in *Japanese Language Support User's Guide.*

ctags

Purpose

Makes a file of tags to help locate objects in source files.

Syntax



OL805457

Description

The **ctags** command makes a tags file for **ex** and **vi** editors from the specified C, Pascal, and FORTRAN source files. A tags file gives the locations of specified objects (in this case functions) in a group of files. Each line of the tags file contains the object name, the file in which it is defined, and an address specification for the object definition. Functions are searched with a pattern. Specifiers are given in separate fields on the line, separated by blanks or tabs. Using the tags file, **ex** and **vi** can quickly find these object definitions.

If a file name ends in **.c** or **.h**, it is assumed to be a C source file and is searched for C routine and macro definitions. Others are first examined to see if they contain any Pascal or FORTRAN routine definitions; if not, they are processed again for C definitions.

The tag **main** is treated specially in C programs. The tag formed is created by prefixing **M** to the file name, removing a trailing **.c** (if any), and removing the leading path name components. This makes use of **ctags** practical in directories with more than one program.

Notes:

1. Recognitions of the keywords **function**, **subroutine**, and **procedure** in FORTRAN and Pascal code is performed in a very simple-minded way. No attempt is made to deal with block structure; if you have two Pascal procedures with the same name but in different blocks, **ctags** may yield inadequate results.
2. The **ctags** command does not know about **#ifdef**.

Flags

-a	Appends to tags file.
-w	Suppresses warning diagnostics.
-x	Causes **ctags** to display a list of object names, the line number and file name on which each is defined, as well as the text of that line. This provides a simple index. If you specify this flag, **ctags** does not build a tags file.
-u	Updates the specified files in tags; that is, all references to them are deleted, and the new values are appended to the file. This flag may be slow. (It is usually faster to simply rebuild the tags file.)

Files

tags	Output tags file.

Related Information

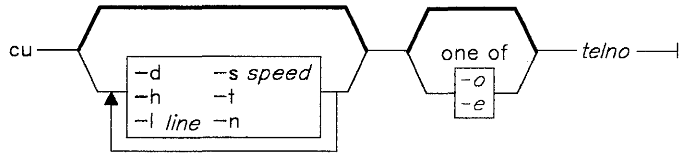
The following commands: "**ex**" on page 407 and "**vi**, **vedit**, **view**" on page 1187.

cu

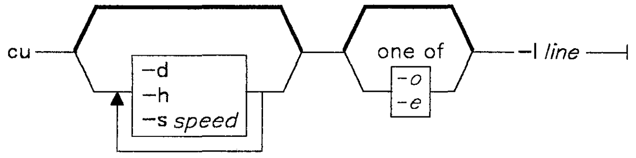
Purpose

Connects directly or indirectly to another UNIX system.

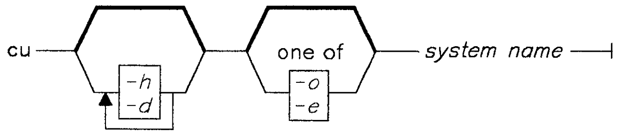
Syntax



OL805553



OL805554



OL805555

Description

The Basic Networking Utilities (BNU) command **cu** connects one system to another UNIX system, to a terminal connected to a UNIX system, or, if the proper hardware and software are installed, to a non-UNIX system. The connection can be established over a hard-wired line, or over a telephone line via a modem.

Once the connection is established, a user can be logged in on both systems at the same time, executing commands on either one without dropping the BNU communication link. If the remote computer is also running under UNIX, the user can transfer ASCII files between the two systems.

Note: The system should already be configured to use the **cu** command. Refer to *Managing the AIX Operating System* for details about this configuration.

After issuing **cu** from the local system, the user must press the **Enter** key (carriage return) and then log in to the remote system.

After making the connection, **cu** runs as two concurrent processes: the ***transmit process*** reads data from standard input and, except for lines beginning with a ~ (tilde), passes that data to the remote terminal. The ***receive process*** accepts data from the remote system and, except for lines beginning with a ~, passes it to standard output. To control input from the remote system so the buffer is not overrun, **cu** uses an automatic DC3/DC1 (**Ctrl-Q/Ctrl-S**) protocol.

In addition to issuing regular AIX commands on the remote system, the user can also issue special **cu** "local commands," which are preceded by a ~. Use these ~ commands to issue AIX commands on the local system and to perform tasks such as transferring files between two UNIX systems.

Local ~ Commands

The transmit process interprets lines beginning with a tilde in the following ways:

~.	Logs the user off the remote computer and terminates the remote connection.
~!	Returns the user to an interactive shell on the local system. Toggle between the local and remote systems using ~! (remote to local) and **Ctrl-D** (local to remote).
~!*cmd...*	Executes the command denoted by *cmd* on the local system via **sh -c**.
~$*cmd...*	Runs the command denoted by *cmd* locally and sends its output to the remote system for execution.
~**%cd**	Changes the directory on the local system.
~**%take** *from* [*to*]	Copies the *from* file on the remote system to the *to* file on the local system. If *to* is omitted, the remote file is copied to the local system under the same file name. As each block of the file is transferred, consecutive single digits are displayed on the terminal screen.
~**%put** *from* [*to*]	Copies the *from* file on the local system to the *to* file on the remote system. If *to* is omitted, the local file is copied to the remote system under the same file name. As each block of the file is transferred, consecutive single digits are displayed on the terminal screen.
~~*line*	Sends the string denoted by ~*line* to the remote system.
~**%break**	Transmits a **BREAK** to the remote system. The **BREAK** can also be specified as ~**%b**.
~**%debug**	Toggles the **-debug** flag on or off; this can also be specified as ~**%d**.
~**t**	Prints the values of the **TERMIO** structure variables for the user's terminal. This is useful for debugging.

~l	Prints the values of the **TERMIO** structure variables for the remote communication line. This is useful for debugging.
~**%nostop**	Toggles between DC3/DC1 input control protocol and no input control. This is useful in case the remote system is one that does not respond properly to the DC3 and DC1 characters.

Note: As soon as the user enters ~**!**,~**%**, ~**$**, ~**t**, or ~**l**, the system displays the name of the local computer in a format such as the following:

~[*system_name*]!/%

The user then enters the command to be executed on the local computer.

Additional Information

- The receive process normally copies data from the remote system to the local system's standard output. Internally, the program accomplishes this by initiating an output diversion to a file when a line from the remote system begins with ~>.

  Data from the remote system is diverted to *file* on the local system. The trailing ~> marks the end of the diversion.

- The use of ~**%put** requires **stty** and **cat** on the remote system. It also requires that the current erase and kill characters on the remote system be identical to these current control characters on the local system. Backslashes are inserted at appropriate places.

- The use of ~**%take** requires **echo** and **cat** on the remote system. Also, **stty tabs** mode should be set on the remote system if tabs are to be copied without expansion to spaces.

- The **cu** command can be used to connect multiple systems, and commands can then be executed on any of the connected systems. For example, issue **cu** on system X to connect to system Y, and then issue **cu** on system Y to connect to system Z. System X is then the local computer, and systems Y and Z are remote computers.

  The user can execute commands on system Z by logging in and issuing the command. Commands can be executed on system X by prefixing the command with a single tilde (~*cmd*), and on system Y by prefixing the command with two tildes (~~*cmd*). In general, one tilde causes the specified command to be executed on the original local computer, and two tildes cause the command to be executed on the next system on which **cu** was issued.

  For example, once the multiple systems are connected, the user can execute the **uname** command with the **-n** flag (to display the node name) on Z, X, and Y as follows:

  ```
  $ uname -n
  Z
  $ ~!uname -n
  X
  $ ~~!uname -n
  Y
  ```

Notes:

1. After executing **cu**, the user must log in to the remote system and press **Enter** (carriage return).
2. The **cu** command does not do integrity checking on data it transfers.
3. Data fields with special **cu** characters may not be transmitted properly.
4. Depending on the interconnection hardware, it may be necessary to use a **~.** to terminate the conversation even if the normal logoff sequence has been used.
5. There is an artificial slowing of transmission by **cu** during the ~**%put** operation so that loss of data is unlikely.
6. The exit code is 0 for normal exit, otherwise, -1.

Flags

-s*speed* Specifies the transmission speed (300, 1200, 2400, 4800, 9600). The default value is "Any" speed, which instructs the system to use the rate appropriate for the default (or specified) transmission line. (The order of the transmission lines is specified in the **/usr/adm/uucp/Devices** file.) Most modems operate at 300, 1200, or 2400 baud, while most hard-wired lines are set to 1200 baud or higher.

-l*line* Specifies a device name to use as the communication line. This can be used to override the search that would otherwise take place for the first available line with the right speed. When the **-l** flag is used without the **-s** flag, the speed of a line is taken from the **/usr/adm/uucp/Devices** file.

When the **-l** and **-s** flags are used together, **cu** searches the **/usr/adm/uucp/Devices** file to check whether the requested speed is available for the specified line. If so, the connection is made at the requested speed; otherwise, an error message is printed, and the call is not made.

The specified device is generally a hard-wired asynchronous line (for example, **/dev/tty2**), in which case a telephone number (*telno*) is not required. If the specified device is associated with a modem, a telephone number must be provided. Using this flag with *system_name* rather than with *telno* does not give the desired result (see *system_name*, below).

Note: Under ordinary circumstances, the user should not have to specify the transmission speed, or a line/device. The defaults set when BNU is installed should be sufficient. Refer to *Managing the AIX Operating System* for information about setting defaults.

-h Emulates local echo, supporting calls to other systems that expect terminals to be set to half-duplex mode.

-t	Used to dial an ASCII terminal that has been set to auto answer. Appropriate mapping of carriage-return to carriage-return line-feed pairs is set.
-d	Prints diagnostic traces.
-o	Designates that odd parity is to be generated for data sent to the remote system.
-e	Designates that even parity is to be generated for data sent to the remote system.
-n	For added security, prompts the user to provide the telephone number to be dialed, rather than taking it from the command line.
telno	When using a modem, the argument is the telephone number, with appropriately placed equal signs for secondary dial tones, or minus signs for delays of 4 seconds.
system_name	A **uucp** system name can be used rather than a telephone number; in that case, **cu** obtains an appropriate hard-wired line or telephone number from **/usr/adm/uucp/Systems**. System names must be ASCII characters only. **Note:** Do not use the *system_name* flag in conjunction with the **-l** and **-s** flags. If you do, **cu** connects to the first available line for the requested system name, ignoring the specified line and speed.

Examples

1. To connect to a remote system using a system name:

   ```
   cu hera
   ```

2. To dial a remote system whose telephone number is 1-201-555-1212, where dialing 9 is required to get an outside dial tone and the baud rate is 1200:

   ```
   cu -s 1200 9=12015551212
   ```

   If the speed is not specified, "Any" is the default value.

3. To log in to a system connected by a hard-wired line:

   ```
   cu -l /dev/tty2
   ```

   or

   ```
   cu -l tty2
   ```

4. To dial a remote system with the specified line and a specific speed:

   ```
   cu -s 1200  -l tty3
   ```

5. To dial a remote system using a specific line associated with a modem:

   ```
   cu -l cul4  9=12015551212
   ```

6. To copy a file from the local system to the remote system (after logging in to the remote system):

   ```
   ~%put /u/amy/file
   ```

   or

   ```
   ~%put /u/amy/file /u/amy/tmpfile
   ```

Files

/etc/locks/LCK..(tty-device)	Prevents multiple use of device.
/usr/adm/uucp/Devices	Information about available links.
/usr/adm/uucp/Dialcodes	Dialing code abbreviations.
/usr/adm/uucp/Dialers	Initial handshaking on a link.
/usr/adm/uucp/Permissions	Access permission codes.
/usr/adm/uucp/Systems	Accessible remote systems.

Related Information

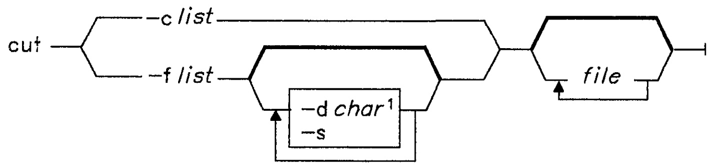
The following commands: "**cat**" on page 137, "**ct**" on page 254, "**echo**" on page 369, "**stty**" on page 1018, "**uuname**" on page 1151, and "**uucp**" on page 1144.

cut

Purpose

Writes out selected fields from each line of a file.

Syntax



1 The default *char* is a tab.

OL805178

Description

The **cut** command cuts out columns from a table or fields from each line of a file and writes these columns or fields to standard output. If you do not specify a *file*, **cut** reads standard input.

You must specify either the **-c** or **-f** flag. The *list* parameter is a comma-separated and/or minus-separated list of integer field numbers (in increasing order). The minus separator indicates ranges. Some sample *list*s are `1,4,7`; `1-3,8`; `-5,10` (short for `1-5,10`); and `3-` (short for third through last field). The fields specified by *list* can be a fixed number of character positions, or the length can vary from line to line and be marked with a field delimiter character, such as a tab character.

You can also use the **grep** command to make horizontal cuts through a file and the **paste** command to put the files back together. To change the order of columns in a file use **cut** and **paste**.

Flags

-c*list*	Specifies character positions. For example, if you specify `-c1-72`, **cut** writes out the first 72 characters in each line of the file. Note that there is no space between **-c** and *list*.

-d*char* Uses the specified *char*acter as the field delimiter when you specify the **-f** flag. You must quote characters with special meaning to the shell, such as the space character. Any ASCII character can be used as *char*.

Japanese Language Support Information

char can either be any ASCII character, or any SJIS character.

-f*list* Specifies a list of fields assumed to be separated in the file by a delimiter character, by default the tab character. For example, if you specify `-f1,7`, **cut** writes out only the first and seventh fields of each line. If a line contains no field delimiters, **cut** passes them through intact (useful for table subheadings), unless you specify the **-s** flag.

-s Suppresses lines that do not contain delimiter characters (use only with the **-f** flag).

Example

To display several fields of each line of a file:

```
cut -f1,5 -d: /etc/passwd
```

This displays the login name and full user name fields of the system password file. These are the first and fifth fields (`-f1,5`) separated by colons (`-d:`).

So, if the **/etc/passwd** file looks like this:

```
su:UHuj9PgdvzOJ":0:0:User with special privileges:/:/bin/sh
daemon:*:1:1::/etc:
bin:*:2:2::/bin:
sys:*:3:3::/usr/src:
adm:*:4:4:System Administrator:/usr/adm:/bin/sh
pierre:boodwqT3irHFE:200:200:Pierre Harper:/u/pierre:/bin/sh
joan:wijBNaYpCZuL.:202:200:Joan Brown:/u/joan:/bin/sh
```

then **cut** produces:

```
su:User with special privileges
daemon:
bin:
sys:
adm:System Administrator
pierre:Pierre Harper
joan:Joan Brown
```

Related Information

The following commands: "**grep**" on page 501 and "**paste**" on page 736.

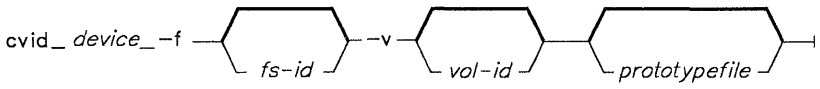
The discussion of Japanese Language Support in *Japanese Language Support User's Guide.*

cvid

Purpose

Creates a VRM install diskette for backup purposes.

Syntax



OL805104

Description

The **cvid** command backs up the VRM minidisk onto a diskette. Since you can reinstall the VRM system from this backup diskette, use **cvid** as a precautionary measure before modifying the VRM. You must be a member of the system group or operating with superuser authority to run this command.

The *device* parameter specifies the device (special file) to which **cvid** copies the VRM. This can be a block device name or a directory name. If *device* is a directory name, **cvid** reads the **/etc/filesystems** file for the corresponding device. **cvid** uses the *prototypefile* parameter to determine the size of the new file system. *prototypefile* defaults to **/vrm/vproto**. For more information on prototype files, see "**mkfs**" on page 658 and "**proto**" on page 780.

When auditing is on, an audit record of the type **cvid** is created.

Warning: If you used the **mv** command to change or modify the order of the files on a minidisk, you could lose files when restoring a **cvid** backup of the minidisk. To avoid problems, modifications, additions or changes to the **/vrm/ldlist/posts** directory must be reflected in the **/vrm/inst.batch** file.

Flags

-f *fs-ID*	Makes *fs-ID* the label for the new file system. The default label is **vrmmnt**.
-v *vol-ID*	Makes *vol-ID* the volume label for the new file system. The default label is **ibmvrm**.

Files

etc/filesystems	Contains device directories.
/vrm/proto	Contains the default file size for new file systems.

Related Information

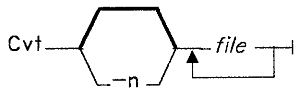
The following commands: "**mkfs**" on page 658 and "**mount**" on page 669.

Cvt

Purpose

Moves old UUCP files into new BNU directories.

Syntax



AJ2FL122

Description

The **Cvt** command moves existing (old) UNIX-to-UNIX Copy Program (UUCP) data and command files into new Basic Networking Utilities (BNU) directories.

After the new BNU programs are installed, issue **Cvt** before attempting to use the BNU functions. The **Cvt** shell handles the command (**C.***) and data (**D.***) files created under the old UUCP facility so that they will run under the new BNU facility. The command first creates the required new BNU directories and then moves the old UUCP files (located in the **/usr/spool/uucp** directory) into those directories.

To issue **Cvt** from the command line, a user must have superuser privileges.

Note: If **Cvt** is not used to move old UUCP command and data files into new BNU directories, those files will not run after the new BNU Program is installed.

Flags

-n	Displays the message explaining the actions that **Cvt** performs but does not execute the command. A *file* can be specified for **D.*** and **C.*** files, but not for **X.*** (execute) files.

Files

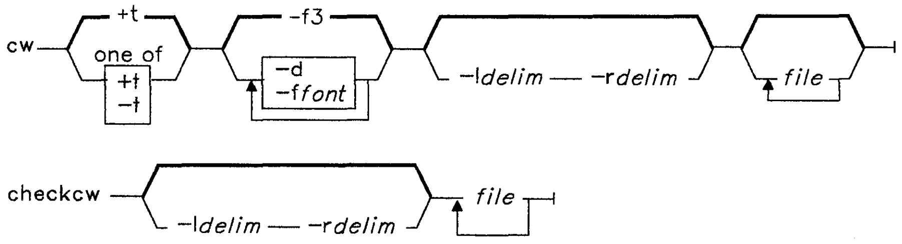
/usr/adm/uucp	Directory in which **Cvt** is stored.
/usr/spool/uucp	Spooling directory.

cw, checkcw

Purpose

Prepares constant-width text for **troff**.

Syntax



OL805427

Description

The **cw** command preprocesses **troff** *file*s containing text to be typeset in the constant-width (CW) font. **cw** reads standard input if you do not specify a *file* or if you specify a -. (minus) as one of the input file names. It writes its output to standard output.

Since the text that is typeset by **cw** resembles the output of line printers and work stations, it can be used to typeset examples of programs and computer output in user manuals and programming texts. It has been designed to be distinctive when used with the Times Roman font.

Because the CW font contains a "nonstandard" set of characters and because text typeset with it requires different character and interword spacing than is used for "standard fonts," you must use **cw** to preprocess documents that use the CW font.

The CW font contains the 94 printing ASCII characters:

```
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789
!$%&()`'*+@.,/:;=?[]|-_^~"<>{}#\
```

plus eight non-ASCII characters represented by four-character **troff** strings (in some cases attaching these strings to "nonstandard" graphics):

Character	Symbol	Troff Name
"Cents" sign	¢	\(ct
EBCDIC "not" sign	¬	\(no
Left arrow	←	\(<-
Right arrow	→	\(->
Down arrow	↓	\(da
Vertical single quote	'	\(fm
Control-shift sign	∩	\(dg
Visible space sign	⊓	\(sq
Hyphen	–	\(hy
Up arrow	↑	\(ua
Home arrow	↖	\(lh

OL805409

The **cw** command recognizes five request lines, as well as user-defined delimiters. The request lines look like **troff** macro requests. **cw** copies them in their entirety onto the output. Thus, you can define them as **troff** macros; in fact, the **.CW** and **.CN** macros ***should*** be so defined. The five requests are:

.CW Marks the start of text to be set in the CW font. This request causes a break. It can take the same flags (in the same format) as those available on the **cw** command line.

.CN Marks the end of text to be set in the CW font. This request causes a break. It can take the same flags (in the same format) as those available on the **cw** command line.

.CD Changes the delimiters and/or settings of other flags. It can take the same flags (in the same format) as those available on the **cw** command line. The purpose of this request is to allow the changing of flags other than at the beginning of a document.

.CP *argument-list* Concatenates all the arguments (delimited like **troff** macro arguments), with the odd-numbered arguments set in the CW font and the even-numbered ones in the prevailing font.

.PC *argument-list* Acts the same as **.CP**, except the even-numbered (rather than odd-numbered) arguments are set in CW font.

The **.CW** and **.CN** requests should bracket text that is to be typeset in the CW font "as is." Normally, **cw** operates in the transparent mode. In that mode, every character between **.CW** and **.CN** request lines represents itself, except for the **.CD** request and the special four-character names listed previously. In particular, **cw** arranges for all periods (.) and apostrophes (') at the beginning of lines, and all backslashes (\) and ligatures (fi, ff, and so on) to be hidden from **troff**. The transparent mode can be turned off by using the **-t** flag, in which case normal **troff** rules apply. In either case, **cw** hides from the user the effect of the font changes generated by the **.CW** and **.CN** requests.

You can also use the **-l** and **-r** flags to define delimiters with the same function as the **.CW** and **.CN** requests. They are meant to enclose words or phrases that are to be set in CW font in the running text. **cw** treats text between delimiters as it does text bracketed by **.CW**/**.CN** pairs, with one exception. Spaces within **.CW**/**.CN** pairs have the same width as other CW characters, while spaces within delimited text are half as wide, so they have the same width as spaces in the prevailing text. Delimiters have no special meaning inside **.CW**/**.CN** pairs.

The **checkcw** command checks that left and right delimiters, and the **.CW**/**.CN** pairs are properly balanced. It prints out all lines in the section with the unmatched delimiters.

Notes:

1. It is unwise to use . (period) or \ (backslash) as delimiter characters.
2. Certain CW characters do not combine well with certain Times Roman characters; for example, the spacing between a CW & (ampersand) followed by a Times Roman comma (,). In such cases, using **troff** half-and quarter-space requests can help.
3. The **troff** code produced by **cw** is difficult to read.
4. The **mm** and **mv** macro packages contain definitions of **.CW** and **.CN** macros that are adequate for most use. If you define your own, make sure that the **.CW** macro invokes the **troff** no-fill (**.nf**) mode, and the **.CN** macro restores the fill mode (**.fi**), if appropriate.
5. When set in running text, the CW font is meant to be set in the same point size as the rest of the text. In displayed matter, on the other hand, it can often be profitably set one point smaller than the prevailing point size. The CW font is sized so that, when it is set in 9-point, there are 12 characters per inch.
6. Documents that contain CW text may also contain tables and equations. If this is the case, the order of preprocessing must be **cw**, **tbl**, and **eqn**. Usually, the tables will not contain any CW text, although it is possible to have elements in the table set in the CW font. Care must be taken that **cw** does not modify the **tbl** format information. Attempts to set equations in the CW font are not likely to be pleasing or successful.
7. In the CW font, overstriking is most easily accomplished with backspaces. Because spaces (and therefore backspaces) are half as wide between delimiters as inside **.CW**/**.CN** pairs, two backspaces are required for each overstrike between delimiters.

Flags

-d	Displays the current flag settings on the standard error output in the form of **troff** comment lines. This flag is meant for debugging.
-f*font*	Replaces *font* with the **cw** font (default=3, replacing the bold font). **-f5** is commonly used for formatters that allow more than four simultaneous fonts. This flag is useful only on the command line.
-l*delim*	Sets the left delimiter as the one-or two-character string *delim*. The left delimiter is undefined by default.
-r*delim*	Set the right delimiter as *delim* The right delimiter is undefined by default. The left and right delimiters may (but need not) be different.
-t	Turns the transparent mode ***off***.
+t	Turns the transparent mode ***on*** (this is the default).

Files

/usr/lib/font/ftCW	CW font-width table.

Related Information

The following commands: "**eqn**, **neqn**, **checkeq**" on page 395, "**mmt**, **checkmm**" on page 666, "**tbl**" on page 1053, and "**troff**" on page 710.

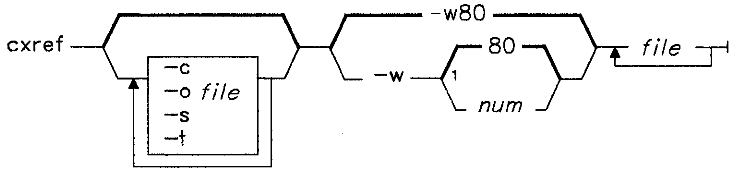
The **mm** and **mv** miscellaneous facilities in *AIX Operating System Technical Reference*.

cxref

Purpose

Creates a C program cross-reference listing.

Syntax



¹ Do not put a space between these items.

OL805180

Description

The **cxref** command analyzes C program *file*s and creates a cross-reference table, using a version of the **cpp** command to include **#define** directives in its symbol table. It writes to standard output a listing of all symbols in each file processed, either separately or in combination (see the **-c** flag on page 279). When a reference to a symbol is that symbol's declaration, an * (asterisk) precedes it.

You can also use the **-D**, **-I**, and **-U** flags from the **cpp** command.

Flags

-c	displays a combined listing of the cross-references in all input files.
-o *file*	Directs the output to the specified *file*.
-s	Does not display the input file names.
-t	Makes the listing 80 columns wide.
-w[*num*]	Makes the listing *num* columns wide, where *num* is a decimal integer greater than or equal to 51. If you do not specify *num* or if *num* is less than 51, the listing will be 80 columns wide.

File

/usr/lib/xcpp	Special version of C-preprocessor.

Related Information

The following commands: "**cc**" on page 140 and "**cpp**" on page 210.

The discussion of **cxref** in *AIX Operating System Programming Tools and Interfaces.*

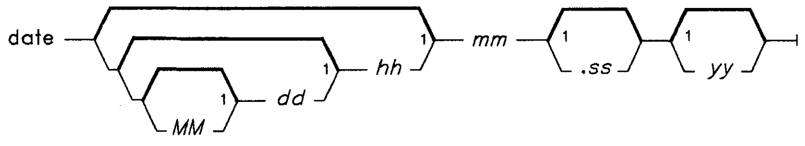
date

Purpose

Displays or sets the date.

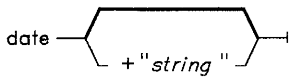
Syntax

Operating With Superuser Authority



OL805105

Operating Without Superuser Authority



[1] Do not put a blank between these items.

OL805357

Description

Warning: Do not change the date while the system is running with more than one user.

If called with no flags or with a flag list that begins with a + (plus sign), the **date** command writes the current date and time to standard output. Otherwise, it sets the current date. Only a user operating with superuser authority can change the date and time. The **NLDATE** variable, if it is defined, controls the ordering of the day and month numbers in the date specifications. The default order is *MMddhhmm.ssyy* where:

- *MM* is the month number
- *dd* is the number of the day in the month
- *hh* is the hour in the day (using a 24-hour clock)
- *mm* is the minute number
- *.ss* is the number of seconds
- *yy* is the last two numbers of the year.

The alternative ordering is *ddMMhhmm.ssyy*.

The current month, day, hour, and year are default values. The system operates in Greenwich Mean Time (GMT). **date** takes care of the conversion to and from local standard and daylight time as specified in the **NLTZ** environment variable.

If you follow **date** with a + (plus sign) and a field descriptor, you can control the output of the command. You must precede each field descriptor with a % (percent sign). The system replaces the field descriptor with the specified value. Enter a literal % as %%. **date** copies any other characters to the output without change. **date** always ends the string with a new-line character. Output fields are fixed size (zero padded if necessary).

Field Descriptors

a	Displays the abbreviated day of the week (`Sun` to `Sat` or the non-English equivalent).
d	Displays the day of month (01 to 31).
D	Displays the date as *mm/dd/yy* (the default), or as *dd/mm/yy*. This format is specified by the **NLDATE** environment variable, if defined.
h	Displays the abbreviated month (`Jan` to `Dec` or the non-English equivalent).
H	Displays the hour (00 to 23).
j	Displays the day of year (001 to 366).
m	Displays the month of year (01 to 12).
M	Displays the minute (00 to 59)
n	Inserts a new-line character.
r	Displays the time in AM/PM notation (or the non-English equivalent).
S	Displays the second (00 to 59).
t	Inserts a tab character.
T	Displays the time as *hh:mm:ss* (the default), or as *mm:hh:ss*. This format is specified by the **NLTIME** environment variable, if defined.
w	Displays the day of the week numerically (Sunday = 0).
y	Displays the last two numbers of year (00 to 99).

Examples

1. To display current date and time:

```
date
```

2. To set the date and time:

   ```
   date  02171425.45
   ```

   This sets the date and time to 14:25:45 (45 seconds after 2:25 p.m.) February 17 of the current year.

3. To display the date and time in a specified format:

   ```
   date  +"%r  %a  %d  %h  %y  (Julian  Date:  %j)"
   ```

   This displays the date (assume current year is 1984) shown in Example 2 as:

   ```
   02:25:03  PM  Fri  17  Feb  84  (Julian  Date:  048)
   ```

Files

/dev/kmem

Related Information

See the **time** and **stime** system calls and the **environment** miscellaneous facility in *AIX Operating System Technical Reference.*

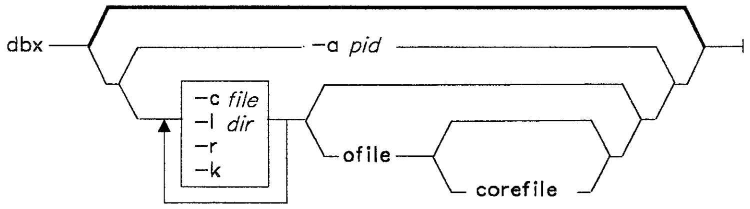
"Overview of International Character Support" in *Managing the AIX Operating System.*

dbx

Purpose

Provides a tool to debug and run programs under AIX.

Syntax



AJ2FL127

Description

The **dbx** command provides a symbolic debugger for C, Pascal, and FORTRAN programs. Use it to do the following:

- Examine object and core 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 *ofile* is an object (executable) file produced by a compiler. Use the **-g** (generate symbol table) flag when compiling your program to produce the information **dbx** needs.

Note: If the object file is not compiled with the **-g** flag or if it contains compiler or loader errors, the symbolic capabilities of **dbx** are limited.

When **dbx** is started, it checks for the **.dbxinit** file in the user's current directory. If the file is not found, it checks the user's **$HOME** directory. If **.dbxinit** exists, its subcommands run at the beginning of the debug session. Use an editor to create a **.dbxinit** file.

If the file **core** exists in the current directory or a *corefile* is specified, use the **dbx** debugger to examine the state of the program when it faulted.

When displaying variables and expressions, **dbx** 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* 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 file name without the language suffix (such as, the **.f** suffix on a FORTRAN program or the **.c** suffix on a C Language program).

Specify expressions in **dbx** with a subset of C and Pascal (or equivalent Modula-2) syntax. A prefix * or a postfix ^ denotes indirection. Use [ ] (square brackets) or ( ) (parentheses) to enclose array subscripts. Use the field reference operator **.** (period) with pointers and records.

Note: This makes the C operator **->** unnecessary (although it is supported). Specify portions of the array by separating the lower and upper bounds with**.** (period).

The **dbx** debugger checks types of expressions. Override types of expressions by using *type-name (expression).* When there is no corresponding named type, use the special construct **&***type-name* to represent a pointer to the named type. Represent a pointer to **enum**, **struct**, or **union tag** with the construct $$*tag-name.*

The following operators are valid in expressions:

Algebraic	`+`, `-`, `*`, `/` (floating), `div` (integral), `mod`, `exp` (exponentiation)
Bitwise	`-`, `|`, `bitand`, `xor`, `~`, `<<`, `>>`
Logical	`or`, `and`, `not`
Comparison	`<`, `>`, `<=`, `>=`, `<>` or `!=`, `=` or `==`
Other	`sizeof`

Flags

-a *pid* Attaches the debugger to a process that is running. The debugger becomes active as soon as the process wakes up. In order to attach the debugger, you need authority to end a process.

-c *file* Runs the **dbx** commands in the file before reading from standard input.

-I *dir* Includes *dir* in the list of directories searched for source files. The default is to look for source files in the current directory and in the directory where the object file is located. The search path is also set with the **use** subcommand.

-k Maps memory addresses, this is useful for kernel debugging.

-r Runs the object file immediately. If it ends successfully, exit **dbx**. Otherwise, enter the debugger and report the reason for termination.

Note: Unless **-r** is specified, **dbx** prompts the user and waits for a command.

Subcommands

Run and Trace Subcommands

call *proc* (*params*)	Executes the object code associated with the named procedure or function. Use **print** *proc* (*params*) to perform the same function, but with a return code of procedure printed.
catch **catch** *signum* **catch** *signame* **ignore** **ignore** *signum* **ignore** *signame*	Starts or stops trapping a signal before it is sent to the program. This subcommand is useful when a program being debugged handles signals such as interrupts. A signal is specified by a number or by a name. Signal names are case insensitive. The **SIG** prefix in names is optional. By default all signals are trapped except **SIGHUP**, **SIGCLD**, **SIGALRM**, and **SIGKILL**.
clear *sline*	Removes all stops at a given source line. The *sline* is an integer or a file name string followed by a **:** (colon) and an integer.
cont **cont** *signum* **cont** *signame*	Continues execution from the current stopping point until the program finishes or another break point is encountered. If a signal is specified, the process continues as though it received the signal. Otherwise, the process is continued as though it had not been stopped.
delete *num* **...**	Removes the traces and stops corresponding to the specified numbers. Use the **status** subcommand to display the numbers associated by **dbx** with a trace or stop.
delete all	Removes all active traces and stops.
detach **detach** *signum* **detach** *signame*	Continues execution from where it stopped without debugger control. If a signal is specified, the process continues as though it received the signal. Otherwise, the debugger will exit, but the debugged process shall continue.

goto *sline*	Makes the specified source line the next line to be executed. **Note:** The source line must be in the same function as the current source line. To override this restriction, set **$vnsafegoto**.
multproc [on] **multproc [off]**	Turns on or off multiprocess debugging. The initial value is off. Issue the command without parameters to check the status of multiprocess debugging.
print *proc* (*params*)	Executes the object code associated with the named procedure or function. You use **call** *proc* (*params*) to perform the same function, but with a return code of procedure called.
next [*num*]	Runs the program up to the next source line. This subcommand and the **step** subcommand differ in that if the line contains a call to a procedure or function, **step** will stop at the beginning of that block, and **next** will not. Use *num* to perform a specfic number of **next** commands.
return [*proc*]	Continues until a return to procedure is executed, or until the current procedure returns if none is specified.

run [*args*][< *file*][> *file*] [> > *file*][2> *file*][2> > *file*] [>& *file*][> >& *file*]

rerun [*args*] [< *file*][> *file*] [> > *file*] [2> *file*][2> > *file*] [>& *file*][> >& *file*]

Starts running the object *file*, passing *args* as command line arguments.

< or > or 2>	Redirects input, output, or standard error, respectively.
>>	Appends redirected output
2>>	Appends redirected standard error.
>&	Redirects both output and standard error to the same file.
>>&	Appends the redirected output and standard error to the same file.

When **rerun** is used without arguments, the previous argument list is passed.

skip *num*	Continues execution from the current stopping point until *num* + 1 breakpoints are encountered or the program finishes.
status [> *file*]	Displays out the currently active trace and stop commands.

step [*num*]	Runs one source line. Use *num* to execute a specific number of lines.
stop if *cond* **stop at** *sline* [**if** *cond*] **stop in** *proc* [**if** *cond*] **stop** *var* [**in** *proc*] [**if** *cond*]	Stops the program when: • The condition is true. • The source line number is reached. • The procedure (or function) is called. • The variable is changed. A condition can be specified for the source line, procedure, or variable stops. The debugger associates numbers with each **stop** subcommand. Use the **status** subcommand to view these numbers. Use the **delete** or **clear** subcommand to turn stopping off. You use the qualified name to get the actual variable stop.
trace **trace in** *proc* [**if** *cond*] **trace** *sline* [**if** *cond*] **trace** *proc* [**in** *proc*][**if** *cond*] **trace** *expr* **at** *sline* [**if** *cond*] **trace** *var* [**in** *proc*][**if** *cond*]	Prints the tracing information for the specified procedure (or function), source line, expression, or variable when the program runs. A condition can be specified. The debugger associates numbers with each **trace** subcommand. Use the **status** subcommand to view these numbers. Use the **delete** subcommand to turn tracing off.
watch *var* [**in** *proc*]	Traces changes to a variable in a watch window if invoked under **xdbx**. Otherwise, this is the same as **trace**.

Subcommands for Examining Program Data

assign *var* = *expr*	Assigns the value of the expression to the variable.

case [default]
case [mixed]
case [lower]
case [upper]
	Changes the way in which the debugger interprets symbols. The default handling of symbols is based upon the current language. Symbols fold to lowercase unless C is the current language. You use this command if a symbol needs to be interpreted in a way not consistent with the current language. Entering this command with no parameters displays the current case mode.

dump [*proc*] [> *file*]	Displays or puts in a file the names and values of variables in the specified procedure. If the procedure specified is . (period), then all active variables are dumped. The default is the current procedure.
print *expr* [,*expr*...]	Prints out the values of the expressions.
whatis *name*	Displays the declaration of *name* where *name* is a variable, procedure, or function name qualified with a block name.
where [> *file*]	Displays a list of the active procedures and functions.
whereis *identifier*	Displays the full qualification of all the symbols whose name matches the specified identifier. The order in which the symbols print is not significant.
which *identifier*	Displays the full qualification of the given identifier (the outer blocks with which the identifier is associated).
up [*count*] **down** [*count*]	Moves the current function, which is used for resolving names, up or down the stack *count* levels. The default is 1.

Subcommands for Accessing Source Files

[*sline-exp* [,sline-exp]] /*regular expression*[/]	Searches forward in the current source file for the specified pattern.
?*regular expression*[?]	Searches backward in the current source file for the specified pattern.
edit [*file*] **edit** *proc*	Invokes an editor with *file* or the current source file if none is specified. If a procedure (or function) *proc* is specified, the editor is invoked on the file that contains it. The default editor

	is **vi**. Override the default by resetting the environment variable **EDITOR** to the name of the desired editor. **Note:** If the procedure has the same name as a file in the same directory, the editor starts the other file, not the file containing the procedure.
file [*file*]	Changes the current source file to *file*, but does not write to the file. If none is specified, displays the name of the current source file.
func [*proc*]	Changes the current function to the specified procedure or function. If none is specified, displays the current function. Changing the current function implicitly changes the current source file to the one containing the function; it also changes the current scope used for name resolution.
list [*proc*]	Lists lines `f-n` to `f+m` where `f` is the first statement in the procedure or function, `n` is a small number, and `m` is the number of lines remaining that fit in the default list window. Use **set $listwindow**=*value* to set or change the number of lines displayed in the list window.
list [*sline-exp*[, *sline-exp*]]	Lists the source lines in the current source file from the first line number to the second inclusive. If no lines are specified, lists the next 10 lines or **$listwindow** lines. An *sline* of $ specifies the current line of execution. An *sline* of `@` specifies the next line to be listed. An *sline-exp* is an *sline* followed by an optional `+` or `-` and an integer.
listi [*proc*]	Lists instructions from the specified procedure or function. The number of instructions displayed is controlled by the **$listwindow** value.
listi at *sline*	Lists instructions beginning with the source line specified.
listi [*address* [*address*]]	Lists instructions from the first address to the second address inclusive. If no lines are specified, list the next **$listwindow** instructions.
move *sline*	Changes the next line to be displayed to *sline*. Changes value of `@`.
use *dir* [*dir* ...]	Sets the list of directories to be searched when looking for source files.

Machine Level Subcommands

address,address/[*mode*][>*file*]

address/[*count*][*mode*] [>*file*]

Displays the contents of memory starting at the first address and continuing up to the second address or until count items are printed. If the address is . (period), the address following the one printed most recently is used. The mode specifies how memory is to be printed; if it is omitted, the previous mode specified is used. The initial mode is **X**. The following modes are supported:

b	Prints a byte in octal.
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.
o	Prints a short word in octal.
O	Prints a long word in octal.
s	Prints a string of characters terminated by a null byte.
x	Prints a short word in hexadecimal.
X	Prints a long word in hexadecimal.

Specify symbolic addresses by preceding the name with an &. Addresses can be expressions made up of other addresses and the operators +, -, and * (indirection). Any expression enclosed in parentheses is interpreted as an address.

cleari *addr*

Remove all the breakpoints at a specified address.

gotoi *addr*

Change program counter address.

registers [>*file*]

Displays the values of all general purpose registers, system control registers, floating point registers, and the current instruction register. General purpose registers are denoted by **$r***n* where *n* is the number of the register. Floating point registers are denoted by **$fr***n*.

stepi [*num*]
nexti [*num*]

Runs a single step as in **step** or **next,** but runs a single instruction rather than source line. If *num* is specified, repeats a single step *num* times.

tracei [*addr*][**if** *cond*]
stopi [*addr*][**if** *cond*]

Traces or sets a stop when the contents of *addr* change.

tracei [*addr*][[**if** *cond*] *addr*][**if** *cond*]

stopi [*var*][**at** *addr*][**if** *cond*]
Turns on tracing or sets a stop at a machine instruction address.

Subcommand Aliases and Variables

alias	Displays aliases for subcommands.
alias *name name* **alias** *name "string"* **alias** *name* (*params*) *"string"*	When subcommands are processed, **dbx** checks first to see if the word is an alias for either a subcommand or a string. If it is, **dbx** treats the input as though the corresponding string (with values substituted for any parameters) has been entered.
unalias *name*	Removes the alias with the given name.
unset *name*	Deletes the debugger variable associated with name.

The set Subcommand

set *var* [= *expr*]
Defines a value (expression) for a debugger variable. The name of the variable cannot conflict with names in the program being debugged. A variable is expanded to the corresponding expression within other commands.

The following variables are selected with **set** and have special meaning:

$dual	Turns on both source- and machine-level **dbx** interface with X-Windows.
$expandunions	Causes **dbx** to display values of each part of variant records or unions.
$frame	Setting this variable to an address causes **dbx** to use the stack frame pointed to by the address for doing stack traces and accessing local variables. This facility is of particular use for kernel debugging.

$hexchars	
$hexin	Causes **dbx** to interpret integers as hexadecimal.
$hexints	
$hexstrings	Causes **dbx** to print out characters, integers, or character pointers respectively in hexadecimal.
$listwindow	Specifies the number of lines to list around a function or when the list command is given without any parameters. Its default value is 10.
$machine	Turns on machine-level **dbx** interface with X-Windows. This variable turns off **$source**.
$mapaddrs	Setting (unsetting) this variable causes **dbx** to start (stop) mapping addresses. This is useful for kernel debugging.
$octin	Causes **dbx** to interpret integers as octal.
$octints	Causes **dbx** to print out integers in octal.
$noargs	Causes **dbx** to omit arguments from commands which walk the stack (where, up, down, dump).
$noflargs	Causes **dbx** to omit display of floating point registers from the **registers** command.
$source	Turns on source-level **dbx** interface with X-Windows.
$unsafeassign	Turns off strict type checking between the two sides of an assign statement. **Note:** Use these variables with great care. They severely limit the usefulness of **dbx** in detecting errors.
$unsafebounds	Turns off subscript checking on arrays.
$unsafecall	Turns off strict type checking for arguments to subroutine or function calls.
$unsafegoto	Turns off **goto** destination checking.

Other Useful Subcommands

help	Prints out a synopsis of common **dbx** commands.
prompt "*string*"	Changes the **dbx** prompt to be the same as *string*.
quit	Quits **dbx**.
screen	Opens a virtual terminal for the **dbx** command interaction. The user continues to operate in the window in which the process originated.
sh *command*	Passes the command line to the shell for execution. The **SHELL** environment variable determines which shell is used.
source *file*	Reads **dbx** commands from the given file.

Files

a.out	Contains object code; object file.
core	Contains core dump.
.dbxinit	Contains initial commands.

Related Information

The following commands: "**cc**" on page 140 and "**xdbx**" on page 1236.

The **a.out** and **core** files in *AIX Operating System Technical Reference.*

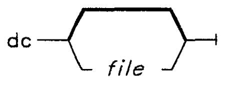
The topic "Debugging Programs" in *AIX Operating System Programming Tools and Interfaces.*

dc

Purpose

Provides an interactive desk calculator for doing arbitrary-precision integer arithmetic.

Syntax



OL805106

Description

The **dc** command is an arbitrary-precision arithmetic calculator. **dc** takes its input from *file* or standard input until it reads an end-of-file character. It writes to standard output. It operates on decimal integers, but you may specify an input base, output base, and a number of fractional digits to be maintained. **dc** is structured overall as a stacking, reverse Polish calculator.

The **bc** command (see page 118) is a preprocessor for **dc**. It provides infix notation and a syntax similar to the C language which implements functions and reasonable control structures for programs.

Subcommands

number	Pushes the specified value onto the stack. A *number* is an unbroken string of the digits 0-9. To specify a negative number, precede it with _ (underscore). A number may contain a decimal point.
+ - / * % ^	Adds (+), subtracts (-), multiplies (*), divides (/), remainders (%), or exponentiates (^) the top two values on the stack. **dc** pops the top two entries off the stack and pushes the result on the stack in their place. **dc** ignores fractional parts of an exponent.
s*x*	Pops the top of the stack and stores it in a register named *x*, where *x* may be any character.
S*x*	Treats *x* as a stack. It pops the top of the main stack and pushes that value onto stack *x*.
l*x*	Pushes the value in register *x* on the stack. The register *x* is not changed. All registers start with zero value.

L*x*	Treats *x* as a stack and pops its top value onto the main stack.
d	Duplicates the top value on the stack.
p	Displays the top value on the stack. The top value remains unchanged. The **p** interprets the top of the stack as an ASCII string, removes it, and displays it.
P	Interprets the top of the stack as a string, removes it, and displays it.
f	Displays all values on the stack.
q	Exits the program. If **dc** is executing a string, it pops the recursion level by two.
Q	Pops the top value on the stack and the string execution level by that value.
x	Treats the top element of the stack as a character string and executes it as a string of **dc** commands.
X	Replaces the number on the top of the stack with its scale factor.
[*string*]	Puts the bracketed *string* onto the top of the stack.
<*x* >*x* =*x*	Pops the top two elements of the stack and compares them. Evaluates register *x* as if it obeys the stated relation.
v	Replaces the top element on the stack by its square root. Any existing fractional part of the argument is taken into account, but otherwise the scale factor is ignored.
!	Interprets the rest of the line as a AIX command.
c	Cleans the stack: **dc** pops all values on the stack.
i	Pops the top value on the stack and uses that value as the number radix for further input.
I	Pushes the input base on the top of the stack.
o	Pops the top value on the stack and uses that value as the number radix for further output.
O	Pushes the output base on the top of the stack.
k	Pops the top of the stack, and uses that value as a nonnegative scale factor. The appropriate number of places displays on output and is maintained during multiplication, division, and exponentiation. The interaction of scale factor, input base, and output base is reasonable if all are changed together.

z	Pushes the number of elements in the stack onto the stack.
Z	Replaces the top number in the stack with the number of digits in that number.
?	Gets and runs a line of input.
;:	**bc** uses these characters for array operations.

Examples

1. To use **dc** as a calculator:

```
    You: 1 4 / p
 System: 0
    You: 1 k        [ Keep 1 decimal place  ]s.
         1 4 / p
 System: 0.2
    You: 3 k        [ Keep 3 decimal places ]s.
         1 4 / p
 System: 0.250
    You: 16 63 5 / + p
 System: 28.600
    You: 16 63 5 + / p
 System: 0.235
```

   You may type the comments (enclosed in `[ ]s.`), but they are provided only for your information.

   When you enter **dc** expressions directly from the keyboard, press **Ctrl-D** to end the **bc** session and return to the shell command line.

2. To load and run a **dc** program file:

```
    You: dc prog.dc
         5 lf x p    [  5 factorial ]s.
 System: 120
    You: 10 lf x p   [ 10 factorial ]s.
 System: 3628800
```

   This interprets the **dc** program saved in `prog.dc`, then reads from the work station keyboard.

The `lf x` evaluates the function stored in register `f`, which could be defined in the program file `prog.c` as:

```
[ f: compute the factorial of n ]s.
[    (n = the top of the stack) ]s.

[ If 1>n do b;  If 1<n do r ]s.
   [d 1 >b d 1 <r] sf

[ Return f(n) = 1           ]s.
   [d - 1 +] sb

[ Return f(n) = n * f(n-1)  ]s.
   [d 1 - lf x *] sr
```

You can create **dc** program files with a text editor, or with the **-c** (compile) flag of the **bc** command. When you enter **dc** expressions directly from the keyboard, press **Ctrl-D** to end the **bc** session and return to the shell command line.

Related Information

The following command: "**bc**" on page 97.

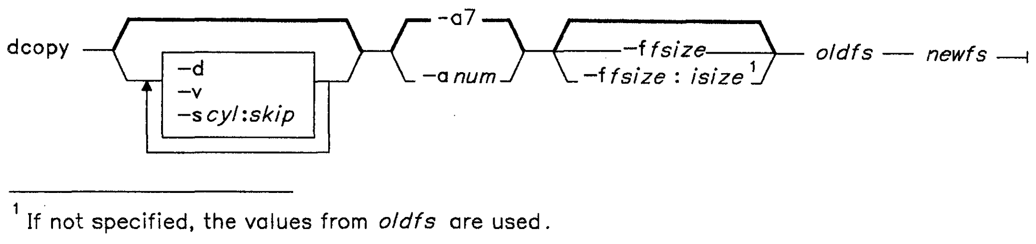
"Overview of International Character Support" in *Managing the AIX Operating System.*

dcopy

Purpose

Copies file systems for the best access time.

Syntax



OL805420

Description

Warning: *oldfs* and *newfs* must not refer to the same minidisk. Doing so will destroy the old file system.

The **dcopy** command copies an existing file system *oldfs* to a new file system *newfs*, appropriately sized to hold the reorganized results. For best results, *oldfs* should be the raw device and *newfs* should be the block device. If *oldfs* or *newfs* is a file system name, **dcopy** uses the corresponding block device given in **/etc/filesystems**. You should run **dcopy** on unmounted file systems (in the case of the root file system, copy to a new minidisk).

If you do not specify any flags, **dcopy** copies files from *oldfs*, compressing directories by removing vacant entries and spacing consecutive blocks in a file by the optimal rotational gap.

The **dcopy** command makes *newfs* identical to *oldfs* and preserves the pack and volume labels. Thus, to compress a file system without moving it, use the **dcopy** command to copy the file to another file system and the **dd** command to copy the file back.

The **dcopy** command catches **INTERRUPT** and **QUIT** signals and reports on its progress. To end **dcopy**, send a **Quit** signal **(Ctrl-V)** and **dcopy** no longer catches **INTERRUPT** or **QUIT**. **dcopy** also attempts to modify its command line arguments so that its progress can be monitored with the **ps** command.

Flags

-a*num*	Places files not accessed in the specified number of days after the free blocks of the destination file system. The default value of *num* is 7. If you do not specify *num*, no files are moved.
-d	Leaves the order of directory entries as is. If you do not specify this flag, **dcopy** moves subdirectories to the beginning of directories.
-f*fsize*[:*isize*]	Specifies the file system and i-node list sizes (in blocks). If not specified, the value from *oldfs* is used.
-s*cyl*:*skip*	Supplies device information for creating the best organization of blocks in a file, where *cyl* is the number of block per cylinder and *skip* is the number of blocks to skip.
-v	Reports how many files were processed and how big the source and destination free lists are.

Related Information

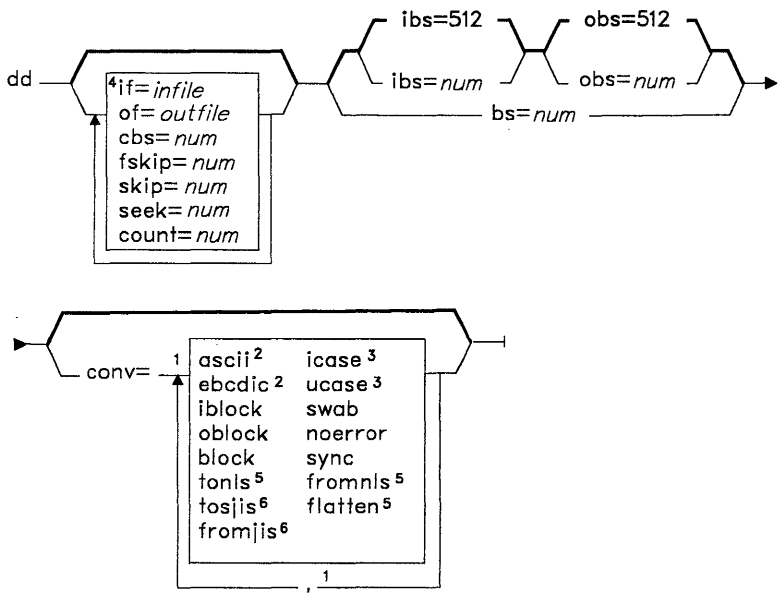
The following commands: "**fsck**, **dfsck**" on page 445, "**mkfs**" on page 658, and "**ps**" on page 786.

dd

Purpose

Converts and copies a file.

Syntax



1 Do not put a blank between these items.
2 Use only one of **ascii** and **ebcdic**.
3 Use only one of **icase** and **ucase**.
4 *infile* and *outfile* default to standard input and standard output.
5 Not active when using Japanese Language Support.
6 Active when using Japanese Language Support.

OL805373

Description

The **dd** command reads the specified *infile* or standard input, does the specified conversions, and copies it to the specified *outfile* or standard output. The input and output block size may be specified to take advantage of raw physical I/O. The terms ***block*** and

record refer to the quantity of data read or written by **dd** in one operation and are not necessarily the same size as a disk block.

Where sizes are specified, a number of bytes is expected. A number may end with **w**, **b**, or **k** to specify multiplication by 2, 512, or 1024 respectively; a pair of numbers can be separated by an **x** to indicate a product.

The conversion requested by **conv=fromnls** translates each extended character in a text file to a printable ASCII escape sequence that uniquely identifies the extended character. The complementary conversion, provided by **conv=tonls**, translates ASCII escape sequences to the corresponding extended character. The conversion requested by **conv=flatten** translates an extended character to the single ASCII character most resembling it in appearance or to a ? (question mark) if no ASCII characters resemble that extended character.

Japanese Language Support Information

The conversion requested by **conv=fromsjis** translates each kanji character in a text file to a printable ASCII escape sequence that uniquely identifies that kanji character. The conversion provided by **conv=tosjis** translates the ASCII escape sequences to the corresponding kanji character.

The character set mappings associated with **conv=ascii** and **conv=ebcdic** are complementary operations, described in the **ebcdic** file in *AIX Operating System Technical Reference*. These attempt to map between ASCII and the subset of EBCDIC that is found on most terminals and keypunches.

The **cbs** specification is used only if the **ascii** or **ebcdic** conversion is specified. For ASCII conversions, **dd** places characters in a conversion buffer of size **cbs**, converts these characters to ASCII, trims trailing blanks and adds new-line characters before sending data specified output. For EBCDIC conversions, it places ASCII characters in the conversion buffer, converts these characters to EBCDIC, adds trailing blanks to create records of size **cbs**.

After it finishes, **dd** reports the number of whole and partial input and output blocks.

Notes:

1. Normally, you need only write access to the output file. However, when the output file is not on a direct access device and you use the **seek** parameter, you also need read access to the file.
2. The **dd** command inserts new-line characters only when converting to ASCII; it pads only when converting to EBCDIC.
3. Use the **backup**, **tar**, or **cpio** commands instead of the **dd** command whenever possible to copy files to tape. These commands are designed for use with tape devices.

4. If you need to use **dd** to copy to a streaming tape and the data is an odd length (not a multiple of 512 bytes), you must use the **conv**=**sync** option to fill the last record. Streaming tape devices permit only multiples of 512 bytes.

Parameters

if=*infile*	Specifies the input file name; standard input is the default.
of=*outfile*	Specifies the output file name; standard output is the default.
ibs=*num*	Specifies the input block size in bytes; the default is 512.
obs=*num*	Specifies the output block size in bytes; the default is 512.
bs=*num*	Specifies both the input and output block size, superseding **ibs** and **obs**.
cbs=*num*	Specifies the conversion buffer size.
skip=*num*	Skip *num* input records before starting copy.
seek=*num*	Seek to the *num*th record from the beginning of output file before copying.
fskip=*num*	Skip past *num* end-of-file characters before starting copy; this parameter is useful for positioning on multifile magnetic tapes.
count=*num*	Copies only *num* input blocks. The default block size is 512 bytes (see the **ibs** parameter).
conv=*spec*[,*spec* . . .]	Specifies one or more of the following conversions:

ascii	Converts EBCDIC to ASCII.
ebcdic	Converts ASCII to EBCDIC.
tonls	Converts ASCII escape sequences to extended characters.
fromnls	Converts extended characters to ASCII escape sequences.
flatten	Converts extended characters to the ASCII character most resembling it, or to a ? (question mark).

Japanese Language Support Information

tosjis	Converts ASCII escape sequences to kanji characters.
fromsjis	Converts kanji characters to ASCII escape sequences.

iblock

oblock **block**	Minimizes data loss resulting from a read or write error on direct access devices. If you specify **iblock** and an error occurs during a block read (where the block size is 512 or the size specified by **ibs**=*num*), **dd** attempts to reread the data block in smaller size units. If **dd** can determine the sector size of the input device, it reads the bad record one sector at a time. Otherwise, it reads it 512 bytes at a time. The input block size (**ibs**) must be a multiple of this "retry size." This allows you to maximize disk input efficiency while ensuring that data loss associated with a read error is confined to a single sector. The **oblock** conversion works similarly on output. Specifying **block** is same as specifying **iblock,oblock**.
lcase	Makes all alphabetic characters lowercase.
ucase	Makes all alphabetic characters uppercase.
swab	Swaps every pair of bytes.
noerror	Does not stop processing on an error.
sync	Pads every input record to **ibs**.

Example

1. To convert an ASCII text file to EBCDIC:

   ```
   dd if=text.ascii of=text.ebcdic conv=ebcdic
   ```

   This converts `text.ascii` to EBCDIC representation, storing the EBCDIC version in `text.ebcdic`.

   Note: When you specify `conv=ebcdic`, **dd** converts the ASCII ^ (circumflex) character to an unused EBCDIC character (9A hexadecimal), and ASCII ~ (tilde) to EBCDIC ¬ (NOT symbol).

2. To use **dd** as a filter:

   ```
   li -l | dd conv=ucase
   ```

   This displays a long listing of the current directory (`li -l`) in uppercase.

Related Information

The following command: "**cp**" on page 202.

The **ebcdic** and **tape** files in *AIX Operating System Technical Reference.*

"Overview of International Character Support" in *Managing the AIX Operating System.*

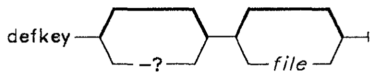
The discussion of Japanese Language Support in *Japanese Language Support User's Guide.*

defkey

Purpose

Defines keyboard key assignments.

Syntax



OL805453

Description

The **defkey** command lets you redefine the keyboard keys on the active virtual terminal. Input to **defkey** comes either interactively from the keyboard or from a redirected file. Key assignments can be a single character, non-spacing characters, or strings.

If you specify a *file* that does not exist, **defkey** creates and opens the file; if *file* exists, **defkey** opens the file. It then displays a menu that prompts you for input. This *file* can then be used as redirected input to **defkey**.

Flags

-? Provides help information.

Examples

1. To redefine a key or keys and create or add to a keyboard definition file:

   ```
   defkey mykeys
   ```

   This creates the file `mykeys` and prompts for input. When **defkey** ends, the keys that you specified will be redefined on the active virtual terminal. You can also use the file `mykeys` to redefine the keyboard on another virtual terminal with the command:

   ```
   defkey  < mykeys
   ```

2. To interactively redefine one or more keyboard keys for the active virtual terminal:

   ```
   defkey
   ```

Related Information

hft and **dispsym** in *AIX Operating System Technical Reference.*

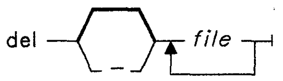
Keyboard Description and Character Reference.

del

Purpose

Deletes files if the request is confirmed.

Syntax



OL805049

Description

The **del** command displays the list of specified *file* names and asks you to confirm your request to delete the group of files. To answer yes (delete the files), press the **Enter** key or enter a line beginning with y. Any other response specifies no (do not delete the files).

Japanese Language Support Information

An affirmative response in Japanese Language Support matches one of the elements in the environment variable **YESSTR**.

The **del** command does not delete directories. See "**rmdir**" on page 838 for information about deleting directories.

Warning: The **del** command ignores file protection, allowing the owner of a file to delete a write-protected file. However, to delete a file, you must have write permission in the directory that the file exists in.

Since pressing the **Enter** key by itself is the same as answering "yes," be careful not to delete files accidentally.

Flag

-	Requests confirmation for each specified *file* rather than for the entire group.

Examples

1. To delete a file:

   ```
   del  chap1.bak
   ```

   This displays the message:

   ```
   delete chap1.bak? (y)
   ```

   to ask for confirmation before deleting `chap1.bak`. The `(y)` reminds you to press the **Enter** key or to enter `y` to answer yes.

2. To use **del** with pattern-matching characters:

   ```
   del  *.bak
   ```

   Before passing the command line to **del**, the shell replaces the pattern `*.bak` with the names of all the files in the current directory that end with `.bak`. (This is known as ***file-name expansion***.) **del** asks for confirmation before deleting them all at one time.

3. To interactively select files to be deleted:

   ```
   del  -  *
   ```

   This displays the name of each file in the current directory one at a time, allowing you to select which ones to delete.

Japanese Language Support Information

The allowed affirmative responses are defined in the environment variable **YESSTR**.

Related Information

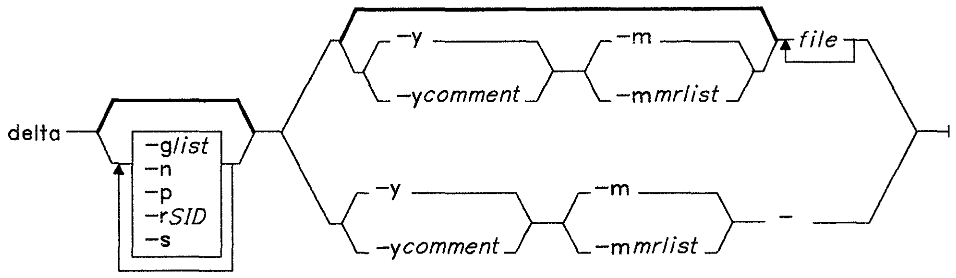
The following commands: "**rmdir**" on page 838 and "**rm**" on page 833.

delta

Purpose

Creates a delta in a Source Code Control System file.

Syntax



OL805056

Description

The **delta** command is used to introduce into the named Source Code Control System (***SCCS***) *file* any changes that were made to the file version retrieved by a **get -e** command.

The **delta** command reads the ***g-files*** that correspond to the specified *file*s (see "SCCS Files" on page 478) and creates a new delta.

If you specify a directory in place of *file*, **delta** performs the requested actions on all SCCS files within that directory (that is, on all files with the **s.** prefix). If you specify a - (minus) in place of *file*, **delta** reads standard input and interprets each line as the name of an SCCS file. When **delta** reads standard input, you must supply the **-y** flag. You must also supply the **-m** flag if the **v** header flag is set. (For more information on header flags, see the discussion in the **admin** command on page 44.) **delta** reads standard input until it reaches END OF FILE (**Ctrl-D**).

If you are not familiar with the delta numbering system, see *AIX Operating System Programming Tools and Interfaces* for more information.

Note: Lines beginning with an **SOH** ASCII character (binary 001) cannot be placed in the SCCS file unless the **SOH** is quoted using a \ (backslash). SOH has special meaning to SCCS and causes an error. See the **sccsfile** file in *AIX Operating System Technical Reference.*

A **get** of many SCCS files, followed by **delta** of those files, should be avoided when the **get** generates a large amount of data. Instead, you should alternate the use of **get** and **delta**.

Flags

-g*list*	Specifies a list of *SID*s (deltas) that are to be ignored when the **get** command creates the g-file. After you use this flag, **get** ignores this delta if it is one that it should not include when it builds the g-file.
-m[*mrlist*]	If the SCCS file has the **v** header flag set, then a Modification Request (MR) number must be supplied as the reason for creating the new delta.
	If you do not specify the **-m** flag, and the **v** header flag is set, **delta** reads MRs from the standard input. If standard input is a work station, **delta** prompts you for the MRs. **delta** continues to take input until it reads END OF FILE (**Ctrl-D**). It always reads MRs before the comments (see the **-y** flag). You can use blanks, tab characters, or both to separate MRs in a list.
	If the **v** header flag has a value, it is interpreted as the name of a program that validates the MR numbers. If **delta** returns a nonzero exit value from the MR validation program, **delta** assumes some of the MR numbers were invalid and stops running.
-n	Retains the g-file, which is normally removed at completion of **delta** processing.
-p	Writes to standard output (in the format of the **diff** command) the SCCS file differences before and after the delta is applied. See "**diff**" on page 320 for an explanation of the format.
-r*SID*	Specifies which delta is to be made to the SCCS file. You must use this flag only if two or more outstanding **get -e** commands were done on the same SCCS file by the same person. The *SID* can be either the SID specified on the **get** command line or the SID to be made as reported by the **get** command (see Figure 2 on page 481 for additional information). An error results if the specified SID cannot be uniquely identified, or if a SID must be specified but it is not.
-s	Suppresses the information normally written to standard output on normal completion of the **delta** command.
-y[*comment*]	Specifies text used to describe the reason for making the delta. A null string is considered a valid *comment.* If your comment line includes special characters or blanks, the line must be enclosed in single or double quotation marks.
	If you do not specify **-y**, **delta** reads comments from standard input until it reads a blank line or END OF FILE (**Ctrl-D**). If input is from the keyboard, **delta** prompts for the comments. If the last character of a line is a backslash, it is ignored. Comments must be no longer than 512 characters.

Japanese Language Support Information

Comments can include kanji characters.

Example

To record changes you have made to an SCCS file:

```
delta  s.prog.c
```

This adds a delta to the SCCS file `s.prog.c`, recording the changes made by editing `prog.c`. **delta** then asks you for a comment that summarizes the changes you made. Enter the comment, then press END OF FILE (**Ctrl-D**) or press the **Enter** key twice to indicate that you have finished the comment.

Related Information

The following commands: "**admin**" on page 41, "**bdiff**" on page 102, "**cdc**" on page 152, "**get**" on page 477, "**help**" on page 513, "**prs**" on page 781, and "**rmdel**" on page 837.

The **sccsfile** file in *AIX Operating System Technical Reference.*

The discussion of SCCS in *AIX Operating System Programming Tools and Interfaces.*

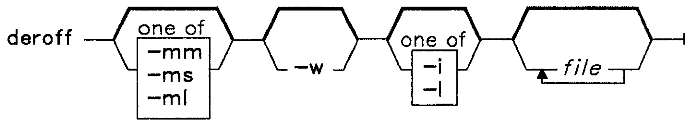
The discussion of Japanese Language Support in *Japanese Language Support User's Guide.*

deroff

Purpose

Removes **nroff, troff**, **troff**, **tbl**, and **eqn** constructs from files.

Syntax



OL805181

Description

The **deroff** command reads *file*s (standard input by default), removes all **troff** requests, macro calls, backslash constructs, **eqn** constructs (between **.EQ** and **.EN** lines and between delimiters), and **tbl** descriptions (perhaps replacing them with blanks or blank lines), and writes the remainder of the file to standard output.

The **deroff** command normally follows chains of included files (**.so** and **.nx troff** commands). If a file has already been included, a **.so** naming it is ignored and a **.nx** naming that file ends execution.

Notes:

1. **deroff** is not a complete **troff** interpreter, so it can be confused by subtle constructs. Most errors result in too much rather than too little output.
2. The **-ml** flag does not handle nested lists correctly.

Flags

-i	Suppresses the processing of included files.
-l	Suppresses the processing of included files whose names begin with **/usr/lib**, such as macro files in **/usr/lib/tmac**.
-mm	Ignores MM macros in text so that only running text is output (no text from macro lines is included).
-ml	Ignores MM macros in text (**-mm**) and also deletes MM list structures.

-ms	Ignores MS macros in text.
-w	Makes the output a word list, with one word per line and all other characters deleted. In text, a word is any string that contains at least two letters and is composed of letters, digits, & (ampersands), and ' (apostrophes). In a macro call, a word is a string that begins with at least two letters and contains a total of at least three letters. Delimiters are any characters other than letters, digits, apostrophes, and ampersands. Trailing apostrophes and ampersands are removed from words.

Related Information

The following commands: "**eqn**, **neqn**, **checkeq**" on page 395, "**nroff, troff**" on page 709, "**tbl**" on page 1053, and "**troff**" on page 710.

devices

Purpose

Adds, deletes, changes, and displays device information.

Syntax

```
devices ──┤
```

OL805306

Description

The **devices** command lets you add, delete, change, or examine information about devices on the system. To use **devices** you must be a member of the system group or have superuser authority.

The **devices** command is an interactive, menu-driven program. For information on how to use it, see *Installing and Customizing the AIX Operating System.* When auditing is enabled for your system, audit records are created. When a device is added or deleted, the audit record is of the type **devices - add** or **devices - del**. When a device is changed, the audit record is of the type **stanza - add** or **stanza - del**.

Files

/etc/filesystems
/etc/predefined
/etc/master
/etc/system
/etc/ports
/etc/qconfig
/tmp/CONFIGREPORT

Related Information

The discussion of **devices** in *Installing and Customizing the AIX Operating System.*

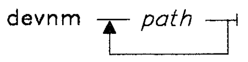
The following command: "**minidisks**" on page 650.

devnm

Purpose

Names a device.

Syntax



OL805114

Description

The **devnm** command reads *path*, identifies the special file associated with the mounted file system where *path* resides, and writes the special file name to standard output. Each *path* must be a full path name.

The most common use of the **devnm** command is by **/etc/rc** to construct a mount table entry for the root device.

Note: This command is for local file systems only.

Examples

1. To identify the device on which a file resides:

   ```
   devnm  /diskette0/bob/textfile
   ```

   This displays the name of the special device file on which `/diskette0/bob/textfile` resides. If a diskette is mounted as **/diskette0**, then **devnm** displays:

   ```
   fd0  /diskette0/bob/textfile
   rfd0  /diskette0/bob/textfile
   ```

   This means that `/diskette0/bob/textfile` resides on the diskette drive **/dev/fd0**.

2. To identify the device on which a file system resides:

   ```
   devnm  /
   ```

   This displays the name of the device on which the root file system (/) resides. The following list appears on the screen:

   ```
   hd0  /
   ```

   This means that / resides on **/dev/hd0**.

Files

/dev	Directory.
/etc/mnttab	Table of mounted devices.

Related Information

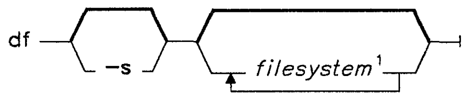
The following commands: "**rc**" on page 806 and "**setmnt**" on page 911.

df

Purpose

Reports number of available disk blocks.

Syntax



1 The default action is to provide information for each file system in **/etc/filesystems** with the attribute `free=true`.

OL805052

Description

The **df** command writes to standard output information about total space and available space on the specified file systems. *filesystem* can be the name of the device on which the file system resides or the directory on which it is mounted. If you do not specify *filesystem*, **df** provides information on all mounted file systems.

Normally, **df** uses free counts maintained in the superblock. Under certain exceptional circumstances, these counts may be in error.

If a file system is being actively modified at the instant **df** is run, the free count may be inaccurate.

Flag

-s This flag is for backwards compatibility only.

Examples

1. To list information about all file systems:

   ```
   df
   ```

   If your system is configured so that the /, **/usr**, **/u**, and **/tmp** directories reside in separate file systems, the output from the **df** command resembles this:

   ```
   Device        Mounted on       total     free  used      ifree  used
   /dev/hd0      /                19368     9976   48%       4714    5%
   /dev/hd1      /usr             24212     4808   80%       5031   19%
   /dev/hd2      /u                9744     9352    4%       1900    4%
   /dev/hd5      /tmp              3868     3856    0%        986    0%
   ```

   Note: On some remote file systems, such as NFS, columns are blank if the server does not provide the information.

2. To list information about the file system on a diskette:

   ```
   df /dev/fd0
   ```

3. To list information about the file system currently mounted as **/diskette0**:

   ```
   df /diskette0
   ```

Related Information

The following command: "**fsck**, **dfsck**" on page 445.

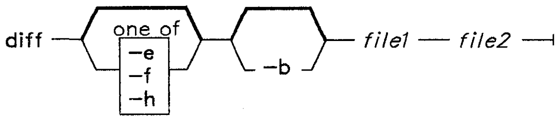
The discussion of **df** in *Managing the AIX Operating System.*

diff

Purpose

Compares text files.

Syntax



OL805046

Description

The **diff** command compares *file1* and *file2* and writes to standard output information about what changes must be made to bring them into agreement. If you specify a - (minus) for *file1* or *file2*, **diff** reads standard input. If *file1* is a directory, then **diff** uses a file in that directory with the name *file2*. If *file2* is a directory, then **diff** uses a file in that directory with the name *file1*.

The normal output contains lines of these forms:

Lines Affected in *file1*	**Action**	**Lines Affected in** *file2*
num1	**a**	*num2*[,*num3*]
num1[,*num2*]	**d**	*num3*
num1[,*num2*]	**c**	*num3*[,*num4*]

These lines resemble **ed** subcommands to convert *file1* into *file2*. The numbers before the action letters pertain to *file1*; those after pertain to *file2*. Thus, by exchanging **a** for **d** and reading backward, you can also tell how to convert *file2* into *file1*. As in **ed**, identical pairs (where *num1* = *num2*) are abbreviated as a single number.

Following each of these lines, **diff** displays all lines affected in the first file preceded by a <, then all lines affected in the second file preceded by a >.

Except in rare circumstances, **diff** finds a smallest sufficient set of file differences. An exit value of 0 indicates no differences, 1 indicates differences found, and 2 indicates an error.

Note: Editing scripts produced by the **-e** or **-f** flags cannot create lines consisting of a single **.** (period).

Flags

-b Ignores trailing spaces and tab characters and considers other strings of blanks to compare as equal.

-e Produces output in a form suitable for use with the **ed** command to convert *file1* to *file2*.

-f Produces output in a form not suitable for use with **ed**, showing the modifications necessary to convert *file1* to *file2* in the reverse order of that produced under the **-e** flag.

-h Performs a faster comparison. This flag only works when the changed sections are short and well separated, but it does work on files of any length. The **-e** and **-f** flags are not available when you use the **-h** flag.

Examples

1. To compare two files:

   ```
   diff chap1.bak chap1
   ```

   This displays the differences between the files `chap1.bak` and `chap1`.

2. To compare two files, ignoring differences in the amount of white space:

   ```
   diff  -b  prog.c.bak  prog.c
   ```

   If two lines differ only in the number of blanks and tabs between words, then **diff** considers them to be the same.

3. To create a file containing commands that **ed** can use to reconstruct one file from another:

   ```
   diff  -e  chap2  chap2.old  >new.to.old.ed
   ```

   This creates a file named `new.to.old.ed` that contains the **ed** commands to change `chap2` back into the version of the text found in `chap2.old`. In most cases, `new.to.old.ed` is a much smaller file than `chap2.old`. You can save disk space by deleting `chap2.old`, and you can reconstruct it at any time by entering:

   ```
   (cat new.to.old.ed ; echo  '1,$p') | ed  -  chap2  >chap2.old
   ```

   The commands in parentheses add `1,$p` to the end of the editing commands sent to **ed**. The `1,$p` causes **ed** to write the file to standard output after editing it. This modified command sequence is then piped to **ed** (`| ed`), and the editor reads it as standard input. The - flag causes **ed** not to display the file size and other extra information since it would be mixed with the text of `chap2.old`. See page 931 for details about grouping commands with parentheses.

Files

/tmp/d?????	Temporary files.
/usr/lib/diffh	For the **-h** flag.

Related Information

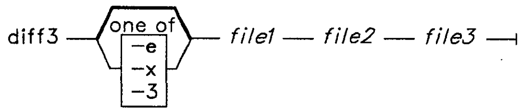
The following commands: "**bdiff**" on page 102 "**cmp**" on page 177, "**comm**" on page 183, "**ed**" on page 371, and "**sdiff**" on page 883.

diff3

Purpose

Compares three files.

Syntax



OL805053

Description

The **diff3** command reads three versions of a file and writes to standard output the ranges of text that differ, flagged with the following codes:

= = = =	All three files differ.
= = = =**1**	*file1* differs.
= = = =**2**	*file2* differs.
= = = =**3**	*file3* differs.

The type of change needed to convert a given range of a given file to match another file is indicated in one of these two ways in the output:

file : *n1* **a**	Text is to be added after line number *n1* in *file*, where *file* is **1**, **2**, or **3**.
file : *n1*[,*n2*] **c**	Text in the range line *n1* to line *n2* is to be changed. If *n1* = *n2*, the range may be abbreviated to *n1*.

The original contents of the range follows immediately after a **c** indication. When the contents of two files are identical, **diff3** does not show the contents of the lower-numbered file, although it shows the location of the identical lines for each.

Notes:

1. Editing scripts produced by the **-e** flag cannot create lines consisting only of a single period (.).
2. The **diff3** command does not work on files longer than 64K bytes.

Flags

-e Creates an edit script for use with the **ed** command to incorporate into *file1* all changes between *file2* and *file3* (that is, the changes that normally would be flagged ==== and ====3).

-x Produces an edit script to incorporate only changes flagged ====.

-3 Produces an edit script to incorporate only changes flagged ====3.

Example

To list the differences among three files:

```
diff3  fruit.a  fruit.b  fruit.c
```

If `fruit.a`, `fruit.b`, and `fruit.c` contain the following data:

fruit.a	**fruit.b**	**fruit.c**
banana grape kiwi lemon mango orange peach pare	apple banana grapefruit kiwi orange peach pear	grape grapefruit kiwi lemon mango orange peach pear

then the output from **diff3** shows the differences between these files as follows. (The comments on the right do not appear in the output.)

```
====     • All three files are different.
1:1,2c   - Lines 1 and 2 of the first file, fruit.a
  banana
  grape
2:1,3c   - Lines 1 through 3 of fruit.b
  apple
  banana
  grapefruit
3:1,2c   - Lines 1 and 2 of fruit.c
  grape
  grapefruit
====2    • The second file, fruit.b, is different.
```

```
1:4,5c    - Lines 4 and 5 are the same in fruit.a and fruit.c.
2:4a      - To make fruit.b look the same, add text after line 4.
3:4,5c
  lemon
  mango
====1   • The first file, fruit.a, is different.
1:8c
  pare
2:7c      - Line 7 of fruit.b and line 8 of fruit.c are the same.
3:8c
  pear
```

Files

/tmp/d3*
/usr/lib/diff3prog

Related Information

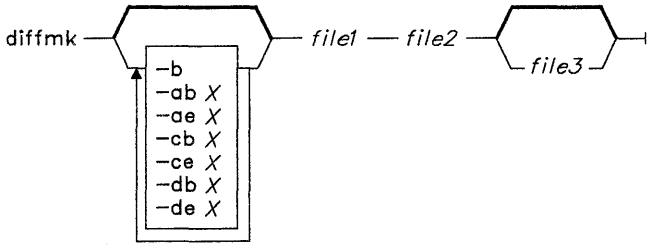
The following command: "**diff**" on page 320.

diffmk

Purpose

Marks differences between files.

Syntax



OL805057

Description

The **diffmk** command compares *file1* and *file2* and creates a third file that includes ***change mark commands*** for the **nroff** and **troff** commands. *file1* and *file2* are the old and new versions of the file. **diffmk** writes the newly created file to *file3*, if specified, or to standard output. This file contains the lines of *file2* with formatter change mark (**.mc**) requests inserted as appropriate. When *file3* is formatted, the changed or inserted text is marked by a | (vertical bar) at the right margin of each line. An * (asterisk) in the margin indicates that a line was deleted.

If the environment parameter DIFFMARK is defined, it names a command string that **diffmk** uses to compare the files. (Normally, **diffmk** uses the **diff** command.) For example, you might set DIFFMARK to `diff -h` in order to better handle extremely large files.

Flags

-ab*X*	Uses *X* to mark where added lines begin.
-ae*X*	Uses *X* to mark where added lines end.
-b	Ignores differences that are only changes in tabs or spaces on a line.
-cb*X*	Uses *X* to mark where changed lines begin.

-ce*X* Uses *X* to mark where changed lines end.

-db*X* Uses *X* to mark where deleted lines begin.

-de*X* Uses *X* to mark where deleted lines end.

Examples

1. To mark the differences between two versions of a text file:

   ```
   diffmk  chapl.old  chapl  > chapl.nroff
   ```

   This produces a copy of `chapl` containing **nroff/troff** change mark commands to identify text that has been added to, changed in, or deleted from `chapl.old`. This copy is saved in the file `chapl.nroff`.

2. To mark differences with non-**nroff/troff** messages:

   ```
   diffmk -ab'>>New:' -ae'<<End New' chapl.old  chapl  >chapl.nroff
   ```

   This causes **diffmk** to write `>>New:` on the line before a section of new lines that have been added to `chapl` and to write `<<End New` on the line following the added lines. Changes and deletions still generate **nroff/troff** commands to put a | or * in the margin.

3. To use different **nroff/troff** marking commands and ignore changes in white space:

   ```
   diffmk  -b  -cb'.mc  %'  chapl.old  chapl  > chapl.nroff
   ```

   This imbeds commands that mark changes with %, additions with |, and deletions with *. It does not mark changes that only involve a different number of spaces or tabs between words (`-b`).

Related Information

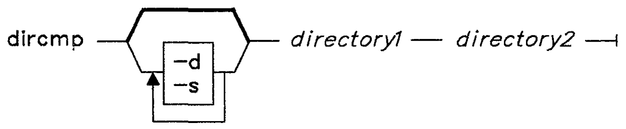
The following commands: "**diff**" on page 320, "**nroff, troff**" on page 709, and "**troff**" on page 710.

dircmp

Purpose

Compares two directories and the contents of their common files.

Syntax



OL805004

Description

The **dircmp** command reads *directory1* and *directory2* and writes information about their contents to standard output. First, **dircmp** compares the file names in each directory. When the same file name appears in both, **dircmp** compares the contents of both files.

In the output, **dircmp** lists the files unique to each directory. It then lists the files with identical names in both directories, but with different contents. With no flag, it also lists files that have identical contents as well as identical names in both directories.

Flags

-d Displays for each common file name both versions of the differing file lines. The display format is the same as that of "**diff**" on page 320.

-s Does not list the names of identical files.

Examples

1. To summarize the differences between the files in two directories:

   ```
   dircmp  proj.ver1  proj.ver2
   ```

   This displays a summary of the differences between the directories `proj.ver1` and `proj.ver2`. The summary lists separately the files found only in one directory or the other, and those found in both. If a file is found in both directories, **dircmp** notes whether or not the two copies are identical.

2. To show the details of the differences between files:

   ```
   dircmp  -d  -s  proj.ver1  proj.ver2
   ```

   The **-s** flag suppresses information about identical files. The **-d** flag displays a **diff** listing for each of the differing files found in both directories.

Related Information

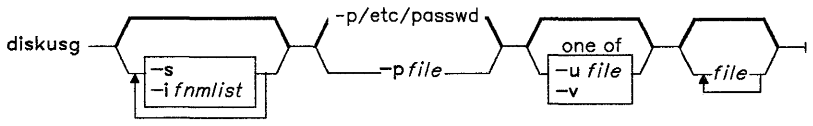
The following commands: "**cmp**" on page 177 and "**diff**" on page 320.

diskusg

Purpose

Generates disk accounting data by user ID.

Syntax



OL805402

Description

The **diskusg** command generates intermediate disk accounting information from data in *file*s or from standard input if you do not specify any files. **diskusg** writes lines to standard output, one per user, in the following format:

uid login #blocks

where:

uid	Is the numerical user ID of the user
login	Is the login name of the user; and
#blocks	Is the total number of disk blocks allocated to this user.

The **diskusg** command normally reads only the i-nodes of file systems for disk accounting. In this case, *file*s are the special file names of these devices.

Note: This command is for local devices only.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Flags

-i *fnmlist* Ignores the data on those file systems with a file system name in *fnmlist*. *fnmlist* is a list of file system names separated by commas or enclosed within quotation marks. **diskusg** compares each name in this list with the file system name stored in the volume ID.

-p *file*	Uses *file* as the name of the password file to generate login names. **/etc/passwd** is used by default.
-s	Combines all lines for a single user into a single line. (The input date is already in **diskusg** output format.)
-u *file*	Writes records to *file* of files that are charged to no one. Records consist of the special file name, the i-node number, and the user ID.
-v	Writes a list to standard error of all files that are charged to no one.

The output of **diskusg** is normally the input to **acctdisk**, which generates total accounting records that can be merged with other accounting records. **diskusg** is normally run in **dodisk** (see "**acct/***" on page 13).

Examples

The following will generate daily disk accounting information:

```
for  i  in  /dev/hd0  /dev/hd1  /dev/hd2  /dev/hd3
do
      diskusg  $i  > dtmp.'basename $i'  &
done
wait
diskusg  -s  dtmp.*  ¦ sort  +0n  +1  ¦  acctdisk  > dacct
```

Files

/etc/passwd	Used for user ID to login name conversions.

Related Information

The following commands: "**acct/***" on page 13, "**acctcms**" on page 18, "**acctcom**" on page 20, "**acctcon**" on page 24, "**acctmerg**" on page 28, "**acctprc**" on page 30, "**fwtmp**" on page 457, and "**runacct**" on page 848.

The **acct** system call and the **acct** and **utmp** files in *AIX Operating System Technical Reference.*

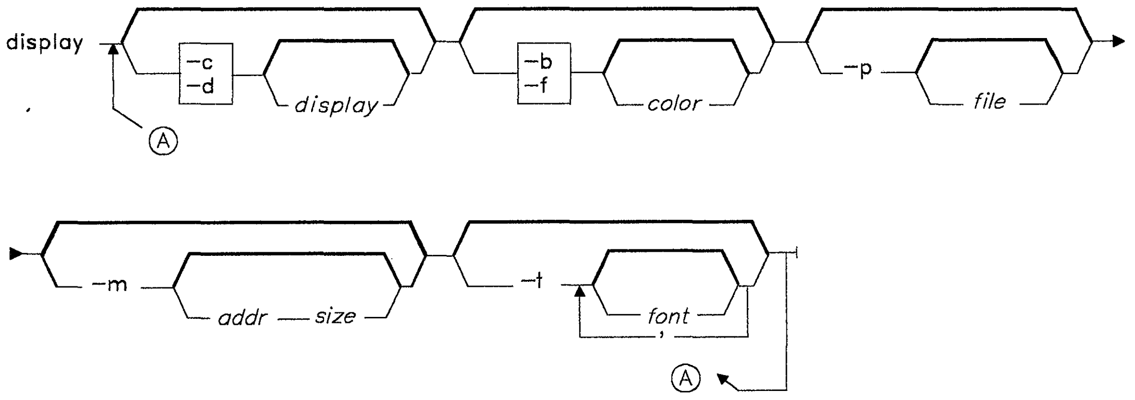
The discussion of accounting in *Managing the AIX Operating System.*

display

Purpose

Selects the physical display that an existing or new virtual terminal uses and sets colors and fonts.

Syntax



OL805442

Description

The **display** command changes the physical display assigned to the current virtual terminal or assigns a default display to be used when you open a virtual terminal. It also sets the foreground and background colors, the active color palette, and the active and alternate fonts on the current display. The *display* parameter can be one of the following names:

pcmono	PC Monochrome Adapter and Display
egamono	Enhanced Graphics Adapter and PC Monochrome Display
egacol	Enhanced Graphics Adapter and Display
advmono	Advanced Monochrome Graphics Adapter and Display
advcol	Advanced Color Graphics Adapter and Display
extmono	Extended Monochrome Graphics Adapter and Display
megapel	IBM Megapel Display Adapter and IBM 5081 Display Models 16 and 19.

You can request only those displays that are actually installed on the system. If you have more than four, only four will be displayed on the **-c** and **-d** menus. Before **display** makes any changes, it checks all arguments for errors and, if it encounters one, displays a list of valid arguments and exits.

Note: You must insure that the **TERM** shell variable contains the proper value for whatever the current display is. See "**termdef**" on page 1062 and the **terminfo** file in *AIX Operating System Technical Reference* for a list of these values.

Flags

-b [*color*]	Selects the background color. The *color* parameter is an integer from 1 to 8 for the Enhanced Graphics Display and from 1 to 16 for other color graphics displays. These values correspond to the first eight or sixteen entries in the active color palette (see the **-p** flag). For example, `-b 5` selects the fifth entry. If you do not specify a number, **display** lists the palette of active background colors and prompts you to select a number for the new background color.
-c [*display*]	Changes the display used by the current virtual terminal. If you do not specify a *display*, you are given a menu of available options. This menu consists of a numbered list of display names and descriptions. The display number reflects the number of physical displays installed and their relative positions in the Real Screen Table. The current default is always display number 1 in this list. Changing the default alters the display number associated with each physical display. If the virtual terminal does not know the display/adapter combination, the Name column will contain the words `Unknown Display` or `?????`. A prompt at the bottom of the display list asks you to enter the new display number for the current or default display setting. Whenever you change the current display, the screen of that display clears.
-d [*display*]	Changes the default display used when a virtual terminal is opened. If you do not specify a *display*, you are given a menu of available options (see the **-c** flag).
-f [*color*]	Selects the foreground color. The *color* parameter is an integer from 1 to 16. These values correspond to the first sixteen entries in the active color palette (see the **-p** flag). If you do not specify a color number, **display** lists the palette of active foreground colors and prompts you to select a number for the new foreground color.
-m [*addr size*]	Changes the DMA pinned page at the specified starting address to *size* 256K blocks. If you do not specify an address and a size, the current starting address and size is displayed.

-p [*file*] Changes the active color palette. The optional *file* parameter is the full path name to a file that contains a list of colors for the current display, one color per line, where each color is the decimal representation of the 32-bit color value. The color palette file can also contain blank lines and comment lines (a comment line must begin with a * character in column one). Each supported display has a corresponding color file which contains its default active color palette. The name of this file is **/etc/vtm/pal.***name* where *name* is the display name described on page 332. This is the default value for the *file* parameter.

-t [*font*[,*font*] . . .] Selects the primary and active alternate fonts for the current virtual terminal on the current display. The first *font* named in the optional list following **-t** will be the primary font. The remaining fonts will be alternates, in the order listed, for the active font table. If you do not specify eight font IDs, the first font will be used to fill out in the remaining entries in the active font table.

Notes:

1. All of the fonts in the list must be of the same size.
2. Some applications that use the **terminfo** file expect the italic font to be the first alternate and the bold font to be the second alternate fonts (see the **terminfo** file in *AIX Operating System Technical Reference* for more information).

If you do not specify any fonts, all of the fonts available for the current display will be listed, and you will be prompted first for the desired primary font ID and then for alternate font IDs until you enter `F`. As you enter alternate fonts, the **display** command checks that they are the same size as the new primary font. If you enter fewer than eight fonts, the primary font will be repeated in the remaining entries of the active font table.

You can specify combinations of the same flags on a single command line. **display** processes **-c** and **-d** flags first. If you specify **-c**, you will see the message `Changing to current display...`, and the current display will be changed. Any menu interface for the color or font parameters will be displayed there. A **-p** flag will be processed next. The screen will be immediately redrawn with the colors from the new color palette. Then any foreground, background, or font flags will be processed.

Examples

1. To change the current virtual terminal display:

```
display -c egamono
```

This changes the display to the Enhanced Graphics Adapter and PC Monochrome Display.

2. To make the Advanced Color Graphics Display the default virtual terminal display:

   ```
   display -d advcol
   ```

3. To change both the current and the default displays:

   ```
   display -c pcmono -d egacol
   ```

   This makes the PC Monochrome Adapter and Display the current display and makes the Enhanced Graphics Adapter and Display the default display.

4. To change the active color palette for the current display:

   ```
   display -p /u/new/palette
   ```

Related Information

The following commands: "**open**" on page 728 and "**termdef**" on page 1062.

The **terminfo** file in *AIX Operating System Technical Reference.*

"Using Display Station Features" in *IBM RT Using the AIX Operating System* and "Managing Display Station Features" in *IBM RT Managing the AIX Operating System.*

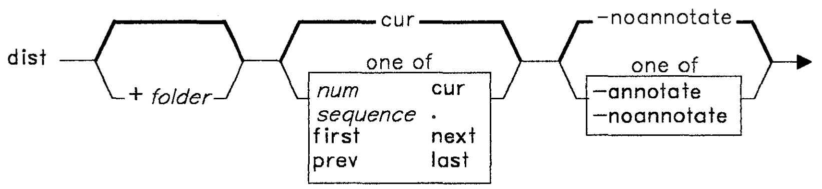
The default color palettes in *Virtual Resource Manager Technical Reference.*

dist

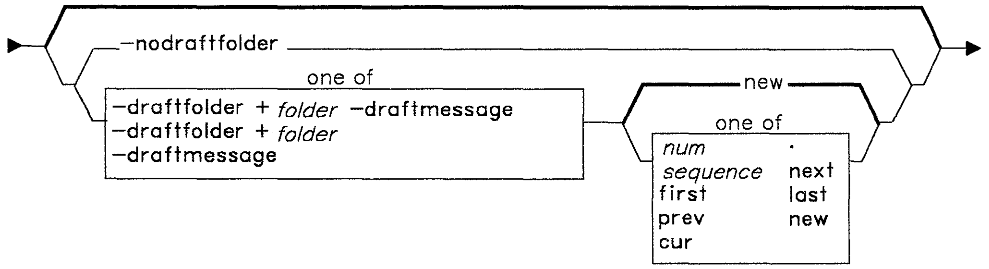
Purpose

Redistributes a message to additional addresses.

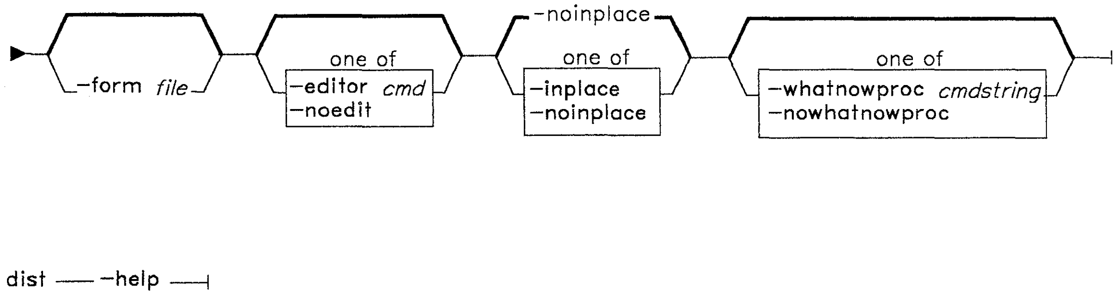
Syntax



AJ2FL242



AJ2FL157



AJ2FL243

Description

The **dist** command is used to redistribute messages to a new list of addresses. **dist** is part of the MH (Message Handling) package and can be used with other MH and AIX commands.

By default, **dist** copies a message form to a new draft message and invokes an editor. You can then fill in the message header fields **Resent-To:** and **Subject:** and fill in or delete the other header fields (such as **Resent-cc:** and **Resent-Bcc:**). Since the body of the message will be the message you are redistributing, do not fill in the body. **dist** does not automatically display the body of the message. When you exit the editor, the **dist** command invokes the MH command **whatnow**. You can press **Enter** to see a list of the available **whatnow** subcommands. These subcommands enable you to continue editing the message header, list the message header, direct the disposition of the message, or end the processing of the **dist** command. "**whatnow**" on page 1215 describes the subcommands.

When you send the draft message, the recipients are sent the headers and body of the original message appended to the new message. **dist** does not automatically store a copy of the original message with the new draft message. The draft message you create using the **dist** command consists of header fields only.

You can specify the message that you want to distribute by using the *+folder msg* flag. If you do not specify a message, **dist** redistributes the current message.

You can specify the format, of the message header by using the **-form** flag. If you do not specify this flag, **dist** uses your default message format located in the file *user_mh_directory*/**distcomps**. If this file does not exist, **dist** uses the system default message format located in **/usr/lib/mh/distcomps**. **dist** prepends the form to the message being redistributed.

Note: The line of dashes or a blank line must be left between the header and the body of the message for the message to be identified when it is sent.

Flags

-annotate Annotates the message being redistributed with the lines:

Resent: *date*
Resent: *addrs*

The annotation appears in the original draft message so that you can maintain a complete list of recipients with the original message. If you do not actually redistribute the message using the immediate **dist** command, the **-annotate** flag may fail to provide annotation. The **-inplace** flag forces annotation to be done in place.

-draftfolder +*folder*	Places the draft message in the specified folder. If you do not specify this flag, **dist** selects a default draft folder according to the information supplied in the MH profiles. You can define a default draft folder in **$HOME/.mh_profile**. If **-draftfolder** +*folder* is followed by *msg*, *msg* represents the **-draftmessage** attribute.
-draftmessage *msg*	Specifies the draft message. You can specify one of the following message references as *msg*:

num	*sequence*	**first**
prev	**cur**	**.**
next	**last**	**new**

The default draft message is **new**. If you specify a draft message, that message becomes the current message.

-editor *cmd*	Specifies that *cmd* is the initial editor for preparing the message for distribution. If you do not specify this flag, **dist** selects a default editor or suppresses the initial edit, according to the information supplied in the MH profiles. You can define a default initial editor in **$HOME/.mh_profile**.
+*folder msg*	Redistributes the specified message in the specified folder. You can specify one of the following message references as *msg*:

num	*sequence*	**first**
prev	**cur**	**.**
next	**last**	

The default message is the current message in the current folder. If you specify a folder, that folder becomes the current folder.

-form *file*	Prepends the form contained in the specified file to the message being resent. **dist** treats each line in *file* as a format string.
-help	Displays help information for the command.
-inplace	Forces annotation to be done in place in order to preserve links to the annotated message.
-noannotate	Does not annotate the message. This flag is the default.
-nodraftfolder	Places the draft in the file *user_mh_directory*/**draft**.
-noedit	Suppresses the initial edit.
-noinplace	Does not perform annotation in place. This flag is the default.

-nowhatnowproc	Does not invoke a program that guides you through the distribution tasks. The **-nowhatnowproc** flag also prevents any edit from occurring.
-whatnowproc *cmdstring*	Invokes *cmdstring* as the program to guide you through the distribution tasks. See "**whatnow**" on page 1215 for information about the default **whatnow** program and its subcommands. **Note:** If you specify `whatnow` for *cmdstring*, **dist** invokes an internal **whatnow** procedure rather than a program with the file name **whatnow**.

Profile Entries

Current-Folder:	Sets your default current folder.
Draft-Folder:	Sets your default folder for drafts.
Editor:	Sets your default initial editor.
fileproc:	Specifies the program used to refile messages.
Path:	Specifies your *user_mh_directory*.
whatnowproc:	Specifies the program used to prompt `What now?` questions.

Files

/usr/lib/mh/distcomps	The system default message skeleton
user_mh_directory/distcomps	The user's default message skeleton. (If it exists, it overrides the system default message skeleton.)
$HOME/.mh_profile	The MH user profile.
user_mh_directory/draft	The draft file.

Related Information

Other MH commands: "**ali**" on page 48, "**anno**" on page 50, "**comp**" on page 185, "**forw**" on page 438, "**prompter**" on page 778, "**refile**" on page 817, "**repl**" on page 821, "**send**" on page 893, "**whatnow**" on page 1215.

The **mh-alias**, **mh-format**, **mh-mail**, and **mh-profile** files in *AIX Operating System Technical Reference.*

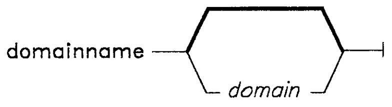
"Overview of the Message Handling Package" in *Managing the AIX Operating System.*

domainname

Purpose

Sets or displays the name of the current Yellow Pages (YP) domain.

Syntax



OL805481

Description

The **domainname** command displays the name of the current YP domain. If you have superuser authority, you can also use this command to set the name of the domain.

A ***domain*** is a group of host machines in the network. The name of the domain typically is set in **/etc/rc.nfs** file.

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

File

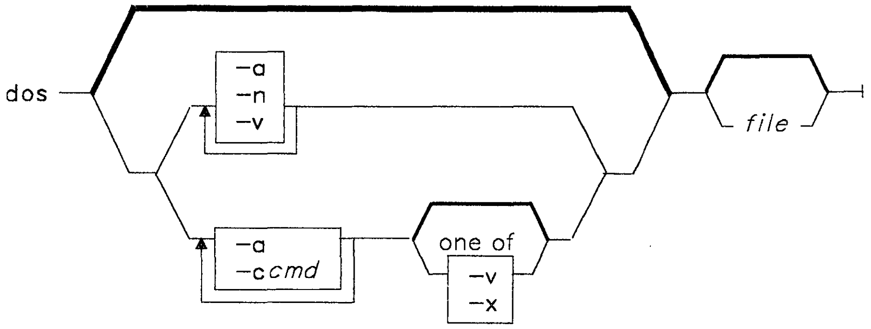
/etc/rc.nfs	The NFS startup shell script.

dos

Purpose

Starts **shell**.

Syntax



OL805330

Description

The **dos** command starts a DOS emulation environment. It interprets DOS commands and runs programs that can use the routines that simulate DOS run-time behavior. (For more information on these routines and this environment, see *DOS Services Reference* and *Installing and Customizing the AIX Operating System.*)

When you enter **dos**, a DOS environment file is created from the process environment. (For details on how this is done, see **dosinit** in *AIX Operating System Technical Reference.*) Upon invocation, **dos** sets the current drive to **A** or the first valid drive. The environment variable **DOSDISK** can be set to define the default current drive (**B**, **C**, **D**, and so on).

The *file* parameter specifies a **dos** batch file to be run. *file* must have the extension **.bat** or **.BAT**.

If the current DOS Services directory contains the batch file **autoexec.bat** or **AUTOEXEC.BAT**, then DOS Services initially reads and runs commands from this file.

DOS commands are either built-in (to the **dos** command itself), or they are external. External commands reside in the **/usr/dos/bin** directory. Normally, the search order for commands that you enter is as follows:

- The directory **/usr/dos/bin**
- The working directory
- Each directory in the **dos** path.

When you enter a command, **dos** searches each directory for a file with a name composed of the command name and either the extension **.BAT**, the extension **.bat**, or no extension. If the file has the extension **.BAT** or **.bat**, it runs as a batch file. Otherwise, it runs as an AIX program. If it is an AIX program, it can be either a compiled program or a shell file. In either case you must have execute access to it.

The **dos** command supports two types of file systems: AIX file systems and DOS file systems. Each **dos** minidisk can contain either an AIX-formatted file system or a DOS-formatted file system. However, diskette drives (such as **/dev/fd0**) may contain only DOS-formatted file systems, unless the device is mounted as an AIX file system before you invoke **dos**.

Warning: Only one user or process at a time can access a **dos** file system. If a **dos** file system resides on a minidisk, two or more users may attempt to access the minidisk at the same time. Because **dos** has no way to warn you that another process is using a minidisk, you should allocate minidisks containing **dos** file systems on a per-user basis.

If a coprocessor on the system accesses a **dos**-formatted minidisk at the same time as an RT process, there is no conflict because only the first process has read/write privileges. Subsequent opens at the device level are limited to read-only access.

There are different restrictions for file names on DOS drives and AIX drives. For DOS Services drives:

- File names cannot be longer than 12 characters.
- The name is always stored in uppercase.
- All files in the directory must have unique names.
- There can be only one period in a file name.

For AIX file systems:

- File names cannot be longer than 14 characters.
- Names may contain either uppercase or lowercase letters.
- Two files in the same directory can have the same name if the letter case is different.
- There can be more than one period in a file name.
- All files in the directory must have unique names.

On AIX drives, file names that begin with a period specify hidden files. On DOS Services drives, hidden files have a bit set in the attribute byte of the file directory.

There are differences between AIX and DOS Services file formats. AIX ASCII files and DOS Services ASCII files are similar and can be converted from one format to the other. Two new commands, **FILETYPE** and **CONVERT**, are available for detecting and changing a file format.

DOS Services Commands and Programs

There are several differences between the set of supported DOS Services commands and DOS commands.

Unsupported DOS Commands and Programs

You can use all of the standard DOS commands except **BREAK**, **CTTY**, **EDLIN**, **EXE2BIN**, **GRAPHICS**, and **SYS**.

Modified DOS Commands

The following DOS Services commands behave differently than the corresponding standard DOS commands:

backup	The **/M** parameter is not valid for DOS Services file systems.
chdir	Unlike DOS, DOS Services may not allow you to change to the highest directory in the file system.
date	This command lets only the superuser change the date.
dir	Does not list file-name extensions in a separate column when executed on an AIX drive.
format	The **/B** is not supported. Two additional flags, **/U** and **/H** are supported. Use the **/U** flag to format a AIX diskette. Use the **/H** flag to format a fixed disk to contain DOS Services file systems in a single partition. **Note:** The **format** command makes use of the **mksf** command, which in turn uses the **/etc/filesystems** file. If you modify this file, it will affect the **format** command.
label	On an AIX-formatted drive, the label is written to a file called **LABEL.VOL**. Reading a label is accomplished by reading this file. Changing a label modifies the contents of this file. **Note:** The command **del *.*** deletes the volume label.
mode	Only option 3 (for an asynchronous communications adapter) is supported.
print	The DOS Services version does not ask you which device to store the print queue on. This information is set up in your user profile. The **/B**, **D**, **M**, **/S**, **/Q**, and **/U** configuration flags are not supported.

set	A **/U** flag lets you display the AIX environment as it is inherited by the **dos** command. You can change the environment variables internal to **dos**. When you exit from **dos**, the environment variables remain unchanged.
time	Allows only the superuser to change the time.

Additional Commands

In addition to DOS commands, the following commands are available:

COMMAND	The new flags which have been added to **dos** also apply to this command.
CONVERT	Converts a DOS format ASCII file to an AIX format ASCII file or an AIX format ASCII file to a DOS format ASCII file.
ed	Starts the line editor.
EXIT	Ends DOS Services. You can also use END OF FILE (**Ctrl-D**).
FILETYPE	Attempts to determine the format (AIX or DOS) and contents of the specified file.
shutdown	Provides for an orderly exit from the system.

Flags

-a	Does not run the **AUTOEXEC.BAT** file.
-c *cmd*	Runs the specified command.
-n	Reads commands but does not run them.
-v	Displays the commands and their flags as they are read.
-x	Displays the commands and their flags as they are run.

Files

/usr/dos/bin/*	DOS Services external commands.
AUTOEXEC.BAT	Batch file that can run commands automatically.
autoexec.bat	Batch file that can run commands automatically.

Related Information

The following commands: "**dosdel**" on page 345, "**dosread**" on page 348 and "**doswrite**" on page 350.

The **dosinit** subroutine in *AIX Operating System Technical Reference.*

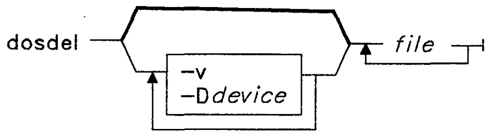
The discussion of **dos** in *Using DOS Services* and *DOS Services Reference.*

dosdel

Purpose

Deletes DOS files.

Syntax



OL805108

Description

The **dosdel** command deletes the DOS file specified by *file*. Use the **-v** flag to obtain format information about the disk.

File-naming conventions are those of DOS, with one exception. **doswrite** replaces the \ (backslash) character used to separate components of a DOS path name with the / (slash) because the backslash can have special meaning to AIX. **dosdel** converts lowercase characters in the *file1* name to uppercase before it checks the disk. Because all file names are assumed to be full (not relative) path names, you need not add the initial / (slash).

Flags

-D *device*	Specifies a device or file system to use as the DOS disk. If you do not specify this flag the default device is **/dev/fd0**.
-v	Writes format information about the disk. Use primarily to verify the identify of a disk or file system as a DOS disk.

Related Information

The following commands: "**dos**" on page 341, "**dosdir**" on page 346, "**dosread**" on page 348, and "**doswrite**" on page 350.

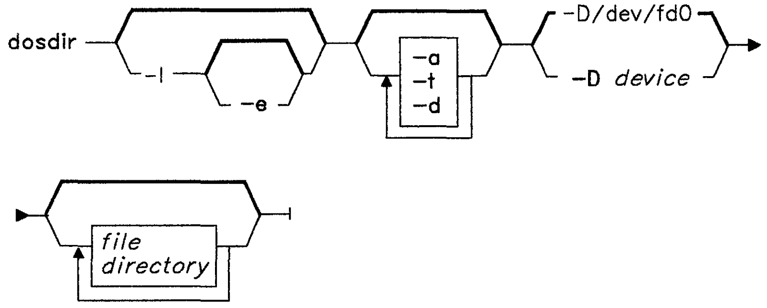
The **pcdos** subroutine in *AIX Operating System Technical Reference.*

dosdir

Purpose

Lists the directory for DOS files.

Syntax



OL805358

Description

The **dosdir** command displays information about the specified DOS file or directory (the current directory by default). If you specify a directory without also specifying the **-d** flag, **dosdir** displays information about the files in that directory.

File-naming conventions are those of DOS, with one exception. **dosdir** replaces the \ (backslash) character used to separate components of a DOS path name with a / (slash) because the backslash can have special meaning to the AIX Operating System. **dosdir** converts lowercase characters in the file or directory name to uppercase before it checks the disk. Because all file names are assumed to be full (not relative) path names, you need not add the initial / (slash).

Flags

-a	Writes information about all files. This includes hidden and system files as well as the . (dot) and .. (dot dot) files.
-d	Treats *file* as a file, even if it is a directory. If a directory is specified, information about the directory is listed rather than information about the files it contains.

-D [*device*]	Specifies a device or file system to use as the DOS disk. If you do not specify this flag the default device is **/dev/fd0**.
-e	Uses the **-l** flag to write the list of clusters allocated to the file.
-l	Produces a long list that includes the creation date, size in bytes, and attributes. The size of a subdirectory is specified as 0 bytes. The attributes have the following meanings:

A	Archive - the file has not been backed up since it was last modified.
D	Directory - the file is a subdirectory, and is not included in the normal DOS directory search.
H	Hidden - the file is not included in the normal DOS directory search.
R	Read-only - the file cannot be modified.
S	System - the file is a system file, and is not included in the normal DOS directory search.

-t	Lists the entire directory tree starting at the named directory.
-v	Writes information about the format of the disk.

Related Information

The following commands: "**dosdel**" on page 345, "**dosread**" on page 348, and "**doswrite**" on page 350.

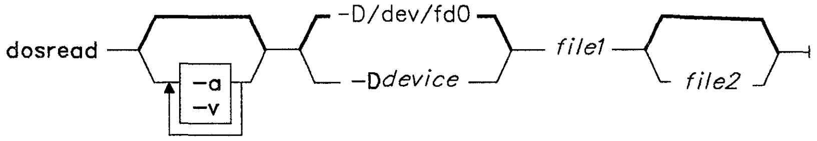
The **pcdos** subroutine in *AIX Operating System Technical Reference.*

dosread

Purpose

Copies a DOS file.

Syntax



OL805111

Description

The **dosread** command copies the specified DOS *file1* to standard output or to the specified AIX *file2* (by default the root directory). Unless otherwise specified, **dosread** copies as many bytes as are specified in the directory entry for *file1*. This means, in particular, that copying directories does not work, since directories by convention have a record size of 0.

File-naming conventions are those of DOS, with one exception. **dosread** replaces the \ (backslash) character used to separate components of a DOS path name with a / (slash) because the backslash can have special meaning to the AIX Operating System. **dosread** converts lowercase characters in the *file1* name to uppercase before it checks the disk. Because all file names are assumed to be full (not relative) path names, you need not add the initial / (slash).

Notes:

1. Wild card characters (* and ?) are not treated in a special way by this command (although they are by the shell). If, for example, you do not specify a file-name extension, the file name is matched as if you had specified a blank extension.
2. This command must be named **dosread**.

Flags

-a	Replaces the sequence **CR-LF** (carriage return-line feed) with **NL** (new-line character) and interprets a **Ctrl-Z** (ASCII SUB) as the end-of-file character.
-D *device*	Specifies the name of the DOS device or file system. The default *device* is **/dev/fd0**. This device must have the DOS-disk format.
-v	Writes information to the standard output about the format of the disk. Use this flag to verify that a device or file system is a DOS disk.

Examples

1. To copy a text file from a DOS diskette to the AIX file system:

   ```
   dosread -a chap1.doc chap1
   ```

   This copies the DOS text file `\CHAP1.DOC` on **/dev/fd0** to the AIX file `chap1` in the current directory.

2. To copy a binary file from a fixed-disk DOS file system to the AIX file system:

   ```
   dosread -D/dev/hd1 /survey/test.dta /u/fran/testdata
   ```

   This copies the DOS data file `\SURVEY\TEST.DTA` on **/dev/hd1** to the AIX file `/u/fran/testdata`.

Files

/dev/fd0	Device name for diskette drive.

Related Information

The following commands: "**dosdel**" on page 345, "**dosdir**" on page 346, and "**doswrite**" on page 350.

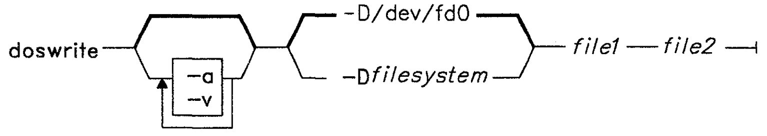
The **pcdos** subroutine in *AIX Operating System Technical Reference.*

doswrite

Purpose

Copies AIX files to DOS files.

Syntax



OL805112

Description

The **doswrite** command copies the specified AIX *file1* to the specified DOS *file2*. If *file2* contains a /, each intervening component must exist as a directory and the last component (the named file), must not exist.

File-naming conventions are those of DOS, with one exception. **doswrite** replaces the \ (backslash) character used to separate components of a DOS path name with the / (slash) because the backslash can have special meaning to AIX. **doswrite** converts lowercase characters in the *file1* name to uppercase before it checks the disk. Because all file names are assumed to be full (not relative) path names, you need not add the initial / (slash).

Notes:

1. Wild card characters (* and ?) are not treated in a special way by this command (although they are by the shell). If, for example, you do not specify a file-name extension, the file name is matched as if you had specified a blank extension.
2. This command must be named **doswrite**.

Flags

-a	Replaces **NL** (new-line characters) with the sequence **CR-LF** (carriage return-line feed). **Ctrl-Z** is added to the output at the end of file.
-D *filesystem*	Specifies the name of the DOS device or file system. The default *device* is **/dev/fd0**. This device must have the DOS-disk format.
-v	Writes information to the standard output about the format of the disk. Use this flag to verify that a device or file system is a DOS disk.

Examples

1. To copy a text file from the AIX file system to a DOS diskette:

   ```
   doswrite  -a  chap1  chap1.doc
   ```

   This copies the AIX file `chap1` in the current directory to the DOS text file `\CHAP1.DOC` on **/dev/fd0**.

2. To copy a binary file from the AIX file system to a fixed-disk DOS file system:

   ```
   doswrite  -D/dev/hd1  /u/fran/testdata  /survey/test.dta
   ```

   This copies the AIX data file `/u/fran/testdata` to the DOS file `\SURVEY\TEST.DTA` on **/dev/hd1**.

Files

/dev/fd0	Device name for diskette drive.

Related Information

The following commands: "**dosdir**" on page 346, "**dosread**" on page 348, and "**dosdel**" on page 345.

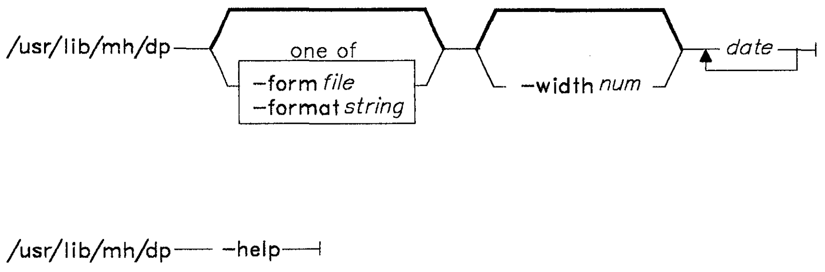
The **pcdos** subroutine in *AIX Operating System Technical Reference.*

dp

Purpose

Parses and reformats dates.

Syntax



AJ2FL227

Description

The **dp** command is used to parse and reformat dates. **dp** is not designed to be run directly by the user; it is designed to be called by other programs. The **dp** command is typically called by its full path name. The **dp** command is part of the MH (Message Handling) package.

The **dp** command parses each string specified as a date and attempts to reformat the string. The default output format for **dp** is the ARPA RFC822 standard. For each string it is unable to parse, **dp** displays an error message.

Flags

-form *file*	Reformats the given dates into the alternate format described in *file*.
-format *string*	Reformats the given dates into the alternate format specified by *string*. The default format string is: `%<(nodate{date})error:%{date}%|%(putstr(pretty{date}))%>`
-help	Displays help information for the command.
-width *num*	Sets the maximum number of columns that **dp** uses to display dates and error messages. The default is the width of the display.

Files

$HOME/.mh_profile	The MH user profile.

Related Information

The MH command "**ap**" on page 53.

The **mh-format** and **mh-profile** files in *AIX Operating System Technical Reference.*

"Overview of the Message Handling Package" in *Managing the AIX Operating System.*

dsipc

Purpose

Installs the Interprocess Communication key mapping in the kernel.

Syntax

```
dsipc——┤
```

OL805461

Description

The **dsipc** command replaces all IPC key mapping currently in the kernel with new mapping from the profile database. The **dsipc** command is usually called, at system startup time, by **/etc/rc.include** to update the Distributed Services kernel. To use **dsipc** command from the command line, you must be a member of the system group or have superuser authority (see "**su**" on page 1026).

Related Information

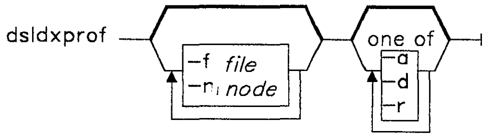
"Using Distributed Services" in *Managing the AIX Operating System.*

dsldxprof

Purpose

Loads translate information into the UID/GID translate profiles.

Syntax



OL805460

Description

The **dsldxprof** command loads translate information from a file into the UID/GID translate profiles. Each line in the file contains a row of translate information in the following format:

Usr/Grp_name U/G Local_id Outbound_id Inbound_id Originating_node

This format is the same as the translate information from the Network Users/Groups Table. You must specify the *U/G*, *Local_id*, and either the *Inbound_id* or *Outbound_id* fields. If you specify the *Inbound_id* field, the *Originating_node* field must also be specified. Data entered after the Originating Nickname/Node ID is treated as a comment and ignored. A - (hyphen) is placed in unused fields as a place holder.

The **dsldxprof** command reads a line of data from the file. If a line begins with a * (asterisk), it is a comment line. Comment lines are ignored and the next line is read. If the line is not a comment line, **dsldxprof** validates the data and loads the data into profiles. Translate rows are rejected due to improper syntax or incorrect values, or they may conflict with translate rows already in the profiles. A translate row is in conflict if there is an existing row in the profiles with a matching *U/G*, *Local_id*, *Inbound_id*, and *Originating_node*, or if there is an existing row in the profiles with a matching *U/G*, *Local_id*, and *Outbound_id*, or both. If there is conflict, you are prompted to replace or reject the conflicting row. Rejected rows are written to standard error along with the information on why they are rejected.

To delete a translate row from the profiles, precede an identical row in the file with ##.

To use **dsldxprof** command, you must be a member of the system group or have superuser authority (see "**su**" on page 1026).

Flags

-a	Places all rows that are in conflict into the profiles without prompting.
-d	Deletes the **pfsuidgid** profile and then recreates it without any entries. This option is handled before the **-f** option if both exist.
-f *filename*	Reads translate information from *filename.*
-n *nodename*	Updates translate profiles on the remote node *nodename.*
-r	Rejects all conflicting rows without prompting.

Files

pfsuidgid	Contains translate information.
.pfsuidgid	Contains indexes defined for **pfsuidgid**.

Related Information

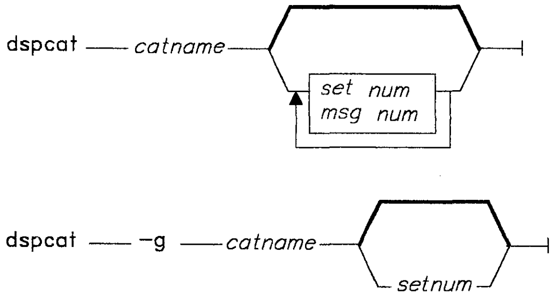
The discussion on distributed services (which includes information on the Network Users/Groups Table) and the discussion on using the **dsldxprof** command in *Managing the AIX Operating System.*

dspcat

Purpose

Displays all or part of a message catalog.

Syntax



OL805482

Description

Use **dspcat** to display a particular message, all of the messages in a set, or all of the messages in a catalog. The syntax for **dspcat** is:

`$ dspcat` *catname* [*set_num*] [*msg_num*]

catname specifies a message catalog, *set_num* specifies a set in the catalog, and *msg_num* specifies a particular message in the set. If you include all three parameters, **dspcat** displays a particular message. If you do not include *msg_num* or the *msg_* is in error, all the messages in the set are displayed. If you specify a nonexistent *set_num*, all messages in the catalog are displayed. If you specify only *catname*, all the messages in the catalog are displayed. You must include *set_num* if you include *msg_num*.

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Flags

-g	Formats the output so that it can be used as input to **gencat**.

Related Information

The following commands: "**dspmsg**" on page 359, "**gencat**" on page 470, "**mkcatdefs**" on page 651, and " **runcat**" on page 852.

The **catopen**, **catgets**, **catgetmsg**, **catclose**, **NLcatopen**, **NLcatgets**, and **NLgetamsg** files in *AIX Operating System Technical Reference.*

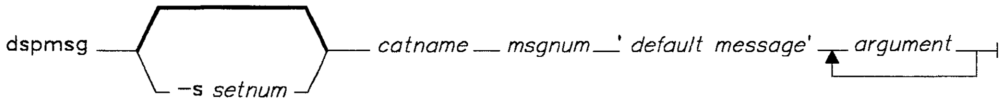
The discussion of **dspcat** in *AIX Operating System Programming Tools and Interfaces.*

dspmsg

Purpose

Displays a selected message from a message catalog.

Syntax



OL805483

Description

dspmsg displays a particular message from a catalog. It allows you to pass up to ten string arguments for substitution into the message if it contains the **printf** conversion specification **%s**, or the **NLprintf** conversion specification **%*n*$s**. The syntax for **dspmsg** is:

`$ dspmsg catname [-s` *set_num*`]` *msg_num* `['`*default-message*`'` `[`*args*`]]`

You must specify the catalog (*catname*) and the message (*msg_num*). The default set number is 1. Specify another set by using the **-s** flag followed by the set number.

If **dspmsg** cannot find the message, the *default-message* is displayed. You must enclose the default message in single quotes if you are using the **%*n*$s** notation for message inserts. If **dspmsg** cannot find the message, and you do not specify a default message, a system-generated error message is displayed.

Follow the default message with up to ten arguments to substitute into the catalog message (or the default message). Missing arguments for conversion specifications are replaced by null strings.

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Related Information

The following commands: "**dspcat**" on page 357, "**gencat**" on page 470, "**mkcatdefs**" on page 651, and " **runcat**" on page 852.

The **catopen**, **catgets**, **catgetamsg**, **catclose**, **NLcatopen** **NLcatgets**, and **NLgetamsg** files in *AIX Operating System Technical Reference.*

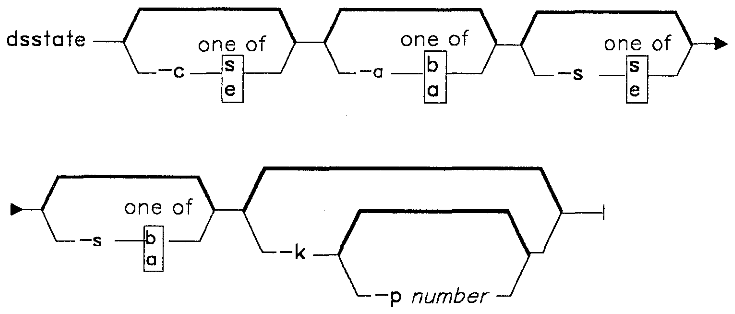
The discussion of **dspmsg** in *AIX Operating System Programming Tools and Interfaces.*

dsstate

Purpose

Sets the state of the Distributed Services kernel logic.

Syntax



OL805462

Description

The **dsstate** command changes the state of the Distributed Services kernel logic, including the number of kernel processes allocated for Distributed Services, whether incoming and outgoing remote requests are allowed, and where temporary storage takes place. Only members of the system group or users operating with superuser authority can use **dsstate** to change the state of the Distributed Services kernel logic (see "**su**" on page 1026). Other users can use **dsstate** with no flags to write to the standard output the current state of the Distributed Services kernel logic.

Flags

-c s	Starts client sync, which forces all files for which this node is the client to be written directly to the server, preventing caching (temporary storage) of the file contents at the client. Starting client sync often affects the performance of file operations, and is used primarily for certain system startup and shutdown routines.
-c e	Ends client sync and allows some data to be stored at the local node.
-a b	Breaks all connections with remote nodes and blocks new requests for remote file services.

-a a	Allows requests from this client node for remote file services.
-s s	Starts server sync, which forces all files for which this node is the server to be written directly to the server, preventing caching (temporary storage) of the file contents at the client node. Starting server sync often affects the performance of file operations, and is used primarily for certain system startup and shutdown routines.
-s e	Ends server sync and allows some data to be stored at the client node.
-s b	Blocks all requests for file services from other nodes, including both new requests and requests for files already in use.
-s a	Allows this server to accept requests for file services from other nodes.
-k	Starts the Distributed Services kernel processes.
-p *number*	Sets the number of active Distributed Services kernel processes to *number*. If *number* is greater than the number of kernel processes allocated for Distributed Services, then those that are available are activated. If *number* is 0 or a negative value, the number of kernel processes is not changed. By adjusting the number of active Distributed Services kernel processes, the rate at which services are provided to remote nodes can be varied. Lowering the number of active Distributed Services kernel processes lowers remote use of this node's processor, leaving more system resources for local use. **Note:** The Distributed Services kernel processes must have been started with a **-k** flag on either this **dsstate** command or an earlier **dsstate** command.

Related Information

The **dsstate** system call in *AIX Operating System Technical Reference.*

"Using Distributed Services" in *Managing the AIX Operating System.*

dsxlate

Purpose

Installs Distributed Services UID/GID translate tables into the kernel.

Syntax

```
dsxlate ──┤
```

OL805463

Description

The **dsxlate** command installs Distributed Services UID/GID translate tables. This command is usually called at system startup by **/etc/rc.ds** to update the kernel. It ensures that the Distributed Services kernel tables reflect the current profiles. All existing Distributed Services kernel information is discarded. To use **dsxlate** command, you must be a member of the system group or have superuser authority (see "**su**" on page 1026).

Related Information

The following commands: "**ipctable**" on page 544, "**ndtable**" on page 685, and "**ugtable**" on page 1109.

The **loadtbl** system call in *AIX Operating System Technical Reference.*

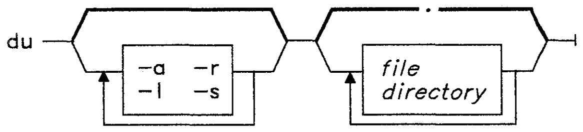
"Using Distributed Services" in *Managing the AIX Operating System.*

du

Purpose

Summarizes disk usage.

Syntax



OL805113

Description

The **du** command gives the number of blocks in all files and (recursively), directories within each specified *directory.* By specifying the **-a** flag, you can also have **du** report the number of blocks in individual files. The block count includes the indirect blocks of each file and is in units of 512 bytes, independent of the cluster size used by the system. If you provide no *file* or *directory* name, **du** uses the current directory.

Notes:

1. If you do not specify the **-a** flag, **du** does not report on any *file*s.
2. If there are too many distinct linked files, **du** counts the excess files more than once.
3. Block counts are based only on file size; therefore, unallocated blocks are not accounted for in the block counts reported.

Flags

-a	Displays disk use for each file.
-l	Allocates blocks in files with multiple links evenly among the links. By default, a file with two or more links is counted only once.
-r	Indicates inaccessible files and directories.
-s	Displays only the grand total (for each of the specified files or directories given).

Examples

1. To summarize the disk usage of a directory tree and each of its subtrees:

   ```
   du  /u/fran
   ```

   For /u/fran and each of its subdirectories, this displays the number of disk blocks that the files in the tree beneath it contain.

2. To display the disk usage of each file:

   ```
   du  -a  /u/fran
   ```

   This displays the number of disk blocks contained in each file and subdirectory of /u/fran. The number beside a directory is the disk usage of that directory tree. The number beside a regular file is the disk usage of that file alone.

3. To display only the total disk usage of a directory tree:

   ```
   du  -rs  /u/fran
   ```

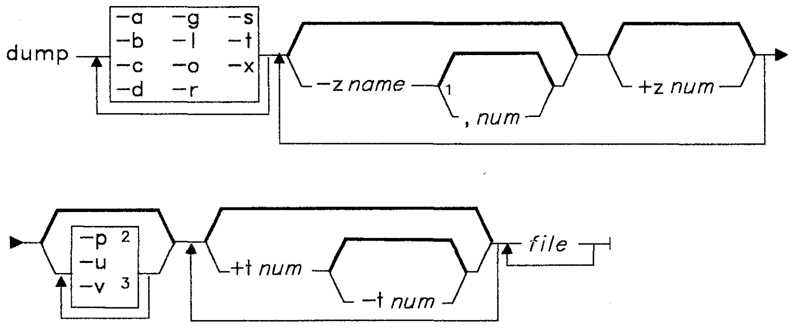
   This displays only the sum total disk usage of /u/fran and the files it contains (-s). The -r flag tells **du** to display an error message if it cannot read a file or directory.

dump

Purpose

Dumps selected parts of an object file.

Syntax



1 Do not put a space between these items.

2 Use **-p** only with **-a**, or **-o** .

3 Do not use **-v** with **-s** or **-o** .

OL805404

Description

The **dump** command dumps selected parts of the specified *file*. **dump** accepts object files, archive object files, and executable files (with the **-x** flag). It writes information in character, hexadecimal, octal, or decimal representation, as appropriate to format the information in a meaningful way.

Flags

You must use at least one of the following flags:

-a	Dumps the archive header of each member of each specified archive.
-b	Dumps the shared library key.

-c	Dumps the string table.
-d	Dumps the contents of the data section.
-g	Dumps the global symbols in the archive symbol table.
-l	Dumps line number information.
-o	Dumps each optional header.
-r	Dumps relocation information.
-s	Dumps the contents of the object file section.
-t	Dumps symbol table entries.
-x	Dumps the object module extended header from executable files. The extended header contains the table of shared libraries that the program uses.

The following optional flags are also available:

-p	Does not print the headers.
-t*num*	Dumps only the index symbol table entry specified with *num*. Use **-t** with the **+t** flag to specify a range of symbol table entries.
+t*num*	Dumps the symbol table entry in the range that ends with *num*. The range starts at either the first symbol table entry or at the entry specified by **-t**.
-u	Underlines the name of the *file*.
-v	Dumps the information in symbolic representation rather the numeric. You can use this with any of the above flags except **-s** or **-o**.
-z*name*[,*num*]	Dumps line number entries for *name* function or a range of line number entries that starts at the specified number. You can use a blank to replace the comma that separates *name* and *num* if the entire argument is quoted.
+z*num*	Dumps all line numbers up to *num*.

Related Information

The following commands: "**ar**" on page 55, "**nm**" on page 705, "**shlib**" on page 939, and "**size**" on page 949.

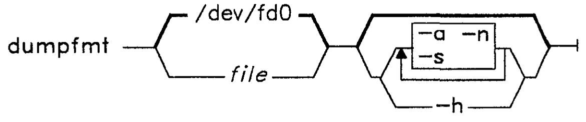
The **a.out** and **ar** files in *AIX Operating System Technical Reference*.

dumpfmt

Purpose

Formats the VRM dump file.

Syntax



OL805109

Description

The **dumpfmt** command formats a file containing VRM dump structures. If you do not specify a *file* name, the system reads data from **/dev/fd0**.

By default, **dumpfmt** is an interactive utility program. To see the list of commands available for selecting a specific structure to format, enter a ? (question mark). To quit, enter **q**.

Flags

-a	Batches the output and formats the entire diskette.
-h	Includes a Dump Data Header. This header contains general information about data on the dump diskette: the module name of the component, the data address of the module containing the component, and the offset address within the module of the component.
-n	Does not display a prompt when the screen fills with data during interactive output.
-s	Limits the output of each structure to a maximum size of 32 bytes.

Related Information

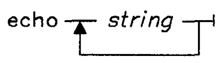
The discussion of **dumpfmt** in *AIX Operating System Programming Tools and Interfaces.*

echo

Purpose

Writes its arguments to standard output.

Syntax



OL805115

Description

The **echo** command writes its arguments to standard output. *string*s are separated by blanks and a new-line character follows the last *string*. Use **echo** to produce diagnostic messages in command files and to send data into a pipe.

The **echo** command recognizes the following escape conventions:

\b	Display a backspace character.
\c	Suppress the new-line character.
\f	Display a form-feed character.
\n	Display a new-line character.
\r	Display a carriage return character.
\t	Display a tab character.
\\	Display a backslash character.
\num	Display an 8-bit character whose ASCII value is the 1-, 2- or 3-digit octal number *num*. The first digit of *num* must be a zero.

Examples

1. To write a message to standard output:

```
echo Please insert diskette . . .
```

2. To display a message containing special characters:

   ```
   echo "\n\n\nI'm at lunch.\nI'll be back at 1:00."
   ```

   This skips three lines and displays the message:

   ```
   I'm at lunch.
   I'll be back at 1:00.
   ```

   Note: You must put the message in quotation marks if it contains escape sequences like `\n`. Otherwise, the shell treats the \ specially. See page 918 for details about quoting.

3. To use **echo** with pattern-matching characters:

   ```
   echo The back-up files are: *.bak
   ```

   This displays the message `The back-up files are:` followed by the file names in the current directory ending with `.bak`.

4. To add a single line of text to a file:

   ```
   echo Remember to set the shell search path to $PATH. >>notes
   ```

   This adds the message to the end of the file `notes` after the shell substitutes the value of the shell variable **PATH**.

5. To write a message to the standard error output:

   ```
   echo Error: file already exists. >&2
   ```

   Use this in shell procedures to write error messages. If the `>&2` is omitted, then the message is written to the standard output. For details about this type of file redirection, see "Input and Output Redirection Using File Descriptors" on page 928.

Related Information

The following commands: "**csh**" on page 225 and "**sh**" on page 913.

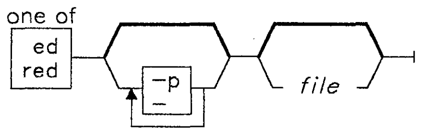
Note: The **csh** command contains a built-in subcommand named **echo**. The command and subcommand do not necessarily work the same way. For information on the subcommand, see the **csh** command.

ed

Purpose

Edits text by line.

Syntax



OL805182

Description

The **ed** command is a line editing program that works on only one file at a time by copying it into a temporary file buffer and making changes to that copy. **ed** does not alter the file itself until you use the write (**w**) subcommand. You can specify on the command line the *file* you want to edit, or you can use the **e**dit subcommands.

When **ed** reads a new file into the buffer, the contents of that file replaces the buffer's previous contents, if any.

There is also a restricted version of **ed**, the **red** command, for use with the restricted shell (see "**sh**" on page 913). With **red**, you can edit only files that reside in the current directory, or in the **/tmp** directory, and you cannot use the *!AIX-cmd* subcommand (see page 383).

An **ed** subcommand consists of zero, one, or two ***addresses***, followed by a single-character subcommand, possibly followed by parameters to that subcommand. These addresses specify one or more lines in the buffer. Because every subcommand has default addresses, you frequently do not need to specify addresses.

The **ed** program operates in one of two modes, ***command mode*** and ***text mode***. In command mode, **ed** recognizes and executes subcommands. In text mode, **ed** adds text to the file buffer but does not recognize subcommands. To leave text mode, enter a **.** (period) alone at the beginning of a line.

Pattern Matching

The **ed** command supports a limited form of ***special pattern-matching characters*** that you can use as ***regular expressions*** (***REs***) to construct ***pattern strings***. You can use these patterns in addresses to specify lines and in some subcommands to specify portions of a line.

Regular Expressions (REs)

The following REs match a *single* character:

char An ordinary *char*acter (one other than one of the special pattern-matching characters), matches itself.

. A **.** (period) matches any single character except for the new-line character.

[*string*] A *string* enclosed in [ ] (square brackets) matches *any one* character in the string. Certain pattern-matching characters have special meanings within square brackets:

^ If the first character of *string* is a ^ (circumflex), the RE ([^*string*]) matches any character ***except*** the characters in *string* and the new-line character. A ^ has this special meaning ***only*** if it occurs first in the string.

- You can use a **-** (minus) to indicate a range of consecutive ASCII characters according to the current collating sequence. For example, `[a-f]` might be equivalent to `[abcdef]` or `[aAbBcCdDeEfF]` or `[aâàbcdeéêf]`. The collating sequence is defined by the environment variable **NLCTAB** or **NLFILE**. See *Managing the AIX Operating System* for more information. A collating sequence may define equivalence classes for characters. For example, if three characters—**e**, **é**, and **è**—are equivalent, the following expressions identify the same sequence of characters:

```
[a-e]
[a-è]
```

The minus character loses its special meaning if it occurs first ([-*string*]), if it immediately follows an initial circumflex ([^-*string*]), or if it appears last ([*string*-]) in the string.

] When the right square bracket (]) is the first character in the string ([]*string*]) or when it immediately follows an initial circumflex ([^]*string*]), it is treated as a part of the string rather than as the string terminator.

Japanese Language Support Information

Japanese Language Support introduces a ***character class expression***, in addition to the standard range expression, see 375.

For information about how a string enclosed in square brackets matches characters, see 375.

\sym	A \ (backslash) followed by a special pattern-matching character matches the special character itself (as a literal character). These special pattern-matching characters are:

. * [\	Always special *except* when they appear within square brackets ([]).
^	Special at the *beginning* of an entire pattern or when it immediately follows the left bracket of a pair of brackets ([^ . . .]).
$	Special at the *end* of an entire pattern.

In addition, the character used to delimit an entire pattern is special for that pattern. (For example, see how / (slash) is used in the **g** subcommand on page 379.)

Forming Patterns

The following rules describe how to form patterns from REs:

1. An RE that consists of a single, ordinary character matches that same character in a string.
2. An RE followed by an * (asterisk) matches zero or more occurrences of the character that the RE matches. For example, the following pattern:

   ```
   ab*cd
   ```

   matches each of the following strings:

   ```
   acd
   abcd
   abbcd
   abbbcd
   ```

   but not the following string:

   ```
   abd
   ```

If there is any choice, the longest matching leftmost string is chosen. For example, given the following string:

```
122333444
```

the pattern `.*` matches `122333444`, the pattern `.*3` matches `122333`, and the pattern `.*2` matches `122`.

3. An RE followed by:

\\{*m*\\}	Matches *exactly m* occurrences of the character matched by the RE.
\\{*m*,\\}	Matches *at least m* occurrences of the character matched by the RE.
\\{*m*,*n*\\}	Matches *any number* of occurrences of the character matched by the RE *from m to n inclusive.*

   m and *n* must be integers from 0 to 255, inclusive. Whenever a choice exists, this pattern matches as many occurrences as possible.

4. You can combine REs into patterns that match strings containing that same sequence of characters. For example, `AB\*CD` matches the string `AB*CD` and `[A-Za-z]*[0-9]*` matches any string that contains any combination of alphabetic characters (including none), followed by any combination of numerals (including none).
5. The character sequence `\(`*pattern*`\)` marks a ***subpattern*** that matches the same string it would match if it were not enclosed.
6. The characters \num match the same string of characters that a subpattern matched earlier in the pattern (see the preceding discussion of item 5). *num* is a digit. The pattern \num matches the string matched by the *num*th subpattern, counting from left to right. For example, the following pattern:

   ```
   \(A\)\(B\)C\2\1
   ```

   matches the string `ABCBA`. You can nest subpatterns.

Restricting What Patterns Match

A pattern can be restricted to match only the first segment of a line, the final segment, or both:

1. A ^ (circumflex) at the beginning of a pattern causes the pattern to match only a string that begins in the first character position on a line.
2. A $ (dollar sign) at the end of a pattern causes that pattern to match only a string that ends with the last character (not including the new-line character) on a line.
3. The construction ^*pattern*$ restricts the pattern to matching only an entire line.

In addition, the null pattern (that is, //) duplicates the previous pattern.

Japanese Language Support Information

Several characters can have the same collating value; for instance, the ASCII letter a and the SJIS roman letter a could be collated together. In such a case, the expression [a-a] would match both the ASCII and the SJIS roman letters.

You can mix ASCII and SJIS characters in a range expression, as long as the character preceding the minus sign collates equal to or lower than the character following the minus sign. If the character preceding the minus collates higher than the character following the sign, the system interprets the range as consisting only of the two end points.

A common use of the range expression is to match a character class. For example, [0-9] is used to mean all digits, and [a-z A-Z] is used to mean all letters. This form may produce unexpected results when ranges are interpreted according to the current collating sequence.

Instead of the preceding form, use a ***character class expression*** within brackets to match characters. The system interprets this type of expression according to the current character class definition. However, you cannot use character class expressions in range expressions.

Following is the syntax of a character class expression:

[:*charclass*:]

that is, a left bracket, followed by a colon, followed by the name of the character class, followed by another colon and a right bracket.

Japanese Language Support supports the following character classes:

[:upper:]	ASCII uppercase letters
[:lower:]	ASCII lowercase letters
[:alpha:]	ASCII uppercase and lowercase letters
[:digit:]	ASCII digits
[:alnum:]	ASCII alphanumeric characters.
[:xdigit:]	ASCII hexadecimal digits
[:punct:]	ASCII punctuation character (neither a control character nor alphanumeric)
[:space:]	ASCII space, tab, carriage return, new-line, vertical tab, or form-feed character
[:print:]	ASCII printing character

[:jalpha:]	SJIS roman characters
[:jdigit:]	SJIS Arabic digits
[:jxdigit:]	SJIS hexadecimal digits
[:jparen:]	SJIS parentheses characters
[:jpunct:]	SJIS punctuation characters
[:jspace:]	SJIS space characters
[:jprint:]	SJIS printing characters
[:jkanji:]	kanji characters
[:jhira:]	Full-width hiragana characters
[:jkana:]	Half-width and full-width katakana characters

The brackets are part of the character class definition. To match any uppercase ASCII letter or ASCII digit, use the following regular expression:

```
[[:upper:] [:digit:]]
```

Do not use the expression [A-Z0-9].

End of Japanese Language Support Information

Addressing

There are three types of **ed** addresses: line number addresses, addresses relative to the current line, and pattern addresses. The ***current line*** (usually the last line affected by a command) is the point of reference in the buffer. This is the default address for several **ed** commands. (See "Subcommands" on page 378 to find out how each subcommand affects the current line.)

Following are guidelines for constructing addresses:

1. . (dot) addresses the current line.
2. $ (dollar sign) addresses the last line of the buffer.
3. *n* addresses the *n*th line of the buffer.
4. *'x* addresses the line marked with a lowercase ASCII letter, *x*, by the **k** subcommand (see page 380).
5. */pattern/* (a pattern enclosed in slashes) addresses the next line contains a matching string. The search begins with the line after the current line and stops when it finds a match for the pattern. If necessary, the search moves to the end of the buffer, wraps

around to the beginning of the buffer, and continues until it either finds a match or returns to the current line.

6. ?*pattern*? (a pattern enclosed in question marks) addresses the previous line that contains a match for the pattern. The ?*pattern*? construct, like /*pattern*/, can search the entire buffer, but it does so in the opposite direction.
7. An address followed by +*n* or -*n* (a plus sign or a minus sign followed by a decimal number) specifies an address plus or minus the indicated number of lines. (The + sign is optional.)
8. An address that begins with + or - specifies a line relative to the current line. For example, `-5` is the equivalent of `.-5` (five lines above the current line).
9. An address that ends with - or + specifies the line immediately before (-) or immediately after (+) the addressed line. Used alone, the - character addresses the line immediately before the current line. The + character addresses the line immediately after the current line; however, the + character is optional. The + and - characters have a cumulative effect; for example, the address -- addresses the line two lines above the current line.
10. For convenience, a `,` (comma) stands for the address pair `1,$` (first line through last line) and a ; (semicolon) stands for the pair `.,$` (current line through last line).

Commands that do not accept addresses regard the presence of an address as an error. Commands that do accept addresses can use either given or default addresses. When given more addresses than it accepts, a command uses the last (rightmost) one(s).

In most cases, commas (`,`) separate addresses (for example `2,8`). Semicolons (`;`) also can separate addresses. A semicolon between addresses causes **ed** to set the current line to the first address and then calculate the second address (for example, to set the starting line for a search based on rules 5 and 6 above). In a pair of addresses, the first must be numerically smaller than the second.

For many purposes, you may prefer to use a different editor that has different features. Refer to the following discussions for more information:

- "**edit**" on page 387, a simple line editor for novice or casual users
- "**sed**" on page 887, a stream editor often used for writing programs
- "**ex**" on page 407, an extended (line) editor with interactive subcommand features
- "**vi, vedit, view**" on page 1187, a visual (screen) editor that also accesses **ex** line editing features while letting you view the text.

The following is a list of **ed** size limitations:

- 64 characters per file name.
- 512 characters per line (although there is currently a system-imposed limit of 255 characters per line entered from the keyboard).
- 256 characters per global subcommand list.

- 128K characters buffer size. (Note that the buffer not only contains the original file but also editing information. Each line occupies one word in the buffer.)

The maximum number of lines depends on the amount of memory available to you. The maximum file size depends on the amount of physical data storage (disk or tape drive) available or on the maximum number of lines permitted in user memory.

Subcommands

In most cases, only one **ed** subcommand can be entered on a line. The exceptions to this rule are the **p** and **l** subcommands, which can be added to any subcommand except **e**, **f**, **r**, or **w**. The **e**, **f**, **r**, and **w** subcommands accept file names as parameters. The **ed** program stores the last file name used with a subcommand as a default file name. The next **e**, **f**, **r**, or **w** given without a file name uses the default file name.

The **ed** program responds to an error condition with one of two messages: ? (question mark) or ?*file*. When **ed** receives an **INTERRUPT** signal (**Alt-Pause**), it displays a ? and returns to command mode. When **ed** reads a file, it discards ASCII NULL characters and all characters after the last new-line character. **ed** cannot edit a file that contains characters not in the ASCII set (for example, an **a.out** file with bit 8 set on).

In the following list of **ed** subcommands, default addresses are shown in parentheses. Do not key in the parentheses. The address . (period) refers to the current line. When a . is shown in the first position on an otherwise empty line, it is the signal to return to command mode.

(.)**a** < *text* > .	The **a**ppend subcommand adds text to the buffer after the addressed line. The **a** subcommand sets the current line to the last inserted line, or, if no lines were inserted, to the addressed line. Address 0 causes the **a** subcommand to add text at the beginning of the buffer.
(.)**c** < *text* > .	The **c**hange subcommand deletes the addressed lines, then replaces them with new input. The **c** command sets the current line to the last new line of input, or, if there were none, to the first line that was not deleted.
(.,.)**d**	The **d**elete subcommand removes the addressed lines from the buffer. The line after the last line deleted becomes the current line. If the deleted lines were originally at the end of the buffer, the new last line becomes the current line.

e *file*	The **e**dit subcommand first deletes any contents from the buffer, then loads another file into the buffer, sets the current line to the last line of the buffer, and displays the number of characters read in to the buffer. If the buffer has been changed since its contents were last saved (with the **w** subcommand), **e** displays a ? (question mark) before it clears the buffer. The **e** subcommand stores *file* as the default file name to be used, if necessary, by subsequent **e**, **r**, or **w** subcommands. (See the **f** subcommand.) When an ! (explanation mark) replaces *file*, **e** takes the rest of the line as a AIX shell (**sh**) command and reads the command output. The **e** subcommand does not store the name of the shell command as a default file name.
E *file*	The **E**dit subcommand works like **e**, with one exception: **E** does not check for changes made to the buffer since the last **w** subcommand.
f [*file*]	The **f**ile name subcommand changes the default file name (the stored name of the last file used) to *file*, if *file* is given. If *file* is not given, the **f** subcommand prints the default file name.
(1,?)**g**/*pattern*/*subcmd-list*	The **g**lobal subcommand first marks every line that matches the pattern. Then, for each marked line, this subcommand sets the current line to that line and executes *subcmd-list.* A single subcommand, or the first subcommand of a list, should appear on the same line with the **g** subcommand; subsequent subcommands should appear on separate lines. Except for the last line, each of these lines should end with a \. The *subcmd-list* can include the **a**, **i**, and **c** subcommands and their input. If the last command in *subcmd-list* would normally be the **.** (period) that ends input mode, the **.** is optional. If there is no *subcmd-list*, **ed** displays the current line. The *subcmd-list* cannot include the **g**, **G**, **v**, or **V** subcommands. **Note:** The **g** subcommand is similar to the **v** subcommand, which executes *subcmd-list* for every line that does not contain a match for the pattern.
(1,?)**G**/*pattern*/	The interactive **G**lobal subcommand first marks every line that matches the pattern, then displays the first marked line, sets the current line to that line, and waits for a subcommand. **G** accepts any but the following **ed** subcommands: **a**, **c**, **i**, **g**, **G**, **v**, and **V**. After the subcommand finishes, **G** displays the next marked line, and so on. **G** takes a new-line character as a null subcommand. A :& (colon ampersand) causes **G** to execute the

previous subcommand again, if there was one. Note that subcommands executed within the **G** subcommand can address and change any lines in the buffer. The **G** subcommand can be terminated by pressing **INTERRUPT** (**Alt-Pause**).

h	The **h**elp subcommand gives a short explanation (help message) for the most recent ? diagnostic or error message.
H	The **H**elp subcommand causes **ed** to display the help messages for all subsequent ? diagnostics. **H** also explains the previous ? if there was one. **H** alternately turns this mode on and off; it is initially off.
(.)**i** *<text>* .	The **i**nsert subcommand inserts text before the addressed line and sets the current line to the last inserted line. If there no lines are inserted, **i** sets the current line to the addressed line. This subcommand differs from the **a** subcommand only in the placement of the input text. Address **0** is not legal for this subcommand.
(.,.+1)**j**	The **j**oin subcommand joins contiguous lines by removing the intervening new-line characters. If given only one address, **j** does nothing. (For splitting lines, see the **s** subcommand.)
(.)**k***x*	The mar**k** subcommand marks the addressed line with name *x*, which must be a lowercase ASCII letter. The address '*x* (single quotation mark before the marking character) then addresses this line. The **k** subcommand does not change the current line.
(.,.)**l**	The **l**ist subcommand displays the addressed line(s). The **l** subcommand wraps long lines and, unlike the **p** subcommand, represents non-printing characters, either with mnemonic overstrikes or in hexadecimal notation. An **l** subcommand may be appended to any **ed** subcommand except: **e**, **f**, **r**, or **w**.
(.,.)**m***a*	The **m**ove subcommand repositions the addressed line(s). The first moved line follows the line addressed by *a*. Address 0 for *a* causes **m** to move the addressed line(s) to the beginning of the file. Address *a* cannot be one of the lines to be moved. The **m** subcommand sets the current line to the last moved line.
(.,.)**n**	The **n**umber subcommand displays the addressed lines, each preceded by its line number and a tab character (displayed as blank spaces); **n** leaves the current line at the last line displayed. An **n** subcommand may be appended to any **ed** subcommand except **e**, **f**, **r**, or **w**.

(.,.)**p** The **p**rint subcommand displays the addressed line(s) and sets the current line set to the last line displayed. A **p** subcommand may be appended to any **ed** subcommand except: **e**, **f**, **r**, or **w**. For example, the subcommand **dp** deletes the current line and displays the new current line.

P The **P** subcommand turns on or off the **ed** prompt string * (asterisk). Initially, **P** is off.

q The **q**uit subcommand exits the **ed** program. Before ending the program **q** checks to determine whether the buffer has been written to a file since the last time it was changed. If not, **q** displays the **?** message.

Q The **Q**uit subcommand exits the **ed** program without checking for changes to the buffer since the last **w** subcommand (compare with the **q** subcommand).

(?)**r** *file* The **r**ead subcommand reads a file into the buffer after the addressed line; **r** does not delete the previous contents of the buffer. When entered without *file*, **r** reads the default file, if any, into the buffer (see **e** and **f** subcommands). **r** does not change the default file name. Address 0 causes *r* to read a file in at the beginning of the buffer. After it reads a file successfully, **r**, displays the number of characters read into the buffer and sets the current line to the last line read.

If ! (exclamation point) replaces *file* in a **r** subcommand, **r** takes the rest of the line as a AIX shell (**sh**) command whose output is to be read. The **r** subcommand does not store the names of shell commands as default file names.

(.,.)**s**/*pattern*/*replacement*/
(.,.)**s**/*pattern*/*replacement*/**g** The **s**ubstitute subcommand searches each addressed line for a string that matches the pattern and then replaces the string with the specified *replacement* string. Without the global indicator (**g**), **s** replaces only the first matching string on each addressed line. With the **g** indicator, **s** replaces every occurrence of the matching string on each addressed line. If **s** does not find a match for the pattern, it returns the error message ?. Any character except a space or a new-line character can separate (delimit) the pattern and *replacement.* The **s** subcommand sets the current line to the last line changed.

An & (ampersand) in the *replacement* string is a special symbol that has the same value as the *pattern* string. For example, the subcommand **s/are/&n't/** has the same effect as the subcommand **s/are/aren't/** and replaces **are** with **aren't** on the current line. A \& (backslash ampersand) removes this special meaning of & in *replacement.*

A subpattern is part of a pattern enclosed by the strings \(and \); the pattern works as if the enclosing characters were not present. In *replacement,* the characters \n refer to strings that match subpatterns; *n*, a decimal number, refers to the *n*th subpattern, counting from the left. (for example, **s/\\(t\\)\\(h\\) \\(e\\)/t\1\2ose)** replaces **the** with **those** if there is a match for the pattern **the** on the current line). Whether subpatterns are nested or in a series, \n refers to the *n*th occurrence, counting from the left, of the delimiting characters, \\).

The % (percent sign) character, when used by itself as *replacement,* causes **s** to use the previous *replacement* again. The % character does not have this special meaning if it is part of a longer *replacement* or if it is preceded by a \\.

Lines may be split by substituting new-line characters into them. In *replacement,* the sequence \Enter quotes the new-line character (not displayed) and moves the cursor to the next line for the remainder of the string. New-lines cannot be substituted as part of a **g** or **v** subcommand list.

(.,.)**t***a*
The **t**ransfer subcommand inserts a copy of the addressed lines after address *a*. The **t** subcommand accepts address 0 (for inserting lines at the beginning of the buffer). The **t** subcommand sets the current line to the last line copied.

u
The **u**ndo subcommand restores the buffer to the state it was in before it was last modified by an **ed** subcommand. The commands that **u** can undo are: **a**, **c**, **d**, **g**, **G**, **i**, **j**, **m**, **r**, **s**, **t**, **v**, and **V**.

(1,?)**v***/pattern/subcmd-list*
The **v** subcommand executes the subcommands in *subcmd-list* for each line that does not contain a match for the pattern.

Note: The **v** subcommand is a complement for the global subcommand **g**, which executes *subcmd-list* for every line that does contain a match for the pattern.

(1,$)**V***/pattern//*
The **V** subcommand first marks every line that does not match the pattern, then displays the first marked line, sets the current line to that line, and waits for a subcommand.

Note: The **V** subcommand complements the **G** subcommand, which marks the lines that do match the pattern.

(1,?)**w** *file*

The write subcommand copies the addressed lines from the buffer to the file named in *file*. If the file does not exist, the **w** subcommand creates it with permission code 666 (read and write permission for everyone), unless the **umask** setting specifies another file creation mode. (For information about file permissions, see "**umask**" on page 1110 and "**chmod**" on page 160.) The **w** subcommand does not change the default file name (unless *file* is the first file name used since you started **ed**). If you do not provide a file name, **ed** uses the default file name, if any (see the **e** and **f** subcommands). The **w** subcommand does not change the current line.

If **ed** successfully writes the file, it displays the number of characters written. When ! replaces *file*, **ed** takes the rest of the line as a AIX shell (**sh**) command whose output is to be read; **w** does not save shell command names as default file names.

Note: 0 is not a legal address for the **w** subcommand. Therefore, it is not possible to create an empty file with **ed.**

($)=

Without an address, the = (equal sign) subcommand displays the current line number. With the address $, = displays the number of the last line in the buffer. The = subcommand does not change the current line and cannot be included in a **g** or **v** subcommand list.

!*AIX-cmd*

The ! (exclamation point) subcommand allows AIX commands to be run from within **ed**. Anything following **!** on an **ed** subcommand line is interpreted as an AIX command. Within the text of that command string, **ed** replaces the unescaped % (percent sign) with the current file name, if there is one.

When used as the first character of a shell command (after the ! that runs a subshell) **ed** replaces the ! character with the previous AIX command; for example, the command **!!** repeats the previous AIX command. If the AIX command interpreter (the **sh** command), expands the command string, **ed** echoes the expanded line. The ! subcommand does not change the current line.

num
+*num*
-*num*

ed interprets a number alone on a line as an address and displays the addressed line. Addresses can be absolute (line numbers or $) or relative to the current line (+*num* or - *num*). Entering a new-line character (a blank line) is equivalent to

+**1p** and is useful for stepping forward through the buffer one line at a time.

Flags

-	Suppresses character counts that the editor displays with the **e**, **r**, and **w** subcommands, suppresses diagnostic messages for the **e** and **q** subcommands, and suppresses the ! prompt after a !*AIX-cmd.*
-p *string*	Sets the editor prompt to *string*. The default for *string* is null (no prompt).

Files

/tmp/e#	Temporary file; # is the process number.
ed.hup	Work is saved here if the terminal hangs up while **ed** is running.

Related Information

The following commands: "**grep**" on page 501, "**sed**" on page 887, "**sh**" on page 913, "**stty**" on page 1018, and "**regcmp**" on page 820.

The **regexp** system call in *AIX Operating System Technical Reference.*

The **environment** miscellaneous facility in *Text Formatting Guide.*

The discussion and examples of **ed** in *Using the AIX Operating System.*

"Overview of International Character Support" in *Managing the AIX Operating System.*

The discussion of Japanese Language Support in *Japanese Language Support User's Guide.*

edconfig

Purpose

Edits values in a **sendmail** configuration file.

Syntax

```
/usr/lib/edconfig —— file ——|
```

AJ2FL249

Description

The **edconfig** command allows you to edit a specified configuration file for the **sendmail** program. It provides a menu interface to changing some of the characteristics defined in the configuration file. To change other characteristics, you must use a text editor. You must have *superuser* authority to edit the configuration file that the system uses (**/usr/adm/sendmail/sendmail.cf**). The **edconfig** command allows you to define or change the following types of entries in the configuration file:

- The content of the *host name* class and macro
- The *domain name* macro
- The four classes that define the separate tokens of the domain name
- Configuration options (with help information) for:
  - Operational logging level
  - Default delivery mode
  - Alias file path
  - Statistics file path
  - Queue file path
  - Maximum mail retention time in queue
  - Queue uses of expensive mailers
- Configuration File Revision level

The *file* parameter provides the path name of the configuration file that you want to edit. The file must be in the format of the configuration file that is supplied with the operating system. The program searches for comment lines in that file of the form *#parameter* to locate the information in that file concerning *parameter*. For example, the comment line `#Or` precedes the line that defines the read timeout option.

To change parameters in the standard **sendmail** configuration file, enter the following command (while operating with superuser authority):

```
edconfig /usr/adm/sendmail/sendmail.cf
```

The program reads the contents of the configuration file into memory. It then displays a menu to help you select what to change. All changes are made only to the copy of the file in memory until you choose to exit and write the changes to the file. You can also exit without writing changes. The program provides information with each step in the menus to help you decide how to enter the correct information. However, you should be familiar with the **sendmail** program and its use before changing information in the configuration file.

Files

/usr/lib/edconfig	The **edconfig** program.
/usr/lib/edconfig.hf	A text file containing the help information that **edconfig** displays.
/usr/adm/sendmail/sendmail.cf	The configuration file for the **sendmail** program.

Related Information

"**sendmail**" on page 897.

The chapter about managing the mail system in *IBM RT Managing the AIX Operating System.*

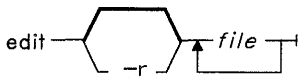
The entry for **sendmail.cf** in *AIX Operating System Technical Reference.*

edit

Purpose

Provides a simple line editor for the new user.

Syntax



OL805329

Description

The **edit** command provides a line editor designed for beginning users. It is a simplified version of the **ex** command (see "**ex**" on page 407). To edit the contents of a file, enter:

`edit` *file*

If *file* is the name of an existing file, **edit** copies it to a buffer and displays the number of lines and characters in it. Then it displays a : (colon) prompt to show that it is ready to read subcommands from standard input. If *file* does not already exist, **edit** tells you this, but still stores the name as the current file name. You can give more than one *file* name, in which case **edit** copies the first file into its buffer and stores the remaining file names in an ***argument list*** for later use.

The **edit** command operates in one of two modes: ***command mode*** and ***text entry mode***. In command mode, **edit** displays the colon prompt to show you that it is ready to accept **edit** subcommands. In text entry mode, **edit** places all input into its editing buffer. The general format of an **edit** subcommand is as follows:

[*addr*]*subcommand* [*parameters*] [*count*]

If you do not specify an *addr*ess, **edit** works on the ***current line***. If you add a numeric *count* to most subcommands, **edit** works on the specified number of lines.

For most subcommands, the last line affected becomes the new current line. That means, for example, that after **edit** reads a file into its buffer, the last line in the file becomes the current line. *addr* can be a line number or a *pattern* to be matched or, in some cases, a range of line numbers or *pattern*s. To specify a range, separate two line numbers or *pattern*s with a comma or a semicolon (for example, `1,5` or `1;5`). In a range, the second address must refer to a line that follows the first addressed line in the range.

Addressing Lines within a File

The simplest way to address a line within a file is to use its line number. But this can be unreliable because line numbers change when you insert and delete lines. **edit** provides a way to search through the buffer for strings. Given the following address:

/pattern/

edit searches forward for *pattern*, while given:

?pattern?

it searches backwards for *pattern*. If a forward search reaches the end of the buffer without finding *pattern*, it continues the search at the beginning of the file until it reaches the current line. A backwards search does just the reverse.

The following characters have special meanings in these search patterns:

^	Matches the beginning of a line.
$	Matches the end of a line.

Thus, you can use */^pattern/* to search for patterns at the beginning of a line, and */pattern$/* to search for patterns at the end of the line.

The current line has a symbolic name, **.** (period) and the last line in the buffer has a symbolic name, dollar sign ($), that you can use in addresses. This is useful when working with a range of lines. For example,

```
.,$print
```

displays all lines from the current line to the last line in the buffer. Arithmetic with line references is also possible, so that `$-5` refers to the fifth line from the last and `.+20` refers to the line 20 lines past the current line. You can also use the = (equal) command to find out the line number of the current line or the last line, as follows:

```
.=
$=
```

To view the next line in the buffer, press the **Enter** key. Press **Ctrl-D** to display the next half-screen of lines.

Note: Do not confuse the meaning of $ in text patterns (end of line) with its meaning in addresses (last line).

Using a Family of Editors

The **edit** command is part of a family of editors that includes **edit**, **ex**, and **vi**. The **edit** command is a simple line editor designed for beginning users. It is a simplified version of **ex**. After you become more experienced with **edit**, you may want to try the advanced features of one of the other editors in the family. Because **edit** is part of a family of editors, you can apply your knowledge of **edit** to the other editors in the family.

The **ex** editor is a powerful interactive line editor. The **edit** subcommands work the same way in **ex**, but the editing environment is somewhat different. For example in **edit**, only the characters ^, $, and \ have special meanings as pattern-matching characters; however, several additional characters also have special meanings in **ex**. For more information on **ex**, see "**ex**" on page 407.

The **vi** editor is a display-based editor designed for experienced users who edit intensively at their display. It contains many of the advanced features of **ex**, but focuses on the display editing portion of **ex**. The **edit** editor prevents you from accidentally entering **vi**'s two alternative modes of editing, the open mode and the visual mode. For more information on **vi**, see "**vi, vedit, view**" on page 1187.

Flag

-r	Recovers *file* after an editor or system crash.

Subcommands

You can enter most **edit** subcommands as either a complete word or an abbreviation. In the following list, a subcommand abbreviation appears in parentheses. Unless noted otherwise, all subcommands work by default on the current line. **edit** recognizes and interprets the following subcommands when it displays the colon prompt:

[*addr*]**a**ppend
text
. Reads the input *text* into the file being edited, placing the text after the line at the specified *addr*ess. If you specify address 0, **edit** places the text at the beginning of the buffer. To return to command mode, enter a line with only a . (period) in the first position.

[*addr1*[,*addr2*]]**c**hange
text
. Replaces the specified line or lines with the input *text*. If any lines are input, the last input line becomes the new current line.

[*addr1*[,*addr2*]]**d**elete [*buffer*]
Removes the specified line or lines from the editing buffer. The line following the last deleted line becomes the current line. If you specify a *buffer* by giving a letter from a to z, **edit** saves the specified lines in that buffer or, if the letter is uppercase, appends the lines to that buffer.

edit *file* Begins an editing session on a new file. The editor first checks to see if the buffer has been modified (***edited***) since the last **write** subcommand. If it has, **edit** issues a warning and cancels the **edit** subcommand. Otherwise, it deletes the complete contents of the editor buffer, makes the named file the current file, and displays the new file name. After insuring that this file can be edited, it reads the file into its buffer. If **edit** reads the file without error, it displays the number of lines and characters that it read. The last line read becomes the new current line.

file
: Displays the current file name along with the following information about it:

- Whether it has been modified since the last **write.**
- What the current line is.
- How many lines are in the buffer.
- What percentage of the way through the buffer the current line is.

file *file*
: Changes the name of the current file to *file*. **edit** considers this file ***not edited***.

[*addr1*[,*addr2*]]global/*pattern*/*cmds*
: Marks each of the specified lines that matches the *pattern*. Then **edit** carries out the specified subcommands (*cmds*) on each marked line.

A single *cmd* or the first *cmd* in a subcommand list appears on same line as **global**. The remaining *cmd*s must appear on separate lines, where each line (except the last) ends with a \ (backslash). The default subcommand is **print**.

The list can include the **append**, **insert**, and **change** subcommands and their associated input. In this case, if the ending period comes on the last line of the command list, you can omit it. The **undo** subcommand and the **global** subcommand itself, however, may not appear in the command list.

[*addr*]insert (i)
text
: Places the given text before the specified line. The last line input becomes the current line. Otherwise, the current line does not change.

[*addr1*[,*addr2*]]move *addr3* (m)
: Repositions the specified line or lines to follow *addr3*. The first of the moved lines becomes the current line.

next (n)
: Copies the next file in the command line argument list to the buffer for editing.

[*addr1*[,*addr2*]]number (nu)
: Displays each specified line or lines preceded by its buffer line number. The last line displayed becomes the current line.

preserve
: Saves the current editor buffer as though the system had just crashed. Use this command when a **write** subcommand has resulted in an error, and you do not know how to save your work.

[*addr1*[,*addr2*]]print (p)	Displays the specified line or lines. The last line becomes the current line.
[*addr*]put *buffer* (pu)	Retrieves the contents of the specified buffer and places it after *addr*. If you do not specify a buffer, **edit** restores the last deleted or yanked text. Thus you can use this subcommand together with **delete** to move lines or with **yank** to duplicate lines between files.
quit (q) quit! (q!)	Ends the editing session. **Note:** The **quit** command does ***not*** write the editor buffer to a file. However, if you have modified the contents of the buffer since the last **write**, **edit** displays a warning message and does not end the session. In this case, either use the **quit!** subcommand to discard the buffer or **write** the buffer and then **quit**.
recover *file*	Recovers *file* from the system save area. Use this after a system crash, or a **preserve** subcommand.
[*addr1*[,*addr2*]]substitute/*pattern*/*repl*/ (s) [*addr1*[,*addr2*]]substitute/*pattern*/*repl*/g	Replaces on each specified line the ***first*** instance of *pattern* with the replacement pattern *repl.* If you add the **g** flag, it replaces *all* instances of *pattern* on each specified line.
undo (u)	Reverses the changes made in the buffer by the last buffer editing subcommand. Note that **global** subcommands are considered a single subcommand to an **undo**. You cannot **undo** a **write** or an **edit**.
[*addr1*,[*addr2*]]write *file* (w)	Writes the contents of the specified line or lines to *file.* The default range is all lines in the buffer. **edit** displays the number of lines and characters that it writes. If you do not specify a *file*, **edit** uses the current file name. If *file* does not exist, **edit** creates it.
[*addr1*,[*addr2*]]yank [*buffer*] (ya)	Places the specified line or lines in *buffer* (a single alpha character name **a** - **z**).
[*addr*]z	Displays a screen of text, beginning with the specified line.
[*addr*]z-	Displays a screen of text, with the specified line at the bottom of the screen.
[*addr*]z.	Displays a screen of text, with the specified line in the middle of the screen.

Related Information

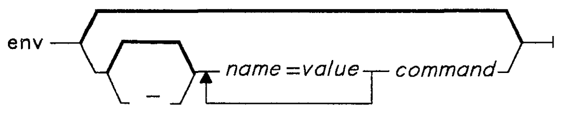
The following commands: "**ed**" on page 371, "**ex**" on page 407, and "**vi, vedit**, **view**" on page 1187.

env

Purpose

Sets the environment for execution of a command.

Syntax



OL805117

Description

The **env** command lets you get and change your current environment, and then run the named *command* with the changed environment. Changes in the form *name*=*value* are added to the current environment before the command is run. If - (minus) is used, the current environment is ignored and the command runs with only the changed environment. Changes are only in effect while the named *command* is running.

If a *command* is not specified, **env** displays your current environment one *name*=*value* pair per line.

Examples

1. To add a shell variable to the environment for the duration of one command:

   ```
   TZ=MST7MDT date
   env TZ=MST7MDT date
   ```

   Each of these commands displays the current date and time in Mountain Standard Time. The two commands shown are equivalent. When **date** is finished, the previous value of **TZ** takes effect again.

2. To replace the environment with another one:

   ```
   env - PATH=$PATH IDIR=/u/jim/include LIBDIR=/u/jim/lib make
   ```

   This runs **make** in an environment that consists *only* of these definitions for PATH, IDIR, and LIBDIR. You must redefine **PATH** so that the shell can find the **make** command.

   When **make** is finished, the previous environment takes effect again.

Related Information

The following command: "**sh**" on page 913.

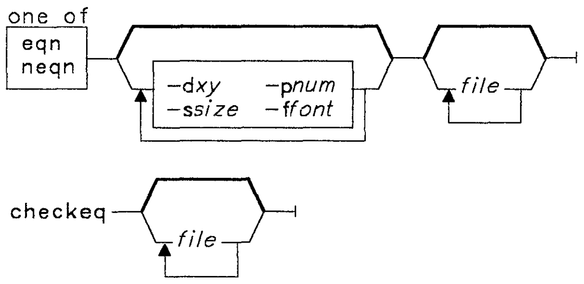
The **exec** system call, the **profile** file, and the **environ** miscellaneous facility in *AIX Operating System Technical Reference.*

eqn, neqn, checkeq

Purpose

Formats mathematical text for the **nroff, troff** and **troff** commands.

Syntax



OL805183

Description

The **eqn** command is a **troff** preprocessor for typesetting mathematical text on a phototypesetter. The **neqn** command is used with **nroff** for other *printing devices.* The output of **eqn** and **neqn** is generally piped into **troff** and **nroff** as follows:

```
eqn file | troff
neqn file | nroff
```

If you do not specify any files or if you specify - as the last file name, the commands read standard input. A line consisting of **.EQ** marks the start of equation text; the end of equation text is marked by a line consisting of **.EN**. Neither of these lines is altered by the commands, so they can be defined in macro packages to give you centering and numbering. The program **checkeq** reports missing or unbalanced delimiter pairs and **.EQ/.EN** pairs. For information on how to format **eqn** text, see *Text Formatting Guide.*

The **eqn** command recognizes the following mathematical words, and prints the associated symbol:

above	**dotdot**	**italic**	**rcol**	**to**
back	**down**	**lcol**	**right**	**under**
bar	**dyad**	**left**	**roman**	**up**
bold	**fat**	**lineup**	**rpile**	**vec**
ccol	**font**	**lpile**	**rpile**	~
col	**from**	**mark**	**size**	^
cpile	**fwd**	**matrix**	**sub**	{}
define	**gfont**	**ndefine**	**sup**	" . . . "
delim	**gsize**	**over**	**tdefine**	
dot	**hat**	**pile**	**tilde**	

Flags

-d*xy* Sets *x* and *y* as one character delimiters of the text to be processed by **eqn**, in addition to the **.EQ** and **.EN** macros. The text between these delimiters will be treated as input to **eqn**.

Note: Within a file, you can also set delimiters for **eqn** text using the command **delim** *xy*. They are turned off by the command **delim off**. All text that is not between delimiters or **.EQ** and **.EN** is passed through unprocessed.

-f*font* Acts the same as **-s** for fonts. See the discussion of **gfont** and **font** in *Text Formatting Guide* for information on changing font within the text.

-p*num* Reduces subscripts and superscripts *num* points in size (the default is 3).

-s*size* Changes point size in all **eqn** processed text to *size*. See the discussion of **gsize** and **size** in *Text Formatting Guide* for information on changing the point size within the text.

Related Information

The following commands: "**cw, checkcw**" on page 275, "**mm, checkmm**" on page 663, "**mmt, checkmm**" on page 666, "**nroff, troff**" on page 709, and "**troff**" on page 710.

The **eqnchar** and **mv** miscellaneous facilities in *AIX Operating System Technical Reference*.

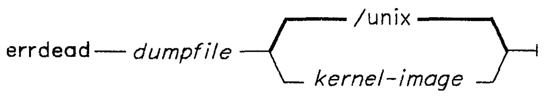
The discussion of **eqn** in *Text Formatting Guide*.

errdead

Purpose

Extracts error records from dump.

Syntax



OL805120

Description

When the system detects a hardware error, it produces an error record containing information pertinent to the error. If **errdemon**, the error-logging daemon, is not running or if the system crashes before it can place the record in the error file, the system holds the error information in a local buffer. **errdead** examines a system dump (or memory), extracts the error records, and passes them to **errpt** to generate a report. Note that no analysis is available because these error entries were never sent back via **errdemon**.

The *dumpfile* parameter specifies the file (or memory) to be examined. The *kernel-image* parameter specifies the system name list, by default **/unix**.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Files

/unix	System kernel image.
/usr/bin/errpt	Analysis program.
/usr/tmp/err*	Temporary file.

Related Information

The following command: "**errpt**, **errpd**" on page 400.

The discussion of **errdead** in *AIX Operating System Programming Tools and Interfaces.*

errdemon

Purpose

Starts the error-logging daemon.

Syntax

```
/usr/lib/errdemon —┤ 1
```

1 This command is not usually run from the command line.

OL805118

Description

The error-logging daemon **errdemon** collects error records from the operating system by reading the special file **/dev/error** and places them in one of two error log files. **errdemon** creates the names of the two log files by adding a **.0** and **.1** to the end of the file name found in **/etc/rasconf**. If an error log file does not already exist, **errdemon** creates one.

The **errdemon** command adds error records to the first error log file until it reaches the maximum allowable length specified in **/etc/rasconf**. At that point, **errdemon** closes the first error log file, changes the file name from *filename*.**0** to *filename*.**1**, and opens a new *filename*.**0**. Thus, the newest error records are always in *filename*.**0**. When it is full, **errdemon** overwrites the first file.

You can stop the error-logging daemon by sending it a **SIGKILL** signal (see "**errstop**" on page 404). Normally, the **/etc/rc** command file runs **errdemon** at system startup. Only a user operating with superuser authority can start **errdemon**, and only one daemon may be active at any time.

If **errdemon** is unable to log an error, it logs it in abbreviated form in **/dev/nvram**. Just one error can be logged in **/dev/nvram**, so each subsequent error overwrites any previous entries. When the system is started, **errdemon** searches for a previously written entry in **/dev/nvram** and, if a record is found, records it in one of the error log files and clears **/dev/nvram**.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Files

/dev/error	Source of error records.
/dev/nvram	Non-volatile read-only memory.
/etc/rasconf	Configuration file.
/etc/rc	System startup file.
/usr/adm/ras/errfile*	Repository for error records.

Related Information

The following commands: "**errpt**, **errpd**" on page 400, "**errstop**" on page 404, and "**kill**" on page 552.

The **error** and **nvram** files in *AIX Operating System Technical Reference.*

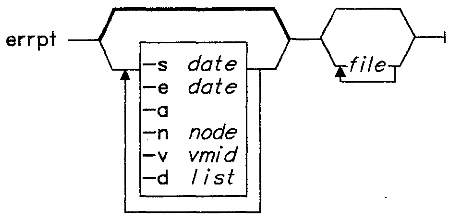
AIX Operating System Programming Tools and Interfaces.

errpt, errpd

Purpose

Processes a report of logged errors.

Syntax



OL805410

Description

The **errpt** command reads a specified error *file* or *file*s, processes the data, and writes a report of that data to standard output. These error files should be named *file*.**0** or *file*.**1**, but do not include the **.0** or **.1** extension when you specify the file name argument. **errpt** adds the extension. If you do not specify a file name, **errpt** uses the file listed in **/etc/rasconf**, adding the **.0** and **.1** extensions (these are usually **/usr/adm/ras/errfile.0** and **/usr/adm/ras/errfile.1**). The default report is a summary of all errors posted in the named file, as well as system information events, such as time changes, system starts, and so on.

The **errpt** command pipes error entries through the program **/usr/lib/errpd**, which adds probable cause information to certain entries. If no probable cause information is added, **errpt** logs records exactly as it receives them.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Flags

-a	Produces a detailed report. This contains specific error information for every event that **errpt** formats.

-d *list*	Limits the report to certain types of error records as defined by *list.* The list items can either be separated by commas or enclosed in double quotation marks and separated by commas or blanks. See "Error Identifiers" for the valid list values.
-e *date*	Includes all records posted earlier than *date,* where *date* has the form *MMddhhmmyy* (month, day, hour, minute and year).
-n *node*	Includes only entries in the error report from the specified *nodename.*
-s *date*	Includes all records posted later than *date,* where *date* has the form *MMddhhmmyy.*
-v *vmid*	Includes only entries in the error report from the system name specified with *vmid.*

Error Identifiers

In the following error identifiers, **0** acts as a wildcard character, such that, for example, **H00** gives you all hardware errors **(H11** to **HFF**), and **H10** gives you all errors from **H11** to **H1F**, and so on.

1. Class

   H00 = Hardware (01)
   S00 = Software (02)
   I00 = IPL/Shutdown (03)
   G00 = General System Condition (04)
   U00 = User Defined, Non-Hardware

2. Class/Subclass

   H10 = Hardware/Processor and Memory Management Card Machine Check
   H11 = Hardware/Main Processor
   H12 = Hardware/Main Memory
   H20 = Hardware/Fixed Disk Drive and Adapter
   H30 = Hardware/Diskette Drive and Adapter
   H40 = Hardware/Tape and Adapter
   H50 = Hardware/Display Station
   H51 = Hardware/5080 Display Adapter
   H52 = Hardware/APA16 Display Adapter
   H60 = Hardware/Display Station Adapter
   H70 = Hardware/Keyboard/Mouse
   H80 = Hardware/Communication Adapters
   H81 = Hardware/RS232 Multi-port
   H84 = Hardware/Serial or Serial/Parallel
   H85 = Hardware/IBM PC Network Adapter
   H86 = Hardware/RS422 Multi-port
   H87 = Hardware/Native Serial I/O

H8E = Hardware/SSLA
H90 = Hardware/Parallel Printer and Adapter
H91 = Hardware/Parallel or Serial/Parallel
H92 = Hardware/Parallel or PC Monochrome
HA0 = Hardware/Printers
HF0
.
.
.
HFF = User Defined Hardware

S10 = Software/Processor and Memory Management Card Program Check
S20 = Software/Abend
S21 = Software/Abend dump taken
S22 = Software/Abend No dump taken
S30 = Software/Program Error AIX
S33 = Software/Program Error Kernel
S40 = Software/Program Error Kernel Device Driver
S42 = Software/5080 Display Device Driver
S50 = Software/Program Error Kernel Device Driver
S60 = Software/Program Error VRM Base
S61 = Software/Program Error VRM Attach Device
S70 = Software/Program Base VRM Component
S72 = Software/Program Base VRM Component - Virtual Terminal
S74 = Software/5080 Display VRM Device Driver
S75 = Software/5080 Peripherals VRM Device Driver Manager
S80 = Software/Program Error Application
S80 = Software/Program Error Application - Error Log Analysis
S80 = Software/Program Error Application - Interactive Workstation
S90 = Software/Program Error Application
SA0 = Software/Program Error Application
SB0 = Software/Program Error Application
SC0 = Software/Program Error Application
SD0 = Software/Program Error Application
SE0 = Software/Program Error Application
SF0 = Software/Program Error Application

I10 = IPL/Shutdown/Manual IPL
I20 = IPL/Shutdown/Soft IPL
I30 = IPL/Shutdown/Auto IPL
I40 = IPL/Shutdown/Shutdown
I50 = IPL/Shutdown/Maintenance Shutdown

G10 = General System Condition/Degraded Config
G20 = General System Condition/Set Date/ Time
G40 = General System Condition/Error Reporting
G50 = General System Condition/LPOST

G41 = General System Condition/Cause Codes
G42 = General System Condition/Device Information
G43 = General System Condition/Counters
G51 = General System Condition/Memory Test LPOST
U10
.
.
.
UFF = User Defined, Non-Hardware

errpd

The error log analysis program, **/usr/lib/errpd**, analyzes the error log data. **/usr/lib/errpd** processes error data to determine if the error is a hardware error and if the error is a temporary or permanent error.

The analysis does the following:

- Generates a number that corresponds to a service request number.
- Analyzes the data and generates the ALERT number.
- Makes the description message ID number. The description consists of the following:
  - Error Analysis determines, from the error data passed, the nature of the operation at the time of the failure. This becomes part of the error description.
  - Error Analysis determines what failed and what the error indication is. This becomes part of the error description and is used to create the ALERT number.
  - Field Replacement Unit (FRU) Analysis determines the Service Request Code. This becomes part of the error description.

Files

/usr/adm/ras/errfile?	Error file.

Related Information

The following command: "**errdemon**" on page 398.

The **errfile** file in *AIX Operating System Technical Reference.*

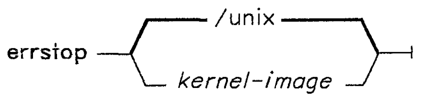
AIX Operating System Programming Tools and Interfaces.

errstop

Purpose

Terminates the error-logging daemon.

Syntax



OL805121

Description

The **errstop** command stops the error-logging daemon **errdemon** by running the **ps** command to determine the daemon process ID and then sending it a Software Terminate signal (see the **signal** system call in *AIX Operating System Technical Reference*). If you do not specify *kernel-image*, **errstop** uses **/unix**. Only a user operating with superuser authority can run **errstop**.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Files

/unix	System kernel image.

Related Information

The following commands: "**errdemon**" on page 398 and "**ps**" on page 786.

The **kill** system call in *AIX Operating System Technical Reference*.

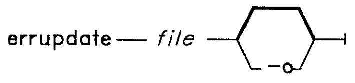
AIX Operating System Programming Tools and Interfaces.

errupdate

Purpose

Updates an error report template.

Syntax



OL805332

Description

The **errupdate** command adds, replaces, or deletes error report format templates in the file **/etc/errfmt**. **errupdate** creates an undo file in the current directory that it names *file*.**undo.err**. You can use this undo file as input to **errupdate** with the **-o** (override) flag to undo the changes **errupdate** has just made.

The **errupdate** command adds the extension **.err** to the *file* name you specify and reads update commands from the file with that name and extension. The first field of each template contains an operator:

+	To add or replace a template
-	To delete a template.

If the operation is +, then the following fields contain the template to be replaced. If the operation is a -, then the second field contains the class/subclass/mask identifier of the template to delete. **errupdate** checks for valid combinations of identifiers and writes error messages if it encounters invalid combinations. When adding or replacing, it compares the version numbers of each input template with the version number of the existing template of the same class/subclass/mask and, if the version number of the input template is later, replaces the old template with the input template. If the template does not already exist, then it is added to the file. The input template *must* contain an identifier line on the first line:

```
* /etc/errfmt
```

or **errupdate** rejects the input file. All delete operations are performed before the add/replace operations.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Flag

-o Does no version number checking.

Example

The following is an example input file:

```
* /etc/errfmt
+ H87 2.0 Native Serial: IODN D2: IOCN D2: Base_Addr D4:\
         Dev_Name A4: \n: Dev_Type X4: DDI_Length D4: Error _Type X1:\
         Last_I/O X1: Line_Status X1: Printer_Status X1:
- H92
```

Files

/etc/errfmt
file.err
file.undo.err

Related Information

The following command: "**errpt**, **errpd**" on page 400.

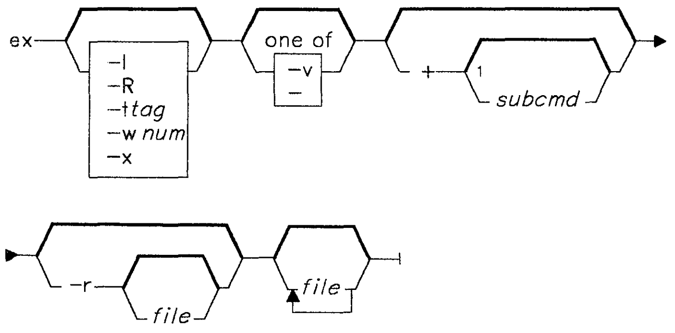
AIX Operating System Programming Tools and Interfaces.

ex

Purpose

Edits lines interactively, with screen display.

Syntax



OL805325

¹ Do not put a blank between these items.

OL805308

Description

The **ex** command is a line-oriented text editor that is a subset of the **vi** screen editor. The **ex** editor is similar to **ed**, but is more powerful, providing multiline displays and access to a screen editing mode. You may prefer to call **vi** directly to have environment variables set for screen editing. Also **edit**, a limited subset of **ex**, is available for novice or casual use. For more information on **vi**, see "**vi**, **vedit**, **view**" on page 1187. For more information on **edit**, see "**edit**" on page 387.

Notes:

1. Some **vi** subcommands have meanings that differ from **ed** subcommands.
2. To determine how to drive your work station more efficiently, **ex** uses the work station capability database **terminfo** and the type of the work station you are using from the shell environment variable **TERM**.

The **ex** editor has the following features:

- You can view text in files. The **z** subcommand lets you access windows of text, and you can scroll through text by pressing **Ctrl-D** and **Ctrl-U**. The **vi** subcommand provides further viewing options and active screen-editing by invoking the **vi** editor.
- You can revoke the last previous subcommand entered (except for **q** and **w**). The **undo** subcommand allows you to "undo" the last subcommand, even if it was an **undo** subcommand. Thus you can switch back and forth between the latest change in the edit file and the last prior file status and view the effect of a subcommand without that effect being permanent. The **ex** command displays changed lines and indicates when more than a few lines are affected by a subcommand. The **undo** subcommand causes all marks to be lost on lines changed and then restored if the marked lines were changed. It does not clear the `buffer modified` condition.
- You can retrieve your work (except changes that were in the buffer) if the system or the editor crashes by re-entering the editor with the **-r** flag and the file name. When the file name is not specified, all open files in your partition are listed.
- You can queue a sequence or group of files to edit. You can list the files in the **ex** command and then use the **next** subcommand to access each file sequentially. Or after you enter the editor, you can enter the **next** subcommand with a list of file names or a pattern (as used by the shell) to specify a set of files. In general, you can designate file names to the editor using the pattern-matching symbols that the shell will accept. You can use the wild card character % to form file names and represent the name of the current edit file.
- You can use a group of buffers (buffers named **a** through **z**) to move text between files and within a file. You can temporarily place text in these buffers and copy or reinsert it in a file, or you can carry it over to another file. The buffers are cleared when you quit the editor. The editor does not notify you if text is placed in a buffer and not used before exiting the editor.
- You can use patterns that match words. For example, you can search only for the word "ink" when your document also contains the word "inkblot" or "blink."
- You can display a window of logical lines. The **z** subcommand allows you to select the number of lines displayed and locate the current line within the display simultaneously. More than a screen of output can result when the file lines are longer than the output display lines because the set number of logical lines are displayed rather than a number of physical lines.
- You can read a file of editor subcommands. The **so** command allows you to read a file of subcommands. Nesting of source files is permitted, allowing one file to call another; however, no return mechanism is provided.

The **ex** editor has the following maximum limits:

- 1024 characters per line
- 256 characters per global command list
- 128 characters in the previous inserted and deleted text
- 100 characters in a shell escape command
- 63 characters in a string-valued option
- 30 characters in a tag name
- 250,000 lines of 1024 characters per line silently enforced
- 32 map macros with 512 characters total.

Editing States

command	Normal and initial state. Input is prompted for by : (colon). Pressing **END OF FILE (Ctrl-D)** clears an uncompleted subcommand from the command line.
entry	Entered by **a, i** and **c**. In this state you can enter text. Entry state ends normally with a line that has only a . (period) on it or ends abnormally if you press **INTERRUPT (Alt-Pause)**.
visual	Entered by **vi**, **vi.**, **vi-**, or **o**. Each of the first three commands gives you a full screen **vi** editor, but puts the current line in a different place on entry. Enter **vi** to put the current line at the top of the screen; enter **vi.** to put the current line in the middle of the screen; and enter **vi-** to put the current line at the bottom of the screen. The **o** command opens a one-line window. All three commands share the input state of the visual editor. Press the **Esc** key to exit the input state. To return to the **ex** command state at the current line, enter **Q** or ^\ while not in the input state.

Subcommands

The following table lists the **ex** subcommands. Most of these subcommands are discussed under "**edit**" on page 387 or "**vi, vedit, view**" on page 1187.

ab	abbrev	**l**	list	**rec**	recover	**x**	exit
a	append	**map**	map	**rew**	rewind	**ya**	yank
ar	args	**ma**	mark	**se**	set	**z**	window
c	change	**m**	move	**sh**	shell	**!**	escape
co	copy	**n**	next	**so**	source	**<**	lshift
d	delete	**nu**	number	**s**	substitute	**CR**	print next
e	edit	**pre**	preserve	**una**	unabbrev	**&**	resubst
f	file	**p**	print	**u**	undo	**>**	rshift
g	global	**pu**	put	**unm**	unmap	**^D**	scroll
i	insert	**q**	quit	**vi**	visual		
j	join	**re**	read	**w**	write		

Subcommand Addresses

$	The last line	*x-num*	The *num*th line before *x*
+	The next line	*x,y*	Lines *x* through *y*
−	The previous line	*'m*	The line marked with *m*
+ *num*	The *num*th line forward	''	The previous context
−*num*	The *num*th previous line	*/$pat*	The next line with *pat* at end of line
%	The first through last lines	*/^pat*	The next line with *pat* at start of line
num	line *num*	*/pat*	The next line with *pat*
.	The current line	*?pat*	The previous line with *pat*

Scanning Pattern Formation

^	The beginning of the line
$	The end of the line
.	Any character
\<	The beginning of the word
\>	The end of the word
[*string*]	Any character in *string*
[^*string*]	Any character not in *string*
[*x-y*]	Any character between *x* and *y*, inclusive
*	Any number of the preceding character.

Flags

-l Indents appropriately for Lisp code, and accepts the () {} [[and]] characters as text rather than interpreting them as **vi** subcommands. The *Lisp* modifier is active in **open** or **visual** modes.

-r [*file*] Recovers *file* after an editor or system crash. If you do not specify *file*, a list of all saved files is displayed.

-R The **readonly** option is set, preventing you from altering the file.

-t *tag* Loads the file that contains *tag* and positions the editor at *tag*.

-v Invokes the **visual** editor.

Note: When the **v** flag is selected, an enlarged set of subcommands are available, including screen editing and cursor movement features. See "**vi, vedit, view**" on page 1187.

- Suppresses all interactive-user feedback. If you use this flag, file input/output errors do not generate a helpful error message.

+*subcmd*	Begins edit at the specified editor search or subcommand. When *subcom*and is not entered, +places the current line to the bottom of the file. Normally **ex** sets current line to the start of the file, or to some specified tag or pattern.

Files

/usr/lib/exrecover	Recover subcommand.
/usr/lib/expreserve	Preserve subcommand.
/usr/lib/*/*	Describes capabilities of work stations.
$HOME/.exrc	Editor startup file.
./.exrc	Editor startup file.
/tmp/Ex*nnnnn*	Editor temporary.
/tmp/Rx*nnnnn*	Names buffer temporary.
/usr/preserve	Preservation directory.

Related Information

The following commands: "**vi, vedit, view**" on page 1187, "**edit**" on page 387, "**awk**" on page 81, "**ed**" on page 371, "**grep**" on page 501, and "**sed**" on page 887.

The **curses** subroutine and the **TERM**, **INIT**, and **terminfo** files in *AIX Operating System Technical Reference.*

expr

Purpose

Evaluates arguments as expressions.

Syntax

```
expr —— expression ——┤
```

OL805048

Description

The **expr** command reads an *expression*, evaluates it, and writes the result to standard output. Within *expression*, you must separate each term with blanks, precede characters special to the shell with a backslash (\), and quote strings containing blanks or other special characters. Note that **expr** returns 0 to indicate a zero value, rather than the null string. Integers may be preceded by a unary minus sign. Internally, integers are treated as 32-bit, twos complement numbers.

The operators and keywords are described in the following listing. Characters that need to be escaped are preceded by a backslash (\). The list is in order of increasing precedence, with equal precedence operators grouped within braces ({}).

expression1 \| *expression2*
: Returns *expression1* if it is neither null nor 0; otherwise it returns *expression2*.

expression1 \& *expression2*
: Returns *expression1* if neither *expression1* nor *expression2* is null or 0; otherwise it returns 0.

expression1 { =, \>, \>=, \<, \<=, != } *expression2*
: Returns the result of an integer comparison if both expressions are integers; otherwise returns the result of a string comparison.

expression1 {+, - } *expression2*
: Adds or subtracts integer-valued arguments.

expression1 { *, /, % } *expression2*
: Multiplies, divides, or provides the remainder from the division of integer-valued arguments.

expression1 : *expression2*
Compares *expression1* with *expression2*, which must be a pattern; pattern syntax is the same as that of the **ed** command (see page 371), except that all patterns are ***anchored***, so ^ (which anchors a pattern to the beginning of a line), is not a special character in this context.

Normally, the matching operator returns the number of characters matched. Alternatively, you can use the **\(. . . \)** symbols in *expression2* to return a portion of *expression1*. In an expression such as **[a-z]**, the minus means "through" according to the current collating sequence.

A collating sequence can define ***equivalence classes*** for use in character ranges. See "Overview of International Character Support" in *Managing the AIX Operating System* for more information on collating sequences and equivalence classes.

Japanese Language Support Information

A collating sequence does not define equivalence classes for use in character ranges. To avoid unpredictable results when using a range expression to match a class of characters, use a ***character class expression*** rather than a standard range expression. For information about character class expressions, see the discussion of this topic included in the description of the command "**ed**" on page 371.

The **expr** command returns the following exit values:

0	The expression is neither null nor 0.
1	The expression is null or 0.
2	The expression is invalid.

Note: After parameter processing by the shell, **expr** cannot distinguish between an operator and an operand except by the value. Thus, if $a is =, the command:

```
expr $a = '='
```

looks like:

```
expr = = =
```

after the shell passes the arguments to **expr**, and they will all be taken as the = operator. The following works:

```
expr X$a = X=
```

Examples

1. To modify a shell variable:

   ```
   COUNT=`expr $COUNT + 1`
   ```

   This adds 1 to the shell variable COUNT. The **expr** command is enclosed in grave accents, which causes the shell to substitute the standard output from **expr** into the COUNT= command. For more details, see "Command Substitution" on page 925.

2. To find the length of a shell variable:

   ```
   LENGTH=`expr $STR : ".*"`
   ```

   This sets LENGTH to the value given by the : (colon) operator. The pattern .* matches any string from beginning to end, so the colon operator gives the length of STR as the number of characters matched. Note that ".*" must be in quotes to prevent the shell from treating the * as a pattern-matching character. The quotes themselves are not part of the pattern.

   If STR is set to the null string, the error message `expr: syntax error` is displayed. This happens because the shell does not normally pass null strings to commands. In other words, the **expr** command sees only

   ```
   : .*
   ```

   (The shell also removes the quotation marks.) This does not work because the colon operator requires two values. This problem can be fixed by enclosing the shell variable in double quotation marks:

   ```
   LENGTH=`expr "$STR" : ".*"`
   ```

   Now if STR is null, LENGTH is set to zero. Enclosing shell variables in double quotation marks is recommended in general. However, do *not* enclose shell variables in single quotation marks. See page 918 for details about using quotation marks.

3. To use part of a string:

   ```
   FLAG=`expr "$FLAG" : "-*\(.*\)"`
   ```

   This removes leading minus signs, if any, from the shell variable FLAG. The colon operator gives the part of FLAG matched by the part of the pattern enclosed in \(\). If you omit the \(\), the colon operator gives the number of characters matched.

   If FLAG is set to - (minus), a syntax error message is displayed. This happens because the shell substitutes the value of FLAG before running the **expr** command. **expr** does not know that the minus is the value of a variable. It can only see:

   ```
   - : -*\(.*\)
   ```

and it interprets the first minus sign as the subtraction operator. We can fix this problem by using:

```
FLAG=expr "x$FLAG" : "x-*\(.*\)"
```

4. To use **expr** in an **if** statement:

   ```
   if expr "$ANSWER" : "[yY]" >/dev/null
   then
      # ANSWER begins with "y" or "Y"
   fi
   ```

   If ANSWER begins with y or Y, the **then** part of the **if** statement is performed. If the match succeeds, the result of the expression is 1 and **expr** returns an exit value of 0, which is recognized as the logical value TRUE by **if**. If the match fails, the result is 0 and the exit value 1 (FALSE).

   Redirecting the standard output of **expr** to the **/dev/null** special file discards the result of the expression. If you do not redirect it, the result is written to the standard output, which is usually your work station display.

5. Consider the following expression:

   ```
   expr "$STR" = "="
   ```

   If STR has the value = (equal sign), then after the shell processes this command **expr** sees the expression:

   ```
   = = =
   ```

   The **expr** command interprets this as three = operators in a row and displays a syntax error message. This happens whenever the value of a shell variable is the same as one of the **expr** operators. You can avoid this problem by doing the following:

   ```
   expr "x$STR" = "x="
   ```

Related Information

The following commands: "**ed**" on page 371 and "**sh**" on page 913.

"Overview of International Character Support" in *Managing the AIX Operating System.*

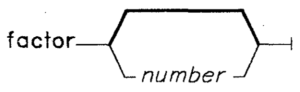
The discussion of Japanese Language Support in *Japanese Language Support User's Guide.*

factor

Purpose

Factors a number.

Syntax



OL805051

Description

When called without an argument, the **factor** command waits for you to enter a positive number less than 2^{56}. It then writes the prime factors of that number to standard output. It displays each factor the proper number of times. To exit, enter a 0 or any nonnumeric character.

When called with an argument, **factor** determines the prime factors of *number*, writes the results to standard output, and exits.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Example

To calculate the prime factors of 123:

```
factor  123
```

This displays:

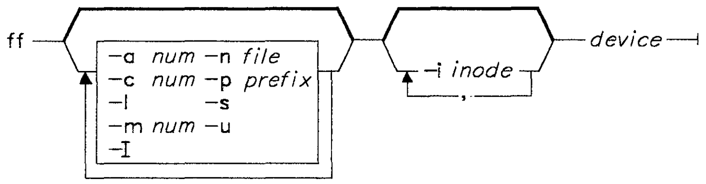
```
123
     3
     41
```

ff

Purpose

Lists the file names and statistics for a file system.

Syntax



OL805122

Description

Warning: This program is not intended for use with diskette-based file systems because of the difference in superblock structure and the general format of the file system.

The **ff** command reads the i-list and directories specified by *device* and writes information about them to standard output. It assumes that *device* is a file system, and saves i-node data for files specified by flags. The output from the **ff** command consists of the path name for each saved i-node, in addition to other file information that you request with the flags. The output is listed in order by i-node number, with tabs between all fields. The default line produced by **ff** includes the path name and i-number fields. With all flags enabled, the output fields include path name, i-number, size, and UID.

The *num* parameter in the flags descriptions is a decimal number, where +*num* means more than *num*, -*num* means less than *num*, and *num* means exactly *num*. A day is defined as a 24-hour period.

The **ff** command lists only a single path name out of many possible ones for an i-node with more than one link, unless you specify the **-l** flag. With **-l**, **ff** applies no selection criteria to the names listed. All possible names for every linked file on the file system are included in the output. On very large file systems, memory may run out before **ff** does.

Flags

-a *num*	Selects if the i-node has been accessed in *num* days.
-c *num*	Selects if the i-node has been changed in *num* days.
-i *i-node*	Generates names for only those i-nodes specified in the *inode* list.
-I	Does not display the i-node number after each path name.
-l	Generates a list of all path names for files with more than one link.
-m *num*	Selects if the file associated with the i-node has been modified in *num* days.
-n *file*	Selects if the file associated with the i-node has been modified more recently than the specified *file*.
-p *prefix*	Adds the specified *prefix* to each path name. The default prefix is **.** (dot).
-s	Writes the file size, in bytes, after each path name.
-u	Writes the owner's login name after each path name.

Examples

1. To list the path names of all files in a given file system:

   ```
   ff  -I  /dev/hd0
   ```

   This displays the path names of the files on the `/dev/hd0` disk. If you do not specify the `-I` flag, then **ff** also displays the i-number of each file.

2. To list files that have been modified recently:

   ```
   ff  -m  -2  -u  /dev/hd0
   ```

   This displays the path name, i-number, and owner's user name (`-u`) of each file on `/dev/hd0` that has been modified within the last two days (`-m -2`).

3. To list files that have *not* been used recently:

   ```
   ff  -a  +30  /dev/hd0
   ```

   This displays the path name and i-number of each file that was last accessed more than 30 days ago (`-a +30`).

4. To find out the path names of certain i-nodes:

   ```
   ff  -l  -i  451,76  /dev/hd0
   ```

   This displays all the path names (-l) associated with i-nodes 451 and 76.

Related Information

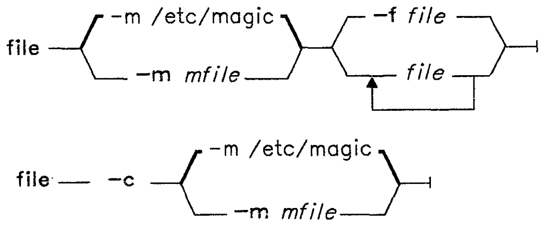
The following commands: "**find**" on page 422 and "**ncheck**" on page 683.

file

Purpose

Determines file type.

Syntax



OL805124

Description

The **file** command reads its input *file*s, performs a series of tests on each one, and attempts to classify them by their types. The command then writes the file types to standard output.

If a file appears to be ASCII, **file** examines the first 512 bytes and tries to determine its language. If a file does not appear to be ASCII, **file** further attempts to distinguish a binary data file from a text file that contains extended characters.

If *file* is an **a.out** file, and the version number is greater than zero (see "**ld**" on page 557), **file** displays the version stamp.

The **file** command uses the file **/etc/magic** to identify files that have a ***magic number***, that is, any file containing a numeric or string constant that indicates its type. Comments at the beginning of **/etc/magic** explain its format.

Japanese Language Support Information

The **file** command uses the **magic.cat** message catalog. If **magic.cat** cannot be opened, the **file** command uses the **/etc/magic** file mentioned earlier.

Flags

-c	Checks the *mfile* (**/etc/magic** by default) for format errors. This validation is not normally done. File typing is not done under this flag.
-f *file*	Reads *file* for a list of files to examine.
-m *mfile*	Specifies *mfile* as the magic file (**/etc/magic** by default).

Examples

1. To display the type of information a file contains:

   ```
   file  myfile
   ```

   This displays the file type of `myfile` (directory, data, ASCII text, C-program source, archive, and so forth).

2. To display the type of each file named in a list of file names:

   ```
   file  -f  filenames
   ```

   This displays the type of each file with a name that appears in `filenames`. Each file name must appear alone on a line.

   To create `filenames`:

   ```
   ls  >filenames
   ```

   then edit `filenames` as desired.

Files

/etc/magic	File type database.

Related Information

"Overview of International Character Support" in *Managing the AIX Operating System.*

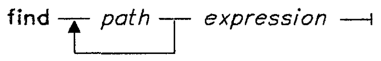
The discussion of Japanese Language Support in *Japanese Language Support User's Guide.*

find

Purpose

Finds files matching expression.

Syntax



OL805125

Description

The **find** command recursively searches the directory tree for each specified *path*, seeking files that match a Boolean *expression* written using the terms given below. The output from **find** depends on the terms used in *expression*.

Expression Terms

In the following descriptions, the parameter *num* is a decimal integer that can be specified as +*num* (more than *num*), -*num* (less than *num*), or *num* (exactly *num*).

-inum *n*	True if file has inode *n*.
-name *file*	True if *file* matches the file name. You can use pattern-matching characters, provided they are quoted. In an expression such as **[a-z]**, the minus means "through" according to the current collating sequence. A collating sequence may define ***equivalence classes*** for use in character ranges. See "Overview of International Character Support" in *Managing the AIX Operating System* for more information on collating sequences and equivalence classes.

Japanese Language Support Information

A collating sequence in Japanese Language Support does not define equivalence classes for use in character ranges. To avoid unpredictable results when using a range expression to match a class of characters, use a ***character class expression*** rather than a standard range expression. For information about character class expressions, see the discussion in "File Name Substitution" on page 4.

-node *nname* True if the file resides in the node *nname* where the nodes are connected through Distributed Services. If *nname* is a valid nickname, it is used as is. If *nname* is not a valid nickname but has a valid NID syntax, it is used as a NID.

-perm *onum* True if the file permission code of the file exactly matches the octal number *onum* (see "**chmod**" on page 160 for an explanation of file permissions). The *onum* parameter may be up to three octal digits. If you want to test the higher-order permission bits (the set-user-ID bit or set-group-ID bit, for example), prefix the *onum* parameter with a minus (-) sign. This makes more flag bits significant (see the **stat** system call for an explanation of the additional bits), and also changes the comparison to:

(flags&onum)= =onum

-type *type* True if the file *type* is of the specified type as follows:

b	Block special file
c	Character special file
d	Directory
f	Plain file
p	FIFO (a named pipe).

-links *num* True if the file has *num* links. See "**ln**" on page 581.

-user *uname* True if the file belongs to the user *uname*. If *uname* is numeric and does not appear as a login name in the **/etc/passwd** file, it is interpreted as a user ID.

-group *gname* True if the file belongs to the group *gname*. If *gname* is numeric and does not appear in the **/etc/group** file, it is interpreted as a group ID.

-size *num* True if the file is *num* blocks long (512 bytes per block). For this comparison the file size is rounded up to the nearest block.

-atime *num* True if the file has been accessed in *num* days.

-mtime *num* True if the file has been modified in *num* days.

-ctime *num*	True if the file i-node has been changed in *num* days.
-exec *cmd*	True if the *cmd* runs and returns a zero value as exit status. The end of *cmd* must be punctuated by a quoted or escaped semicolon. A command parameter {} is replaced by the current path name.
-ok *cmd*	The **find** command asks you whether it should start *cmd*. If your response begins with `y`, *cmd* is started. The end of *cmd* must be punctuated by a quoted or escaped semicolon.
-print	Always true; causes the current path name to be displayed. **find** does not display path names unless you specify this expression term.
-cpio *device*	Write the current file to *device* in **cpio** format. See "**cpio**" on page 205.
-newer *file*	True if the current file has been modified more recently than the file indicated by *file*.
-depth	Always true. This causes the descent of the directory hierarchy to be done so that all entries in a directory are affected before the directory itself. This can be useful when **find** is used with **cpio** to transfer files that are contained in directories without write permission.
\\(*expression* \\)	True if the expression in parentheses is true.

You may perform the following logical operations on these terms (listed in order of decreasing precedence):

- Negate a term (! is the NOT operator).
- Concatenate terms (juxtaposing two terms implies the AND operation).
- Alternate terms (**-o** is the OR operator).

Examples

1. To list all files in the file system with a given base file name:

   ```
   find / -name .profile -print
   ```

   This searches the entire file system and writes the complete path names of all files named `.profile`. The / tells **find** to search the root directory and all of its subdirectories. This may take a while, so it is best to limit the search by specifying the directories where you think the files might be.

2. To list the files with a specific permission code in the current directory tree:

   ```
   find . -perm 0600 -print
   ```

   This lists the names of the files that have *only* owner-read and owner-write permission. The . (dot) tells **find** to search the current directory and its subdirectories. See "**chmod**" on page 160 for details about permission codes.

3. To search several directories for files with certain permission codes:

   ```
   find  manual  clients  proposals  -perm  -0600  -print
   ```

   This lists the names of the files that have owner-read and owner-write permission *and possibly other permissions*. The directories `manual`, `clients`, and `proposals`, and their subdirectories, are searched. Note that `-perm 0600` in the previous example selects only files with permission codes that match `0600` *exactly*. In this example, `-perm -0600` selects files with permission codes that allow *at least* the accesses indicated by `0600`. This also matches the permission codes `0622` and `2744`.

4. To search for regular files with multiple links:

   ```
   find  .  -type  f  -links  +1  -print
   ```

   This lists the names of the ordinary files (`-type f`) that have more than one link (`-links +1`). Note that every directory has at least two links: the entry in its parent directory and its own . (dot) entry. See "**ln**" on page 581 for details about multiple file links.

5. To back up selected files in **cpio** format:

   ```
   find  .  -name  "*.c"  -cpio  /dev/rfd0
   ```

   This saves all the `.c` files onto the diskette in **cpio** format. See "**cpio**" on page 205 for details. Note that the pattern `"*.c"` must be quoted to prevent the shell from treating the * as a pattern-matching character. This is a special case in which **find** itself decodes the pattern-matching characters.

6. To perform an action on all files that meet complex requirements:

   ```
   find . \( -name a.out -o -name "*.o" \) -atime +7 -exec rm {} \;
   ```

   This deletes (`-exec rm {} \;`) all files named `a.out` or that end with `.o`, and that were last accessed over seven days ago (`-atime +7`). The `-o` flag is the logical OR operator.

Files

/etc/group	File that contains all known groups.
/etc/passwd	File that contains all known users.

Related Information

The following commands: "**cpio**" on page 205, "**sh**" on page 913, and "**test**" on page 1064.

The **stat** system call and the **cpio** and **fs** files in *AIX Operating System Technical Reference.*

"Overview of International Character Support" and "Using Distributed Services" in *Managing the AIX Operating System.*

The discussion of Japanese Language Support in *Japanese Language Support User's Guide.*

fish

Purpose

Plays the card game Go Fish.

Syntax

```
/usr/games/fish ─┤
```

OL805189

Description

The object of the **fish** game is to accumulate ***books*** of four cards with the same face value. You and the program take turns asking each other for a card in your hand. If your opponent has one or more cards of that value, he must hand them over. If not, he says `GO FISH`, and you draw a card from the pool of undealt cards. If you draw the card you asked for, you draw again. As books are made, they are laid down on the table. Play continues until there are no cards left. The player with the largest number of books wins the game. **fish** tells you the winner and exits.

The **fish** game asks if you want instructions before play begins. To see the instructions, enter `y` or `yes`.

Entering **p** as your first move gives you the professional-level game.

The **fish** game tells you the cards in your hand each time it prompts for a move. It tells you when either side makes a book, says `GO FISH` for you, and draws for you. All you must enter as play progresses is the value of the card you want to ask for. If you press only the **Enter** key, you are given information about the number of cards in your opponent's hand and in the pool.

To exit the game before play is completed, press INTERRUPT (**Alt-Pause**).

fmt

Purpose

Formats mail messages prior to sending.

Syntax

```
/usr/bin/fmt —— file ——|
```

AJ2FL123

Description

The **fmt** command invokes a simple text formatter that reads the concatenation of input *files* (or standard input if no *files* are specified). It then produces, on standard output, a version of the input with lines as close to 72 characters long as possible. The spacing at the beginning of the input lines is preserved in the output, as are blank lines and interword spacing.

The **fmt** command is generally used to format mail messages prior to sending them through the mail facility. It may also be useful, however, for other simple formatting tasks. For example, within visual mode of a text editing program such as **vi**, the command `!}fmt` reformats a paragraph so that all lines are approximately 72 characters long.

Note: The **fmt** command is a fast, simple formatting program. Standard text editing programs are more appropriate than **fmt** for complex formatting operations.

Related Information

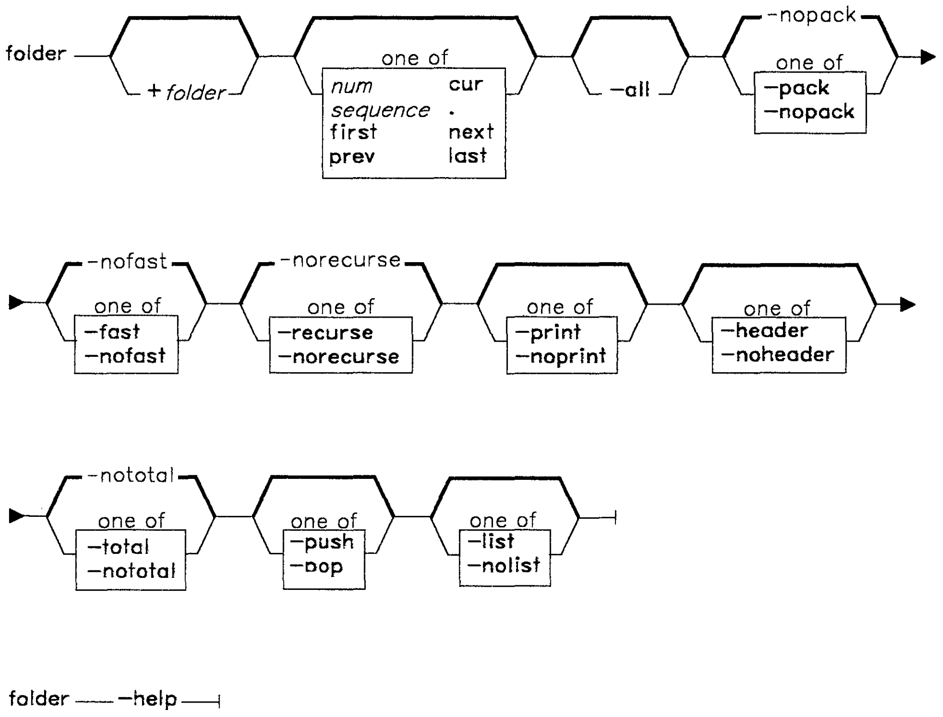
The following commands: "**mail, Mail**" on page 608, and "**nroff, troff**" on page 709.

folder

Purpose

Selects and lists folders and messages.

Syntax



AJ2FL228

Description

The **folder** command is used to set the current folder and the current message for that folder, and to list information about your folders. The **folder** command is part of the MH (Message Handling) package and can be used with other MH and AIX commands.

The **folder** command entered with only the **-all** flag gives a line of information for each folder in your mail directory. This information tells how many messages are in each folder, what the current message is for each folder, what the range of message numbers is in each folder, and if a folder is the current folder. The **folder** command specified without arguments provides information for the current folder. The **-recurse** flag displays this information for all folders and subfolders in your entire mail directory structure.

Specify a folder with the **folder** command to make it the current folder, and a message to make it the current message for that folder. You can also create a folder-stack to manipulate a group of folders by using the **-push**, **-pop**, and **-list** flags.

Flags

-all Displays a line of information about each folder in your mail directory. Specify *+folder* to display the information about that folder and the subfolders within that folder's directory. Add the **-recurse** flag to display the information about the specified folder and for all subfolders in all directories under the specified folder's directory.

-fast Displays only the names of the folders.

+folder msg Sets the specified folder as the current folder, and the specified message as the current message. You can use the following message references when specifying *msgs*:

num	*sequence*	**first**
prev	**cur**	**.**
next	**last**	

If you specify a *sequence*, that sequence must contain one message only.

The default folder is the current folder. The default message is none.

-header Displays column headings for the folder information.

-help Displays help information for the command.

-list Displays the current folder followed by the contents of the folder-stack.

-nofast Displays information about each folder. This flag is the default.

-noheader Suppresses column headings for the folder information. This flag is the default.

-nolist Suppresses the display of the folder-stack contents. This flag is the default.

-nopack Does not renumber the messages in the folder. This flag is the default.

-noprint Does not display folder information. If **-push**, **-pop**, or **-list** is specified, **-noprint** is the default.

-norecurse	Displays information about the top-level folders in your current folder only, not about subfolders. This flag is the default.
-nototal	Does not display a total of all messages and folders in your mail directory structure. **-total** is the default when **-all** is specified, otherwise **-nototal** is the default.
-pack	Renumbers the messages in the specified folder. This renumbering eliminates the gaps in the message numbering after messages have been deleted.
-pop	Removes the folder from the top of the folder-stack and makes it the current folder. +*folder* cannot be specified with the **-pop** flag.
-print	Displays information about the folders. This information includes the number of messages in each folder, the current message for each folder, and the current folder. If **-push**, **-pop**, or **-list** is specified, **-noprint** is the default, otherwise **-print** is the default.
-push	Moves the current folder to the top of the folder-stack and sets the specified folder as the current folder. If no folder is specified, **-push** swaps the current folder with the folder on the top of the folder-stack.
-recurse	Displays information about all folders and subfolders in your current folder. You can specify a folder to display the information about that folder and its subfolders only.
-total	Displays a total of all messages and folders in your mail directory structure. **-total** does not display information for subfolders unless you also specify the **-recurse** flag. The **-total** flag is the default if **-all** is specified.

Profile Entries

Current-Folder:	Sets your default current folder.
Folder-Protect:	Sets the protection level for your new folder directories.
Folder-Stack:	Specifies your folder stack.
lsproc:	Specifies the program used to list the contents of a folder.
Path:	Specifies your *user_mh_directory*.

Files

$HOME/.mh_profile	The MH user profile.

Related Information

Other MH commands: “**folders**” on page 433, “**mhpath**” on page 648, “**packf**” on page 733, “**refile**” on page 817.

The **mh-profile** file in *AIX Operating System Technical Reference.*

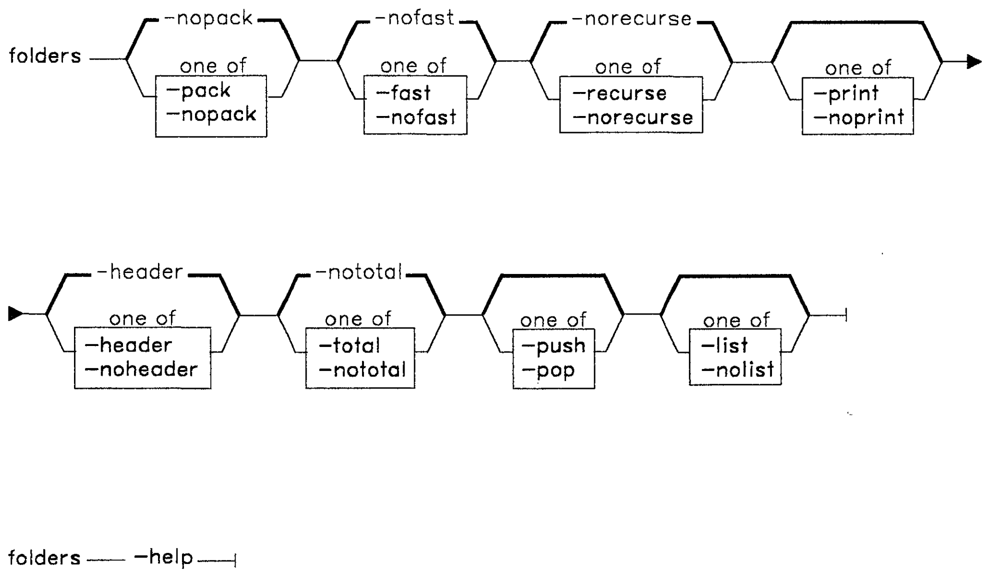
“Overview of the Message Handling Package” in *Managing the AIX Operating System.*

folders

Purpose

Lists folders and messages.

Syntax



AJ2FL229

Description

The **folders** command is used to list information about your folders. The **folders** command is part of the MH (Message Handling) package and can be used with other MH and AIX commands.

The **folders** command is equivalent to the **folder** command specified with the **-all** flag.

Flags

-fast	Displays only the names of the folders in your mail directory.
-header	Displays column headings for the folder information. This flag is the default.
-help	Displays help information for the command.
-list	Displays the current folder followed by the contents of the folder-stack.
-nofast	Displays information about each folder in your mail directory. This flag is the default.
-noheader	Suppresses column headings for the folder information.
-nolist	Suppresses the display of the folder-stack contents. This flag is the default.
-nopack	Does not renumber the messages in the folder. This flag is the default.
-noprint	Does not display folder information. If **-push**, **-pop**, or **-list** is specified, **-noprint** is the default.
-norecurse	Displays information about the folders in your mail directory only, not about subfolders. This flag is the default.
-nototal	Does not display a total of all messages and folders in your mail directory structure.
-pack	Renumbers the messages in the folders. This eliminates the gaps in the message numbering after messages have been deleted.
-pop	Removes the folder from the top of the folder-stack and makes it the current folder.
-print	Displays information about the folders. This information includes the number of messages in each folder, the current message for each folder, and the current folder. If **-push**, **-pop**, or **-list** is specified, **-noprint** is the default, otherwise **-print** is the default.
-push	Swaps the current folder with the folder on the top of the folder-stack.
-recurse	Displays information about all folders and subfolders in your entire mail directory structure.
-total	Displays a total of all messages and folders in your mail directory structure. **-total** does not display information for subfolders unless you also specify the **-recurse** flag. The **-total** flag is the default.

Profile Entries

Current-Folder:	Sets your default current folder.
Folder-Protect:	Sets the protection level for your new folder directories.
Folder-Stack:	Specifies your folder stack.
lsproc:	Specifies the program used to list the contents of a folder.
Path:	Specifies your *user_mh_directory*.

Files

$HOME/.mh_profile	The MH user profile.

Related Information

Other MH commands: "**folder**" on page 429, "**mhpath**" on page 648, "**packf**" on page 733, "**refile**" on page 817.

The **mh-profile** file in *AIX Operating System Technical Reference.*

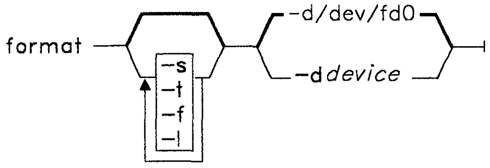
"Overview of the Message Handling Package" in *Managing the AIX Operating System.*

format

Purpose

Formats diskettes.

Syntax



OL805395

Description

The **format** command formats diskettes in the specified *device* (**/dev/fd0** by default). **format** determines the device type, either a 360K or a 1.2M diskette drive. By default, it formats a diskette in a 360K drive to have 40 cylinders, 9 sectors per track, and 2 sides and a diskette in a 1.2M drive to have 80 cylinders, 15 sectors per track, and 2 sides.

Flags

-d*device*	Specifies the device containing the diskette to be formatted.
-f	Formats the diskette without checking for bad tracks, thus formatting the diskette faster.
-l	Formats a 360K diskette in a 1.2M diskette drive. **Warning:** A 360K diskette drive may not be able to read a 360K diskette that has been formatted in a 1.2M drive.
-s	Specifies a single-sided diskette. Use only for 360K diskette drives.
-t	Specifies that the number of sectors on a 360K diskette should be 8.

Related Information

The **fd** file in *AIX Operating System Technical Reference.*

fortune

Purpose

Tells a fortune.

Syntax

```
/usr/games/fortune —|
```

OL805190

Description

The **fortune** game tells a fortune, selected at random from the file **/usr/games/lib/fortunes**, and exits.

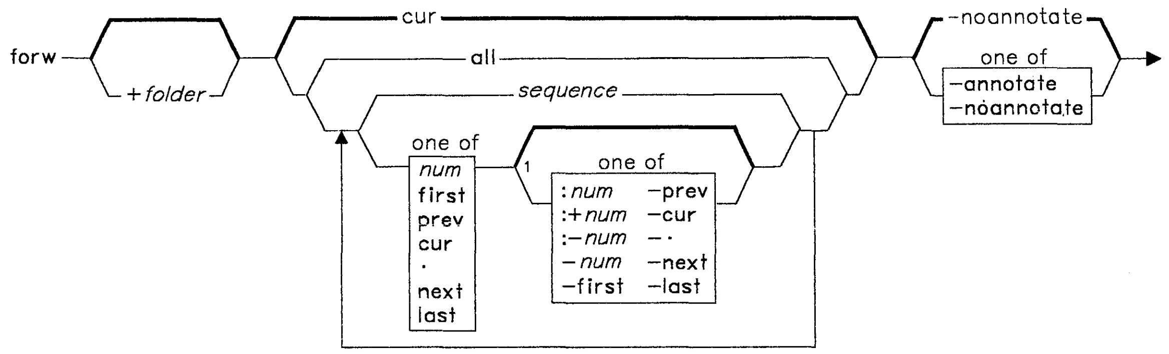
You can edit the file **/usr/games/lib/fortunes** to add your own fortunes. Each saying in the file should be a single line. **fortune** folds long sayings into multiple lines as necessary.

forw

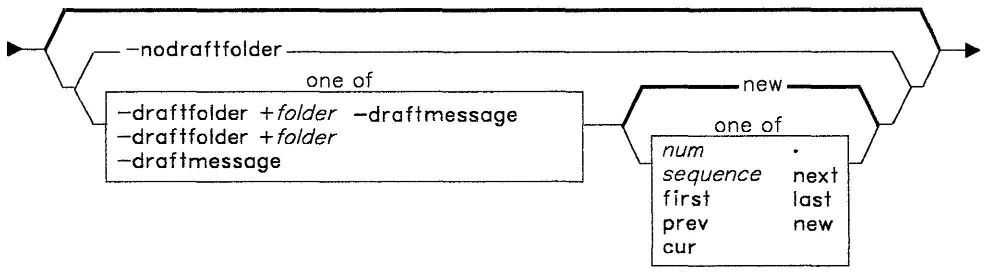
Purpose

Forwards messages.

Syntax



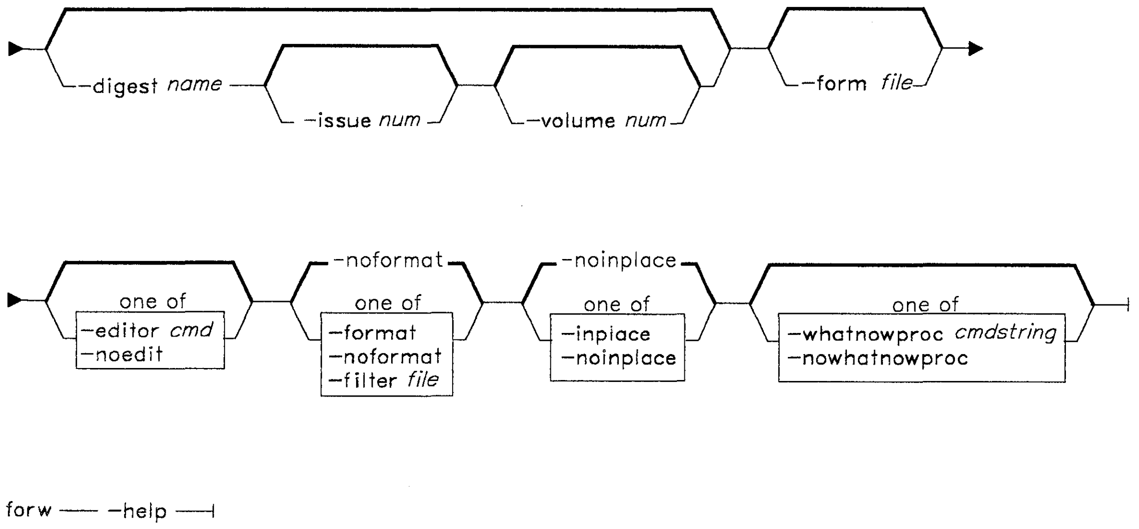
AJ2FL218



AJ2FL157

[1] Do not put a blank between these items.

OL805308



AJ2FL219

Description

The **forw** command is used to create a message containing other messages. **forw** is part of the MH (Message Handling) package and can be used with other MH and AIX commands.

By default, **forw** copies a message form to a new draft message and invokes an editor. You can then fill in the message header fields **To:** and **Subject:** and fill in or delete the other header fields (such as **cc:** and **Bcc:**). When you exit the editor, the **forw** command invokes the MH command **whatnow**. You can press **Enter** to see a list of the available **whatnow** subcommands. These subcommands enable you to continue editing the message, list the message, direct the disposition of the message, or end the processing of the **forw** command. See "**whatnow**" on page 1215 for a description of the subcommands.

You can specify the messages that you want to distribute by using the *+folder msgs* flag. If you do not specify a message, **forw** forwards the current message.

You can specify the format of the message by using the **-form** flag. If you do not specify this flag, **forw** uses your default message format located in the file *user_mh_directory*/**forwcomps**. If this file does not exist, **forw** uses the system default message format located in **/usr/lib/mh/forwcomps**.

Note: The line of dashes or a blank line must be left between the header and the body of the message for the message to be identified when it is sent.

Flags

-annotate Annotates the messages being forwarded with the lines:

```
Forwarded: date
Forwarded: addrs
```

The annotation appears in the original draft message so that you can maintain a complete list of recipients with the original message. If you do not actually redistribute the message using the immediate **forw** command, the **-annotate** may fail to provide annotation. The **-inplace** flag forces annotation to be done in place.

-digest *name* Uses the digest facility to create a new issue for the digest called *name*. **forw** expands the format strings in the *components* file (using the same format string mechanism used by the **repl** command) and composes the draft using the standard digest encapsulation algorithm. After the draft has been composed, **forw** writes out the volume and issue entries for the digest and invokes the editor.

-draftfolder *+folder* Places the draft message in the specified folder. If you do not specify this flag, **forw** selects a default draft folder according to the information supplied in the MH profiles. You can define a default draft folder in **$HOME/.mh_profile**. If **-draftfolder** *+folder* is followed by *msg*, *msg* represents the **-draftmessage** attribute.

-draftmessage *msg* Specifies the draft message. You can specify one of the following message references as *msg*:

num	*sequence*	**first**
prev	**cur**	**.**
next	**last**	**new**

The default draft message is **new**.

-editor *cmd* Specifies that *cmd* is the initial editor for preparing the message. If you do not specify this flag, **forw** selects a default editor or suppresses the initial edit, according to the information supplied in the MH profiles. You can define a default initial editor in **$HOME/.mh_profile**.

-filter *file* Reformats each message being forwarded and places the reformatted message in the draft message. **-filter** uses the **mhl** command and the specified format file. When you also specify the **-digest** flag, you may want to use the filter file **/usr/lib/mh/mhl.digest**.

+*folder msgs*	Specifies the messages that you want to forward. *msgs* can be several messages, a range of messages, or a single message. You can use the following message references when specifying *msgs*:

num	**first**	**prev**
cur	**.**	**next**
last	**all**	*sequence*

The default message is the current message in the current folder. If you specify several messages, the first message forwarded becomes the current message. If you specify a folder, that folder becomes the current folder.

-form *file*	Uses the form contained in the specified file to construct the beginning of the message. **forw** treats each line in *file* as a format string. If the **-digest** flag is also specified, **forw** uses the form specified in *file* for the format of the digest. If you do not specify a form for a digest, **forw** uses the format in the file *user_mh_directory*/**digestcomps**. If this file does not exist, **forw** uses the system default digest form specified in the file **/usr/lib/mh/digestcomps**.
-format	Reformats each message being forwarded and places the reformatted message in the draft message. **-format** uses the **mhl** command and a default format file. If *user_mh_directory*/**mhl.forward** exists, it contains the default format. Otherwise, **/usr/lib/mh/mhl.forw** contains the default format.
-help	Displays help information for the command.
-inplace	Forces annotation to be done in place in order to preserve links to the annotated message.
-issue *num*	Specifies the issue number of the digest. The default issue number is one greater than current value of the *digest_name*-**issue-list** entry in *user_mh_directory*/**context**.
-noannotate	Does not annotate the message. This flag is the default.
-nodraftfolder	Places the draft in the file *user_mh_directory*/**draft**.
-noedit	Suppresses the initial edit.
-noformat	Does not reformat the messages being forwarded. This flag is the default.
-noinplace	Does not perform annotation in place. This flag is the default.

-nowhatnowproc	Does not invoke a program that guides you through the forwarding tasks. The **-nowhatnowproc** flag also prevents any edit from occurring.
-volume *num*	Specifies the volume number of the digest. The default volume number is the current value of the *digest_name*-**volume-list** entry in *user_mh_directory*/**context**.
-whatnowproc *cmdstring*	Invokes *cmdstring* as the program to guide you through the forwarding tasks. See "**whatnow**" on page 1215 for information about the default **whatnow** program and its subcommands.

Note: If you specify `whatnow` for *cmdstring*, **forw** invokes an internal **whatnow** procedure rather than a program with the file name **whatnow**.

Profile Entries

Current-Folder:	Sets your default current folder.
Draft-Folder:	Sets your default folder for drafts.
Editor:	Sets your default initial editor.
fileproc:	Specifies the program used to refile messages.
mhlproc:	Specifies the program used to filter messages being forwarded.
Msg-Protect:	Sets the protection level for your new message files.
Path:	Specifies your *user_mh_directory*.
whatnowproc:	Specifies the program used to prompt `What now?` questions.

Files

/usr/lib/mh/forwcomps	The MH default message skeleton.
user_mh_directory/forwcomps	The user's default message skeleton. (If it exists, it overrides the MH default message skeleton.)
/usr/lib/mh/digestcomps	The MH default message skeleton when **-digest** is specified.
user_mh_directory/digestcomps	The user's default message skeleton when **-digest** is specified. (If it exists, it overrides the MH default message skeleton.)
/usr/lib/mh/mhl.forward	The default MH message filter.
user_mh_directory/mhl.forward	The user's default message filter. (If it exists, it overrides the MH default message filter.)
$HOME/.mh_profile	The MH user profile.
user_mh_directory/draft	The draft file.
user_mh_directory/context	The context file.

Related Information

Other MH commands: "**ali**" on page 48, "**anno**" on page 50, "**comp**" on page 185, "**dist**" on page 336, "**dp**" on page 352, "**inc**" on page 518, "**mhl**" on page 643, "**msh**" on page 677, "**repl**" on page 821, "**send**" on page 893, "**whatnow**" on page 1215, "**whom**" on page 1222.

The **mh-alias**, **mh-format**, **mh-mail**, and **mh-profile** files in *AIX Operating System Technical Reference.*

"Overview of the Message Handling Package" in *Managing the AIX Operating System.*

fptype

Purpose

Displays the floating point configuration of the system.

Syntax

```
fptype —|
```

OL805471

Description

The **fptype** command calls the subroutine, **_FPtype**, which determines the current floating point configuration. Then **fptype** returns one of the following values and displays the corresponding message:

Value	**Message**
0	Floating Point Type = Software Emulation
1	Floating Point Type = FPA Card
2	Floating Point Type = APC Card with MC68881
4	Floating Point Type = AFPA - No DMA Support
12	Floating Point Type = AFPA - With DMA Support

Related Information

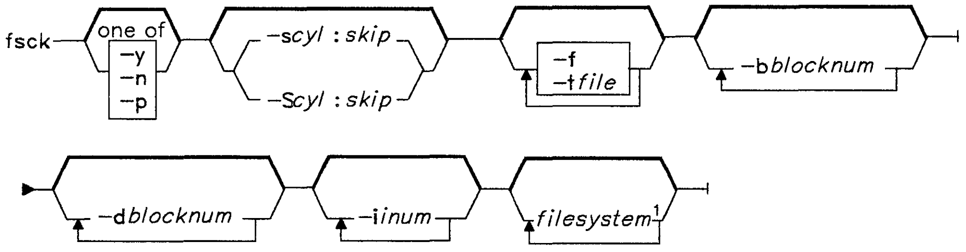
The **fpfp** subroutine in *AIX Operating System Technical Reference.*

fsck, dfsck

Purpose

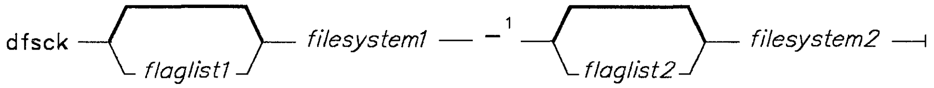
Checks file system consistency and interactively repairs the file system.

Syntax



1 The default action is to check every file system with the attribute check=true in the file /etc/filesystem.

OL8055384



1 Use a -- to separate the groups when you specify flags as part of the argument.

OL805456

Description

Warning: Always run **fsck** on file systems after a system crash. Corrective actions may result in some loss of data. The default action for each consistency correction is to wait for the operator to respond yes or no. If you do not have write permission for an affected file, **fsck** defaults to a no response in spite of your actual response.

The **fsck** command checks and interactively repairs inconsistent *filesystems*. It should be run on every file system as part of system initialization (see "**rc**" on page 806). You must have superuser authority to run **fsck**. Normally, the file system is consistent, and **fsck** merely reports on the number of files, used blocks and free blocks in the file system. If the *filesystem* is inconsistent, **fsck** displays information about the inconsistencies found and prompts you for permission to repair them. **fsck** is conservative in its repair efforts and tries to avoid actions that might result in the loss of valid data. In certain cases, however, **fsck** recommends the destruction of a damaged file.

If you do not specify *filesystem*, **fsck** looks at **/etc/filesystems** to find a list of file systems to check by default. **fsck** can perform checks (on separate arms) in parallel (running in parallel processes). This can reduce the time required to check a large number of file systems.

In **/etc/filesystems**, automatic checking may be enabled by adding a line in the stanza, as follows:

```
check=true
```

If you specify the **-p** flag, **fsck** can perform multiple checks at the same time. To tell **fsck** which file systems are on the same drives, change the **check** specification in **/etc/filesystems** as follows:

check=*number*

The *number* tells **fsck** which group contains a particular file system. File systems on a single drive are placed in the same group. Each group is checked in a separate parallel process. File systems are checked, one at a time, in the order that they appear in **/etc/filesystems**. All `check=true` file systems are in group 1. **fsck** attempts to check the root file system before any other file system regardless of order specified on the command line or in **/etc/filesystems**.

The **fsck** command checks for the following inconsistencies:

- Blocks allocated to multiple files or to a file and the free list.
- Blocks allocated to a file or on the free list outside the range allowable block numbers.
- Discrepancies between the number of directory references to a file and the link count in the file.
- Size checks:
  - Incorrect number of blocks.
  - Directory size not 16-byte aligned.
- Bad i-node format.
- Blocks not accounted for anywhere.
- Directory checks:
  - File pointing to an i-node that is not allocated.
  - I-node number out of range.
  - Dot (.) link missing or not pointing to itself.
  - Dot dot (..) link missing or not pointing to the parent directory.
  - Files that are not referenced or directories that are not reachable.

- Superblock checks:
  - More than 65535 i-nodes.
  - More blocks for i-nodes than there are in the file system.
- Bad free block list format.
- Total free block and/or free i-node count incorrect.

Orphaned files and directories (those that cannot be reached) are, if you allow it, reconnected by placing them in the **lost+found** subdirectory in the root directory. The name assigned is the i-node number. The only restriction is that the directory **lost+found** must already exist in the root directory of the file system being checked and must have empty slots in which entries can be made (accomplished by copying a number of files to the directory and then removing them before you run **fsck**). If you do not allow **fsck** to reattach an orphaned file, it requests permission to destroy the file. When **fsck** displays i-node information, the **NLTIME** environment variable controls the format of the modification time.

In addition to its messages, **fsck** records the outcome of its checks and repairs through its exit value. This exit value can be any sum of the following conditions:

0	All checked file systems are now okay.
2	**fsck** was interrupted before it could complete checks or repairs.
4	**fsck** changed the mounted file system; the user must restart the system immediately.
8	The file system contains unrepaired damage.

When the system is being started up normally, **fsck** runs with the **-p** flag from **/etc/rc** (see "**rc**" on page 806). If **fsck** detects and repairs errors on the root or other mounted file systems, it displays a message on the console and restarts the system, if possible. If it cannot restart the system or if it detects errors that it cannot repair, it displays appropriate messages on the console and returns an exit value indicating that an immediate restart is necessary.

Note: All statistics reported by **fsck** are in 512-byte blocks, regardless of the actual block size of the file system being checked. All user specifications should be specified in 512-byte blocks.

dfsck

The **dfsck** command lets you simultaneously check two file systems on two different drives. Use the *flaglist1* and *flaglist2* arguments to pass flags and parameters for the two sets of file systems. Use a - (minus) to separate the file system groups if you specify flags as part of the arguments.

The **dfsck** command permits you to interact with two **fsck** commands at once. To aid in this, **dfsck** displays the file system name with each message. When responding to a question from **dfsck**, you must prefix your response with a **1** or a **2** to indicate whether the answer refers to the first or second file system group.

Warning: Do not use **dfsck** to check the root file system (**/dev/hd0**).

Flags

-b*blocknum*	Designates a block as bad. **fsck** searches for any files that contain the specified block. If it finds any such files, it asks permission to delete them. If it finds no such files or is told to delete all such files, the specified block is added to the bad block list in i-node 1. This keeps the block out of circulation so that it cannot be allocated to any user file.
-d*blocknum*	Searches for references to a specified disk block. Whenever **fsck** encounters a file that contains a specified block, it displays the i-node number and all path names that refer to it.
-f	Performs a fast check. Under normal circumstances, the only file systems likely to be affected by halting the system without shutting down properly are those that were mounted when the system stopped. The **-f** flag tells **fsck** not to check file systems that were cleanly unmounted. The **fsck** command determines this by inspecting the **s_fmod** flag in the file system superblock. This flag is set whenever a file system is mounted and cleared when it is cleanly unmounted. If a file system was cleanly unmounted, it is unlikely to have any problems. Because most file systems are cleanly unmounted, not checking those file systems can reduce the checking time.
-i*inum*	Searches for references to a specified i-node. Whenever **fsck** encounters a directory reference to a specified i-node number, it displays the full path name of the reference.
-n	Assumes a `no` response to all questions asked by **fsck**; does not open *filesystem* for writing.
-p	Does not display messages about minor problems, but fixes them automatically. This flag does not grant the wholesale license that the **-y** flag does and is useful for performing automatic checks when the system is to be started normally. You should use this flag whenever the system is being run automatically as part of the system startup procedures.
-s[*cyl*:*skip*]	Ignores the actual free list and unconditionally reconstructs a new one. You can specify an optional interleave specification with this flag: *cyl* specifies the number of blocks per cylinder; *skip* specifies the number of blocks to skip. If you do not specify *cyl* or *skip*, **fsck** uses the interleave parameters in the superblock. The file system should be unmounted while this is done; if this is not possible, be sure that you are running no programs and that you perform a system restart immediately afterwards so that the old copy of the superblock in memory is not written to disk.
-S[*cyl*:*skip*]	Conditionally reconstructs the free list. This flag is like the **-s** flag except that the free list is rebuilt only if there are no discrepancies discovered in the file system. Using **-S** forces a `no` response to all questions asked by **fsck**. Use this flag to force free list reorganization on uncontaminated file systems.

-t*file*	Uses *file* as a scratch file if **fsck** cannot obtain enough memory to keep its tables. If you do not specify **-t** and **fsck** needs a scratch file, it prompts you for the name of the scratch file. However, if you have specified the **-p** flag, **fsck** fails. The file chosen must not be on the file system being checked. If it is not a special file, it is removed when **fsck** ends.
-y	Assumes a `yes` response to all questions asked by **fsck**. This lets **fsck** take any action that it considers necessary. Use this flag only on severely damaged file systems.

Examples

1. To check all the default file systems:

   ```
   fsck
   ```

   This checks all the file systems marked `check=true` in **/etc/filesystems**. This form of the **fsck** command asks you for permission before making any changes to a file system.

2. To fix minor problems with the default file systems automatically:

   ```
   fsck -p
   ```

3. To check a specific file system:

   ```
   fsck /dev/hd1
   ```

   This checks the unmounted file system located on the `/dev/hd1` device.

4. To simultaneously check two file systems on two different drives:

   ```
   dfsck  -p /dev/hd1  -  -p /dev/hd7
   ```

   This checks both file systems simultaneously, if the file systems on the devices `/dev/hd1` and `/dev/hd7` are located on two different drives. You can also specify the file system names that are found in the **/etc/filesystems** file.

Files

/etc/filesystems	Contains default list of file systems to check.

Related Information

The following commands: "**rc**" on page 806, "**fsdb**" on page 450, "**istat**" on page 545, "**mkfs**" on page 658, "**ncheck**" on page 683, and "**shutdown**" on page 946.

The **filesystems** and **fs** files in *AIX Operating System Technical Reference.*

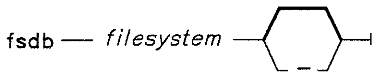
The discussion of **fsck** and **dfsck** in *Managing the AIX Operating System*

fsdb

Purpose

Debugs file systems.

Syntax



OL805244

Description

Warning: This program is not intended for use with diskette-based file systems because of the difference in superblock structure and the general format of the file system.

You can use the **fsdb** command to examine and patch a damaged file system after a system crash. It allows you to access blocks and i-numbers and to examine various parts of an i-node. You can reference components of the i-node symbolically. These features simplify procedures for correcting control-block entries or for descending the file-system tree.

The file system to be examined can be specified by a block device name, a raw device name, or a mounted file system name. In the latter case, **fsdb** determines the associated file name by reading the file **/etc/filesystems**.

Any numbers you enter are considered decimal by default, unless you prefix them with a 0 (zero) to indicate an octal number.

Because **fsdb** reads and writes one block at a time, it works with raw as well as with block I/O. It uses a buffer management routine to retain commonly used blocks of data in order to reduce the number of **read** system calls. All assignment operations write the corresponding block immediately.

Flag

-	Disables the error checking routines used to verify i-node and block addresses. The **O** subcommand toggles these routines on and off. When these routines are running, **fsdb** reads the i-size and f-size entries from the superblock of the file system.

Subcommands

The subcommands you give to **fsdb** are requests to display or modify information. A display subcommand is a block address optionally followed by a display format specification. A field modification subcommand is similar to the display subcommand but may include a subfield specification, an operator, and a value. An address specification is a number optionally followed by a type specifier and subfield specification.

The display subcommands are:

num	Display data at absolute address *num.*
*i-number*i	Display data at *i-number.*
*block-address*b	Display data at *block-address.*
*directory-slot-offset*d	Display data at *directory-slot-offset.*
q	Quit.
!	Escape to the shell.

The display formats are:

p	General display facilities
f	File display facility.

You can step through the i-node information examining each byte, word, or double word. Select the desired display mode by entering one of the following subcommands:

B	Begin displaying in byte mode.
D	Begin displaying in double-word mode.
W	Begin displaying in word mode.
O	Toggle error checking on or off.

Moving forward or backward through the i-node data is done with the following symbols:

+*num*	Move forward the specified number of units currently in effect.
-*num*	Move backward the specified number of units currently in effect.

The following symbols allow you to store the current address and return to it conveniently:

>*address*	Store *address* for later reference. If you do not specify *address*, **fsdb** stores the current address.
<	Return to the previously stored address.

The display format applied to the information at the selected address is the one currently in effect. You may receive an error message indicating improper alignment if the address you specify does not fall on an even boundary.

The display facilities display a formatted output in various styles. The current address is normalized to an appropriate boundary before display begins. The boundary advances with display and is left at the address of the last item displayed. The output can be ended at any time by pressing INTERRUPT (**Alt-Pause**).

If you enter a number after the **p** symbol, **fsdb** displays that number of entries. A check is made to detect block boundary overflows because logically sequential blocks are generally not physically sequential. If you enter a count of zero, **fsdb** displays all entries to the end of the current block.

The display formats available are:

i	Display as i-nodes.
d	Display as directories.
o	Display as octal words.
e	Display as decimal words.
c	Display as characters.
b	Display as octal bytes.
y	Display as hex bytes.

Use the **f** symbol to display data blocks associated with the current i-node. If you enter a number after **f**, **fsdb** displays that block of the file. Block numbering begins at zero. The desired display subcommand follows the block number, if present, or the **f** symbol. The display facility works for large as well as small files. It checks for special devices and also checks the data are not zero.

You can use dots (.), tabs, and spaces as subcommand delimiters, but they are not necessary. Pressing just the **Enter** key (entering a blank line) increments the current address by the size of the data type last displayed. That is, the address is set to the next byte, word, double word, directory entry or i-node, allowing you to step through a region of a file system. **fsdb** displays information in a format appropriate to the data type. Bytes, words and double words are displayed as an octal address followed by the octal representation of the data at that address and the decimal equivalent enclosed in parentheses. **fsdb** adds a **.B** or **.D** to the end of the address to indicate a display of byte or double word values. It displays directories as a directory slot offset followed by the decimal i-number and the character representation of the entry name. It displays i-nodes with labeled fields describing each element. The environment variables **NLLDATE** and **NLTIME** control the formats of the date and time.

The following mnemonics are used for the names of the fields of an i-node and refer to the current working i-node:

md	Permission mode
ln	Link count
uid	User number
gid	Group number
sz	File size
a*n*	Data block numbers (0 - 12)
at	Access time
mt	Modification time
maj	Major device number
min	Minor device number.

The general form for assigning new values is:

mnemonic operator new-value

The **fsdb** command modifies the value of the field specified by *mnemonic* according to the *operator* and *new-value.*

Valid operators include:

=	Assign *new-value* to the specified *mnemonic.*
=+	Increment the *mnemonic* by the specified *new-value.* The default *new-value* is 1.
=-	Decrease the *mnemonic* by the specified *new-value.* The default *new-value* is 1.
="	Assign character string *new-value* to the specified *mnemonic.*

Examples

The following examples show subcommands that you can use after starting **fsdb**.

1. To display an i-node:

   ```
   386i
   ```

   This displays i-number 386 in i-node format. It now becomes the current i-node.

2. To change the link count for the current i-node to 4:

   ```
   ln=4
   ```

3. To increase the link count of the current i-node by 1:

   ```
   ln=+1
   ```

4. To display part of the file associated with the current i-node:

   ```
   fc
   ```

   This displays as ASCII text block zero of the file associated with the current i-node.

5. To display entries of a directory:

   ```
   2i.fd
   ```

   This changes the current i-node to the root i-node (i-node 2), then displays the directory entries in the first block associated with that i-node.

6. To go down a level of the directory tree:

   ```
   d5i.fc
   ```

   This changes the current i-node to the one associated with directory entry 5. Then it displays the first block of the file as ASCII text (`fc`). Directory entries are numbered starting from 0 (zero).

7. To display a block when you know its block number:

   ```
   1b.p0o
   ```

   This displays the superblock (block 1) of file system in octal.

8. To change the i-number of a directory entry:

   ```
   2i.a0b.d7=3
   ```

   This changes the i-number of directory entry 7 in the root directory (2i) to 3. This example also shows how several operations can be combined on one line.

9. To change the file name of a directory entry:

   ```
   d7.nm="chap1.rec"
   ```

   This changes the name field of directory entry 7 to chap1.rec.

10. To display a given block of the file associated with the current i-node:

    ```
    a2b.p0d
    ```

    This displays block 2 of the current i-node as directory entries.

Related Information

The following command: "**fsck**, **dfsck**" on page 445.

The **fs** and **dir** files and the **environment** miscellaneous facility in *AIX Operating System Technical Reference.*

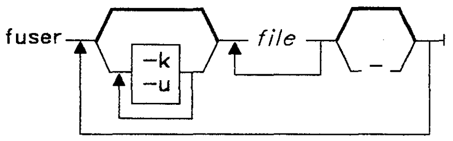
"Overview of International Character Support" in *Managing the AIX Operating System.*

fuser

Purpose

Identifies processes using a file or file structure.

Syntax



OL805055

Description

The **fuser** command lists, for local processes, the process numbers of the processes using the specified local or remote *file*. For remote processes that use local *file*s, **fuser** lists the node (NID) that has the files open. It does not list the process numbers, user names, or usage information. For block special devices, all processes using any file on that device are listed. The process number is followed by a letter indicating how the process is using the file:

c	Using *file* as the current directory
p	Using *file* as the parent of the current directory (only when in use by the system)
r	Using *file* as the root directory.

The process numbers are written as a single line to standard output, separated by spaces and ended with a single new-line character. All other output is written to standard error.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Flags

-k Sends the **SIGKILL** signal to each local process. Only the person operating with superuser authority can kill another user's process (see "**kill**" on page 552). **SIGKILL** is not sent to remote processes.

-u	Indicates the login name in parentheses after the process number. The login name is not listed for remote processes.
-	Cancels any flags selected for the previous set of file or files.

Flags may be respecified between groups of files on the command line. The new set of flags replaces the old set.

Examples

1. To list the ID numbers of the processes using the **/etc/passwd** file:

   ```
   fuser /etc/passwd
   ```

2. To list the process IDs and user names of the processes using the **/etc/filesystems** file:

   ```
   fuser -u /etc/filesystems
   ```

3. To stop all of the processes using a given disk drive:

   ```
   fuser -k -u /dev/hd1
   ```

   This lists the process ID and user name, and then stops each process that is using the `/dev/hd1` disk drive. You must have superuser authority to stop processes that belong to someone else. You might want to do this if you are trying to unmount `/dev/hd1`, and a process accessing it is preventing you from doing so.

4. To perform the actions of the previous examples in reverse order:

   ```
   fuser -k -u /dev/hd1 - -u /etc/filesystems - /etc/passwd
   ```

   Note that lone dashes before the `-u` and before `/etc/passwd` turn off both the `-k` and `-u` flags.

Files

/unix	System kernel image.
/dev/kmem	For system image.
/dev/mem	Also for system image.

Related Information

The following commands: "**killall**" on page 555, "**mount**" on page 669, and "**ps**" on page 786.

The **kill** and **signal** system calls in *AIX Operating System Technical Reference.*

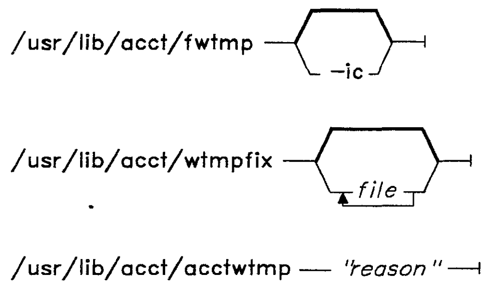
"Using Distributed Services" in *Managing the AIX Operating System.*

fwtmp

Purpose

Manipulates connect accounting records.

Syntax



OL805239

Description

fwtmp

The **fwtmp** command reads **wtmp** records from standard input and converts them to formatted ASCII records, which it writes to standard output.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Flag

-ic Reads ASCII input and writes output in binary form.

acctwtmp

The **acctwtmp** command writes to standard output a **utmp** record containing the string *reason* and the current date and time. A *reason* can contain 11 or fewer characters.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

---

wtmpfix

The **wtmpfix** command examines standard input or the named *file*s containing records in **wtmp** format, corrects the date and time stamps to make the entries consistent, and writes the corrected input to standard output. (It is necessary that date and time stamps be consistent because **acctconl** generates an error and stops when it encounters inconsistent date change records.)

Each time the date is set (on system startup or with the **date** command) a pair of date change records is written to **/usr/adm/wtmp**. The first record is the old date, denoted by the string **old time** placed in the line field and the flag **OLD_TIME** placed in the type field. The second record is the new date, denoted by the string **new time** placed in the line field and the flag **NEW_TIME** placed in the type field. The **wtmpfix** command uses these records to synchronize all date and time stamps in the file.

In addition to correcting date and time stamps, **wtmpfix** checks the validity of the name field to ensure that it consists solely of alphanumeric characters, a dollar sign ($), or spaces. If it encounters an invalid name, it changes the login name to **INVALID** and writes a diagnostic to standard error. In this way, **wtmpfix** reduces the chance that **acctcon2** will fail when it processes connect accounting records.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Files

/usr/adm/wtmp	Contains records of date changes that include an old date and a new date.
/usr/include/utmp.h	Contains history records that include a reason, date, and time.

Related Information

The following commands: "**acct/***" on page 13, "**acctcms**" on page 18, "**acctcom**" on page 20, "**acctcon**" on page 24, "**acctdisk, acctdusg**" on page 26, "**acctmerg**" on page 28, "**acctprc**" on page 30, and "**runacct**" on page 848.

The **acct** system call and the **acct** and **utmp** files in *AIX Operating System Technical Reference.*

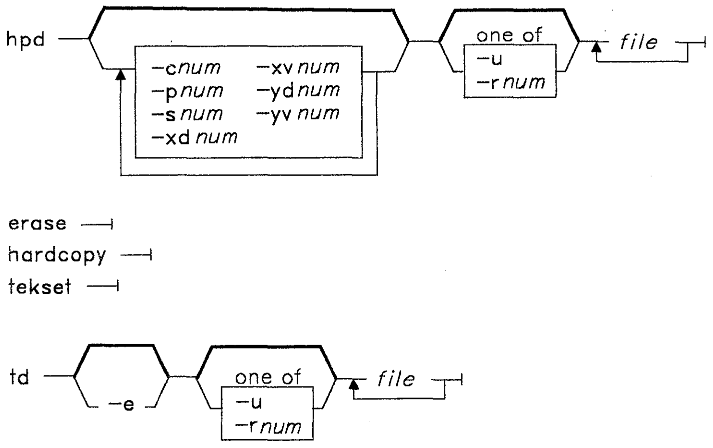
"Running System Accounting" in *Managing the AIX Operating System.*

gdev

Purpose

Provides graphical device routines and filters.

Syntax



OL777036

Description

The following commands provide various graphical device routines and filters. They all reside in the **/usr/bin/graf** directory (see "**graphics**" on page 497).

hpd

The **hpd** command takes a graphical file in **gps** format (standard input by default), and translates it into instructions for the Hewlett-Packard 7221A Graphics Plotter. See the **gps** file in *AIX Operating System Technical Reference* for a description of this file format. It computes a viewing window from the maximum and minimum points in the file, unless you specify the **-r** or **-u** flags.

Flags

c*num*	Selects character set *num*, where *num* is an integer between 0 and 5. See the Hewlett-Packard 7221A Graphics Plotter documentation for a list of these character sets.
p*num*	Selects pen-numbered *num*, where *num* is an integer between 1 and 4 inclusive.
r*num*	Displays a window on a **GPS** region, where *num* is an integer from 1 to 25 inclusive.
s*num*	Slants characters *num* degrees clockwise from the vertical.
u	Displays window on the entire **GPS** universe.
xd*num*	Sets **x** displacement of the view port's lower left corner to *num* inches.
xv*num*	Sets width of view port to *num* inches.
yd*num*	Sets **y** displacement of the view port to *num* inches.
yv*num*	Sets height of view port to *num* inches.

erase

The **erase** command sends characters to a Tektronix 4010 series storage terminal to erase the screen.

hardcopy

When issued at a Tektronix display terminal with a hard copy unit, the **hardcopy** command produces a screen copy on the unit.

tekset

The **tekset** command send characters to a Tektronix terminal to clear the display screen, set the display mode to alpha, and set characters to the smallest font.

td

The **td** command translates a **GPS** object to scope code for a Tektronix 4010 series storage terminal. It computes a viewing window from the maximum and minimum points in *file*, unless you specify the **-u** or **-r***num* flag. Standard input is the default input file.

Flags

e	Does not erase the screen before initiating display.
r*num*	Displays **GPS** region *num*, where *num* is an integer between 1 and 25 inclusive.
u	Displays the entire **GPS** universe.

Related Information

The following commands: "**ged**" on page 463, "**gend**" on page 475, and "**graphics**" on page 497.

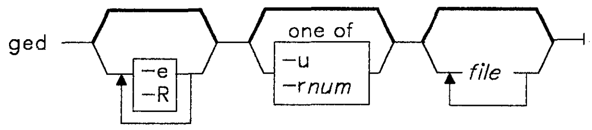
The **gps** file in *AIX Operating System Technical Reference.*

ged

Purpose

Displays, makes, and edits graphical files on Tektronix 4010 terminals.

Syntax



OL777037

Description

The **ged** command is an interactive graphical editor used to edit drawings on Tektronix 4010 series display terminals. The drawings are a sequence of objects that consist of ***lines***, ***arcs***, and ***text***. With **ged** you can view the objects at various magnifications and from various locations. The drawings are stored in graphics primitive string (**GPS**) files. If you specify **-** (minus) as the file name, **ged** reads standard input into the edit buffer.

An arc or lines object has a start point (***object-handle***), followed by zero or more points (***point-handles***). A text object has only an object-handle. These objects are positioned within a Cartesian plane (***universe***), having 64K (-32K to +32K) points (***universe-units***) on each axis. The **GPS** universe is divided into 25 equal sized areas called ***regions***. These regions are arranged in five rows of five squares each, numbered 1 to 25 from the lower left of the universe to the upper right.

The **ged** command maps rectangular areas (***windows***) from the universe onto the display screen. Windows let you view pictures from different locations and at different magnifications. The ***universe-window*** is the window with minimum magnification; that is, the window that views the entire universe. The ***home-window*** is the window that completely displays the contents of the display buffer.

Flags

-e	Does not erase the screen before the initial display
-r*num*	Displays region number *num*.
-u	Displays the entire **GPS** universe.
-R	Invokes the restricted shell on use of ! (exclamation character).

Subcommands

The **ged** subcommands are entered in ***stages***. Typically each stage ends with a **<cr>** (Return). Prior to the final **<cr>**, you may cancel the subcommand by pressing INTERRUPT (**Alt-Pause**). You can edit the input of a stage, during the stage, by using the erase and kill characters of the calling shell. The * (star) prompt indicates that **ged** is waiting at stage 1.

Each subcommand consists of a subset of the following stages:

1. ***Command line***, whose format is the same as the format of a shell command:

   subcommand-name [*-flags*] [*filename*]

   followed by pressing the **Enter** key. The *subcommand-name* consists of the first character of the subcommand. **ged** echoes the full command name and pauses for the remainder of the command line. Flags are indicated by a leading - (minus). To generate a list of **ged** subcommands, enter: ?.

2. ***Text***, a sequence of characters terminated by an unescaped **Enter**. You can have a maximum of 120 lines of text.

3. ***Points***, a sequence of one or more screen locations (maximum of 30), indicated either by the terminal cross hairs or by name. The prompt for entering points is the appearance of the cross hairs. When the cross hairs are visible, type:

sp (space)	Enters the current location as a point. The point is identified by a number.
$*num*	Enters the previous point numbered *num*.
>*x*	Labels the last point entered with the upper case letter *x*.
$*x*	Enters the point labeled *x*.
.	Establishes the previous points as the current points. At the start of a command, the previous points are those locations given with the previous command.
=	Echoes the current points.
$.*num*	Enters the point.
#	Erases the last point entered.
@	Erases all of the points entered.

4. ***Pivot***, a single location entered by pressing the **Enter** key or by using the **$** operator and indicated with a * (star).

5. ***Destination***, a single location entered by pressing the **Enter** key or by using **$** (dollar sign).

Subcommand Summary

In the following lists, characters printed in **bold** are to be entered literally. Subcommand stages are printed in ***bold italics***. Arguments surrounded by [] (brackets) are optional. Parentheses surrounding arguments separated by "or" indicate that you must specify exactly one of the arguments.

Construct Subcommands

Arc	[-echo,style,weight] ***points***
Box	[-echo,style,weight] ***text***
Circle	[-echo,style,weight] ***point***
Hardware	[-echo] ***text points***
Lines	[-echo,style,weight] ***points***
Text	[-angle,echo,height, mid-point,right-point,text, weight] ***text points***

Edit Subcommands

Delete	(-(universe or view) or ***points***)
Edit	[-angle,echo,height,style,weight] (-(universe or view) or ***points***)
Kopy	[-echo,points,x] ***points pivot destination***
Move	[-echo,points,x] ***points pivot destination***
Rotate	[-angle,echo,kopy,x] ***points pivot destination***
Scale	[-echo,factor,kopy,x] ***points pivot destination***

View Subcommands

coordinates	***points***
erase	
new-display	
object-handles	(-(universe or view) or ***points***)
point-handles	(-(*l*abelled-points or universe or view) or ***points***)
view	(-(home or universe or region) or [-x] ***pivot destination***)
x	[-view] ***points***
zoom	[-out] ***points***

Other Subcommands

quit or Quit

read [-angle,echo,height, mid-point,right-point,text, weight] ***file-name[destination]***

set [-angle,echo,factor, height,kopy,mid-point,points, right-point,style,text, weight,x]

write ***file-name***

!***command***

?

Options

Options specify parameters used to construct, edit, and view graphical objects. If a parameter used by a subcommand is not specified as an *option*, the default value for the parameter will be used (see set following). The format of subcommand *options* is:

-option[,option]

where *option* is *keyletter[value].* Flags take on the *values* of true or false indicated by + and - respectively. If no *value* is given with a flag, true is assumed.

Object Options

angle*n*	Specifies an angle of *n* degrees.
echo	When true, changes made to the display buffer are echoed to the screen.
factor*n*	Specifies a scale factor is *n* percent.
height*n*	Sets the height of text to *n* universe-units ($0 \leq n < 1280$).
kopy	When true, copies rather than moves.
mid-point	When true, uses the mid-point of a text string to locate string.
points	When true, operates on points; otherwise operates on objects.
right-point	When true, uses the rightmost point of the text string to locate string.
style*type*	Sets the line style to one of following *types*:
	so solid
	da dashed
	dd dot-dashed
	do dotted
	ld long-dashed.
text	When false, outlines rather than draws text strings.

weight*type*	Sets line weight to one of following *types*: **n** narrow **m** medium **b** bold.

Area Options

home	References the home-window.
out	Reduces magnification during zoom.
region*n*	References the region *n*.
universe	References the universe-window.
view	References those objects currently in view.
x	Indicates the center of the referenced area.

Subcommand Descriptions

Construct Subcommands

Arc
Lines Behave similarly. Each consists of a ***command line*** followed by ***points***. The first ***point*** entered is the object-handle. Successive ***points*** are point-handles. Lines connects the handles in numerical order. Arc fits a curve to the handles (currently a maximum of 3 points will be fit with a circular arc; splines will be added in a later version).

Box
Circle Special cases of Lines and Arc, respectively. Box generates a rectangle with sides parallel to the universe axes. A diagonal of the rectangle would connect the first ***point*** entered with the last ***point***. The first ***point*** is the object-handle. Point-handles are created at each of the vertices. Circle generates a circular arc centered about the ***point*** numbered zero and passing through the last ***point***. The circle's object-handle coincides with the last ***point***. A point-handle is generated 180 degrees around the circle from the object-handle.

Text
Hardware Generate ***text*** objects. Each consists of a ***command line***, ***text*** and ***points*** ***Text*** is a sequence of characters delimited by **<cr>**. Multiple lines of text may be entered by preceding a **cr** with a \ (backslash). The Text subcommand creates software-generated characters. Each line of software text is treated as a separate ***text*** object. The first ***point*** entered is the object-handle for the first line of text. The Hardware command sends the characters in ***text***, uninterpreted, to the terminal.

Edit Subcommands

Edit subcommands operate on portions of the display buffer called ***defined-areas***. A defined-area is referenced either with an area *option* or interactively. If an area *option* is not given, the perimeter of the defined-area is indicated by ***points***. If no ***point*** is entered, a small defined-area is built around the location of the **<cr>**. This is useful to reference a single ***point***. If only one ***point*** is entered, the location of the **<cr>** is taken in conjunction with the ***point*** to indicate a diagonal of a rectangle. A defined-area referenced by ***points*** will be outlined with dotted lines.

Delete	Removes all objects whose object-handle lies within a defined-area. The **u**niverse option removes all objects and erases the screen.
Edit	Modifies the parameters of the objects within a defined-area. Parameters that can be edited are:

angle	Specifies the angle of *text*
height	Specifies the height of *text*
style	Specifies the style of *lines* and *arc*
weight	Specifies the weight of *lines*, *arc*, and *text*

Kopy **M**ove	Copies (or moves) object- and/or point-handles within a defined-area by the displacement from the ***pivot*** to the ***destination***.
Rotate	Rotates objects within a defined-area around the ***pivot***. If the **k**opy flag is true, the objects are copied rather than moved.
Scale	For object whose object-handles are within a defined-area, point displacements from the ***pivot*** are scaled by **f**actor percent. If the **k**opy flag is true then the objects are copied rather than moved.

View Subcommands

coordinates	Displays the location of ***point***(s) in universe- and screen-units.
erase	Clears the screen (but not the display buffer).
new-display	Erases the screen; then displays the display buffer.
object-handles **p**oint-handles	Labels object- (and/or point-handles) that lie within the defined-area with **O** (or **P**). **p**oint-handles identifies labeled points when the **l**abeled-points flag is true.
view	Moves the window so that the universe point corresponding to the ***pivot*** coincides with the screen point corresponding to the ***destination***. Options for **h**ome, **u**niverse, and **r**egion display particular windows in the universe.

x	Indicates the center of a defined-area. Option view indicates the center of the screen.
zoom	Decreases (zoom **o**ut) or increases the magnification of the viewing window based on the defined-area. For increased magnification, the window is set to circumscribe the defined-area. For a decrease in magnification the current window is inscribed within the defined-area.

Other Subcommands

quit **Q**uit	Exit from **ged**. **q**uit responds with **?** if the display buffer has not been written since the last modification.
read	Inputs the contents of a file. If the file contains a **GPS** object, it is read directly. If the file contains text it is converted into *text* object(s). The first line of a text file begins at ***destination***.
set	When given *option*(s) resets default parameters; otherwise it prints current default values.
write	Outputs the contents of the display buffer to a file.
!	Escapes **ged** to execute an AIX Operating System command.
?	Lists **ged** subcommands.

Related Information

The following commands: "**gdev**" on page 460, "**graphics**" on page 497, and "**sh**" on page 913.

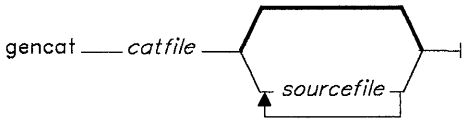
The **GPS** file in *AIX Operating System Technical Reference*.

gencat

Purpose

Creates and modifies a message catalog.

Syntax



OL805484

Description

Use **gencat** to create a message catalog from a message text source file. Because **gencat** conforms to X/Open specifications, **gencat** does not accept symbolic identifiers. You must run **mkcatdefs** as described in "Symbolic Message Identifiers" on page 474 if you want to use symbolic identifiers.

The format for **gencat** is:

$ gencat *catfile* [*sourcefiles*]

If a message catalog with the name *catfile* exists, **gencat** will modify it according to the statements in the message source files. If it does not exist, **gencat** creates a catalog file with the name *catfile.*

You may specify any number of message text source files. **gencat** processes multiple source files one after the other in the sequence that you specify them. Each successive source file will modify the catalog. If you do not specify a source file, **gencat** accepts message source data from standard input.

A message text source file is a text file that you create to enter messages into. You can use any text editor to enter the messages. Assign message set numbers and message ID numbers to each message by using the commands described in this section.

Use the **$set** command in a source file to give a group of messages a set number. The format of the **$set** command is:

$set *n* [*comment*]

The message set number is specified by *n*. All messages following the **$set** command are assigned that set number, up until the next occurrence of a **$set** command. You must specify at least one set number if the source file contains message text. The set numbers must be assigned in ascending order, but need not be contiguous. Large gaps in the number sequence do not affect performance, but will increase the size of the catalog. There is no performance advantage to using more than one set number in a catalog. Set numbers must start at 1. The highest set number in a X/Open-conforming application is 255.

You may include a comment in the **$set** command, but it is not required. The following example includes a comment:

```
$set 10  Communication Error Messages
```

Use the **$delset** command to remove all of the messages belonging to the specified set from a catalog. The format of the **$delset** command is:

`$delset` *n* [*comment*]

The message set is specified by *n*. The **$delset** command must be placed in the proper set number order with respect to any **$set** commands in the same source file. You may include a comment in the **$delset** command also.

You may include a comment line anywhere in the source file, except within message text. Indicate comments as shown below:

`$` [*comment*]

You must leave at least one space after the **$**.

Enter the message text and assign message ID numbers as follows:

m message-text

This assigns the message ID number *m* to the text that follows it. You must leave at least one space after the message number.[1] Message numbers must be in ascending order within a single message set, but need not be contiguous. Large gaps in the number sequence do not affect performance, but will increase the size of the catalog. In an X/Open-conforming application, message numbers must be in the range 1-32767.

All text following the message number is included as message text, up to the end of the line. Use the escape character \ to continue message text on the following line. The \ must be the last character on the line. Consider the following example:

```
5 This is the text associated with \
message number 5.
```

1 AIX allows any amount of white space after the message ID number; however X/Open specifies that you leave only one space between the message number and the message text.

These two lines define the single-line message:

```
This is the text associated with message number 5.
```

The escape character \ may be used to include special characters in the message text. These special characters are defined as follows:

\n	Performs a new-line function when the message is displayed.
\t	Inserts a horizontal tab character when the message is displayed.
\v	Inserts a vertical tab when the message is displayed.
\b	Performs a backspace function when the message is displayed.
\r	Inserts a carriage return character when the message is displayed.
\f	Inserts a form feed character when the message is displayed.
\\\\	Displays the \ (backslash) character in the message.
\ddd	Displays the single-byte character associated with the octal value represented by the valid octal digits *ddd*. One, two, or three octal digits may be specified; however you must include leading zeros if the characters following the octal digits are also valid octal digits. For example, the octal value for **$** is **44**. To display `$5.00` use `\0445.00`, not `\445.00`, or the 5 will be parsed as part of the octal value.
\**x***dddd*[2]	Displays the single-byte or double-byte character associated with the hexadecimal value represented by the four valid hexadecimal digits *dddd*. You may specify one, two, three, or four digits, but you must include leading zeros to avoid parsing errors (see \*ddd*).

You can also include **printf** conversion specifications in messages that are displayed by applications using **printf** or **NLprintf** (see **printf** in *AIX Operating System Technical Reference*). If you display a message from a shell script with **dspmsg**, the message can contain the **%s** or **%*n*$s** conversion specifications (see "**dspmsg**" on page 359).

You can use the **$quote** command in a message source file to define a character for delimiting message text. The format for this command is:

`$quote` [*char*] [*comment*]

Use the specified character before and after the message text as shown in the following example source file:

2 This escape sequence is an AIX extension to X/Open specifications.

```
$quote "    Use a double quote to delimit message text

$set 10            Message Facility - Quote command messages

1   "Use the $quote command to define a character \
\n for delimiting message text"

2   "You can include the \"quote\" character in a message \n \
by placing a \\ in front of it"

3   You can include the "quote" character in a message \n \
by having another character as the first nonblank \
\n character after the message ID number

$quote

4   You can disable the quote mechanism by \n \
using the $quote command without \n a character \
after it
```

In this example, the **$quote** command defines the double quote (") as the ***quote*** character. The quote character must be the first non-blank character following the message number. Any text following the next occurrence of the quote character is ignored.

The example also shows two ways the quote character can be included in the message text:

- Place a \ in front of the quote character.
- Use some other character as the first non-blank character following the message number. This disables the quote character only for that message.

The example shows these other things:

- A \ is still required to split a quoted message across lines.
- To display a \ in a message you must place another \ in front of it.
- You can format your message with a new-line character by using **\n**.
- If you use the **$quote** command with no character argument, you disable the quote mechanism.

After entering your messages into a source file you must use the message facility program **gencat** to process the source file to create a message catalog.

Symbolic Message Identifiers

AIX provides a mechanism that allows symbolic references to messages by letting you use alphanumeric identifiers instead of set numbers and message ID numbers.[3] You assign the identifiers to sets and messages in the source file in the same manner that you assign set numbers and message ID numbers.

The symbolic identifiers can contain ASCII letters, digits, and underscores. The first character cannot be a digit. The maximum length cannot exceed 64 bytes.

The following example shows a message source file with symbolic message identifiers:

```
$set symbolic      Message Facility - Symbolic ID's
$quote *

ID_names  *Symbolic identifier syntax: \n \
\talphanumerics or underscores \n \
\tnon-digit first character \n \
\t64 byte maximum length *

set_use   *To assign set ID: \n \
\t$set "identifier" [comment] *

msg_use   *To assign message ID: \n \
\t"identifier" message-text *
```

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Related Information

The following commands: "**dspcat**" on page 357, "**dspmsg**" on page 359, "**mkcatdefs**" on page 651, and " **runcat**" on page 852.

The **catopen**, **catgets**, **catgetamsg**, **catclose**, **NLcatopen**, **NLcatgets**, and **NLgetamsg** files in *AIX Operating System Technical Reference.*

The discussion of **gencat** in *AIX Operating System Programming Tools and Interfaces.*

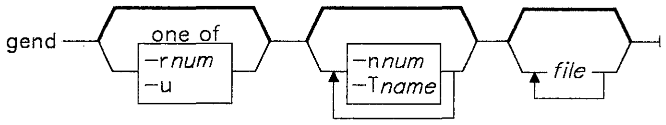
[3] Symbolic references are not defined by the X/Open specification. This mechanism is an AIX extension.

gend

Purpose

Provides a general graphics device backend.

Syntax



OL805458

Description

The **/usr/bin/graf/gend** command displays **GPS** files on the graphics output devices supported by the Advanced Display Graphics Support Library (GSL). For more information about GSL, see the “Advanced Display Graphics Support Library” in *AIX Operating System Technical Reference.* By default, **gend** reads standard input and writes to the current display (see “**display**” on page 332), but **gend** can also drive printers and plotters if you have installed the VDI drivers that are in the Extended Services Program. You can specify the name of one or more **GPS** files on the command line. If you enter a file name of - (minus), **gend** reads standard input.

When **gend** displays an image, it opens a new virtual terminal. You can move to and from this virtual terminal by pressing Next Window (**Alt-Action**). See “**open**” on page 728 and *Using the AIX Operating System* for information on virtual terminals. To end **gend** and close the virtual terminal, press END OF FILE (**Ctrl-D**).

Note: The **gend** command produces the standard **GPS** line style attributes with one exception. Line style 4 (long dashed) is rendered as dash-dot-dot.

Flags

-n*num* Specifies the number of chords per circle. Legal values are **64**, **128**, **256**, or **512**. The default value is **128**.

Rather than drawing truly circular arcs or circles, **gend** converts them into a series of very short line segments (***chords***), whose end points lie on the circle. For most devices and images, the default value of 128 is satisfactory. The higher values give a smoother image; the lower value provides faster drawing time.

-r*num*	Displays data in **GPS** region *num*. A **GPS** object is defined in a Cartesian plane of 64K points on each axis. The plane, or universe, is divided into 25 square regions numbered 1 to 25 from the lower left to the upper right.
-u	Displays data in the entire **GPS** universe.
-T*name*	Uses the device specified by the *name* environment variable. The default is the current display (this must be supported by **/dev/hft**). When the image is to be displayed on other devices, you must ensure that the proper VDI device handler is installed.

Files

/usr/bin/graf/gend	The general devices backend.
/tmp/dev.XXXXXX	Temporary file.

Related Information

The following commands: "**ged**" on page 463, "**gdev**" on page 460, "**graphics**" on page 497, and "**open**" on page 728.

"Advanced Display Graphics Support Library" in *AIX Operating System Technical Reference*

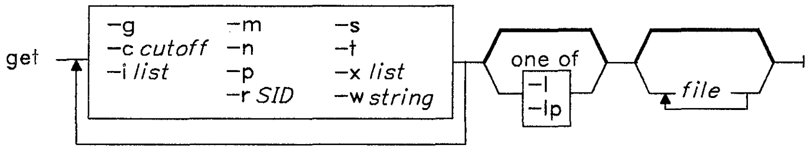
Installing programs in *Installing and Customizing the AIX Operating System.*

get

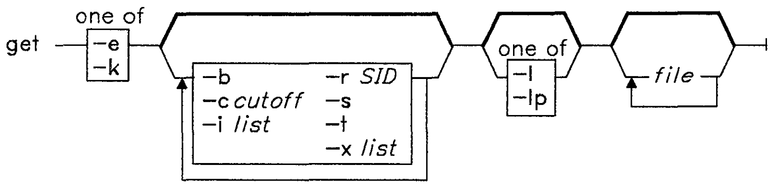
Purpose

Creates a specified version of a Source Code Control System (SCCS) file.

Syntax



OL805058



OL805355

Description

The **get** command reads the specified versions of the named Source Code Control System (***SCCS***) *file*s, creates an ASCII text file for each *file* according to the specified flags, and writes each text file to a file with the same name as the original SCCS file without the **s.** (s period) prefix (the ***g-file***). The flags and *file*s can be specified in any order, and all flags apply to all named files.

If you specify a directory in place of *file*, **get** performs the requested actions on all the files in the directory that begin with the **s.** prefix. If you specify a - (minus) in place of a *file*, **get** reads standard input and interprets each line as the name of an SCCS file. **get** continues to read input until it reads END OF FILE (**Ctrl-D**).

If the effective user has write permission in the directory containing the SCCS files but the real user does not, then only one file can be named when the **-e** flag is used.

If you are not familiar with the terms ***SID*** and ***delta*** or you do not know the numbering system of the deltas, see *AIX Operating System Programming Tools and Interfaces* for more information.

SCCS Files

In addition to the file with the **s.** prefix (the ***s-file***), **get** can create several auxiliary files: the ***g-file***, ***l-file***, ***p-file***, and ***z-file***. These files are identified by their ***tag***, the letter before the hyphen. **get** names auxiliary files by replacing the leading **s.** in the SCCS file name with the proper tag, except for the g-file, which is named by removing the **s.** prefix. So, for a file named **s.sample**, the auxiliary file names would be **sample**, **l.sample**, **p.sample**, and **z.sample**.

These files serve the following purposes:

s-file This file contains the original file text and all the changes (***deltas***) made to the file. It also includes information about who can change the file contents, who has made changes, when those changes were made, and what the changes were. You cannot edit this file directly since the file is read-only. It contains the information needed by the SCCS commands to build the g-file, the file you can edit.

g-file The g-file is an ASCII text file that contains the text of the SCCS file version that you specify with the **-r** flag (or the latest trunk version by default). You can edit this file directly. When you have made all your changes and you want to make a new delta to the file, you can then apply the **delta** command to the file. **get** creates the g-file in the current directory.

The **get** command creates a g-file whenever it runs, unless the **-g** flag or the **-p** flag is specified. The real user owns it (not the effective user). If you do not specify the **-k** or the **-e** flag, the file is read-only. If the **-k** or the **-e** flag is specified, the owner has write permission for the g-file. You must have write permission in the current directory to create a g-file.

l-file The **get** command creates the l-file when the **-l** flag is specified. The l-file is a read-only file. It contains a table showing which deltas were applied in generating the g-file. You must have write permission in the current directory to create an l-file. Lines in the l-file have the following format:

1. A blank character if the delta was applied; a * appears otherwise.
2. A blank character if the delta was applied or was not applied and ignored; a * appears if the delta was not applied and was not ignored.
3. A code indicating a special reason why the delta was or was not applied:

Blank	Included or excluded normally.
I	Included using the **-i** flag.
X	Excluded using the **-x** flag.
C	Cut off using the **-c** flag.

4. The SID.
5. The date and time the file was created.
6. The login name of person who created the delta.

Comments and MR data follow on subsequent lines, indented one horizontal tab character. A blank line ends each entry.

For example, for a delta cutoff with the **-c** flag, the entry in the l-file might be:

```
**C 1.3 85/03/13 12:44:16 pat
```

and the entry for the initial delta might be:

```
    1.1 85/02/27 15:42:20 pat
date and time created 85/02/27 15:42:20 by pat
```

p-file The **get** command creates the p-file when the **-e** or the **-k** flag is specified. The p-file passes information resulting from a **get -e** to a **delta** command. The p-file also prevents a subsequent execution of **get** with a **-e** flag for the same SID until **delta** is run or the joint edit key letter (j) is set in the SCCS file. The **j** key letter allows several **get**s on the same SID. The p-file is created in the directory containing the SCCS *file*. To create a p-file in the SCCS directory, you must have write permission in that directory. The permission code of the p-file is read-only to all but its owner, and it is owned by the effective user. The p-file contains:

- The current SID
- The SID of new delta to be created
- The user name
- The date and time of the **get**
- The **-i** flag, if it was present
- The **-x** flag, if it was present.

The p-file contains an entry with the preceding information for each pending delta for the file. No two lines have the same new delta SID.

z-file The z-file is a lock mechanism against simultaneous updates. The z-file contains the binary process number of the **get** command that created it. It is created in the directory containing the SCCS file and exists only while the **get** command is running.

When you use the **get** command, it displays the SID being accessed and the number of lines created from the SCCS file. If you specify the **-e** flag, the SID of the delta to be made appears after the SID is accessed and before the number of lines created. If you specify more than one file, or a directory, or standard input, **get** displays the file name before each file is processed. If you specify the **-i** flag, **get** lists included deltas below the word `Included`. If you specify the **-x** flag, **get** lists excluded deltas below the word `Excluded`.

Identification Keywords

You can use identification keywords in your files to insert identifying information. These keywords are replaced by their values in the g-file when **get** is invoked without the **-e** or **-k** flag. The following identification keywords can be used in SCCS files:

%M%	Module name: the value of the **m** flag in the SCCS file.
%I%	The SID (%R%.%L%.%B%.%S%) of the g-file.
%R%	Release.
%L%	Level.
%B%	Branch.
%S%	Sequence.
%D%	Date of the current **get** (YY/MM/DD).
%H%	Date of the current **get** (MM/DD/YY).
%T%	Time of the current **get** (HH:MM:SS).
%E%	Date newest applied delta was created (YY/MM/DD).
%G%	Date newest applied delta was created (MM/DD/YY).
%U%	Time newest applied delta was created (HH:MM:SS).
%Y%	Module type: the value of the **t** flag in the SCCS file.
%F%	SCCS file name.
%P%	Full path name of the SCCS file.
%Q%	The value of the **q** flag in the file.
%C%	The current line number. This keyword is intended for identifying messages output by the program. It is not intended to be used on every line to provide sequence numbers.
%Z%	The 4-character string `@(#)` recognized by the **what** command.
%W%	A shorthand notation for constructing **what** strings for AIX program files. Its value is the characters and key letters: `%W% = %Z%%M%<horizontal-tab>%I%`
%A%	Another shorthand notation for constructing **what** strings for non-AIX program files. Its value is the key letters: `%A% = %Z%%Y% %M% %I%%Z%`

The following table illustrates how **get** determines the SID of the file it retrieves, and what the pending SID is. The column **SID Specified** shows the various ways the SID can be specified with the **-r** flag. The two columns illustrate the various conditions that can exist, including whether or not the **-b** flag is used with the **get -e**. The **SID Retrieved** indicates the SID of the file that makes up the g-file. The **SID of Delta to be Created** column indicates the SID of the version that will be created when **delta** is applied.

SID Specified	-b Used	Other Conditions	SID Retrieved	SID of Delta to be Created
none[1]	no	R defaults to mR[2]	mR.mL	mR.(mL + 1)
none[1]	yes	R defaults to mR	mR.mL	mR.mL.(mB + 1).1
(R)elease	no	R > mR	mR.mL	R.1[3]
R	no	R = mR	mR.mL	mR.(mL + 1)
R	yes	R > mR	mR.mL	mR.mL.(mB + 1).1
R	yes	R = mR	mR.mL	mR.mL.(mB + 1).1
R	N/A	R < mR and R does not exist	hR.mL[4]	hR.mL.(mB + 1).1
R	N/A	R < mR and R exists	R.mL	R.mL.(mB + 1).1
R.(L)evel	no	No trunk successor	R.L	R.(L + 1)
R.L	yes	No trunk successor	R.L	R.L(mB + 1).1
R.L	N/A	Trunk successor in release ≥ R	R.L	R.L.(mB + 1).1
R.L.(B)ranch	no	No branch successor	R.L.B.mS	R.L.B.(mS + 1)
R.L.B	yes	No branch successor	R.L.B.mS	R.L.(mB + 1).1
R.L.B.(S)equence	no	No branch successor	R.L.B.S	R.L.B.(S + 1)
R.L.B.S	yes	No branch successor	R.L.B.S	R.L.(mB + 1).1
R.L.B.S	N/A	Branch successor	R.L.B.S	R.L.(mB + 1).1

[1] Applies only if the **d** (default SID) flag is not present in the file (see "**admin**" on page 41).
[2] The mR indicates the maximum existing release.
[3] Forces creation of the first delta in a new release.
[4] The hR is the highest existing release that is lower than the specified, nonexistent, release R.

Figure 2. SID Determination

Flags

-b Specifies that the delta to be created should have an SID in a new branch. The new SID is numbered according to the rules stated in Figure 2. You can use **-b** only with the **-e** flag. It is only necessary when you want to branch from a ***leaf delta*** (a delta without a successor). Attempting to create a delta at a nonleaf delta automatically results in a branch, even if the **b** header flag is not set. If you do not specify the **b** header flag in the SCCS file, **get** ignores the **-b** flag because the file does not allow branching (see the discussion of header flags on page 44).

-c*cutoff* Specifies a *cutoff* date and time, in the form: *YY*[*MM*[*DD*[*HH*[*MM*[*SS*]]]]] **get** includes no deltas to the SCCS file created after the specified *cutoff* in the g-file. The values of any unspecified items in the *cutoff* default to their maximum allowable values. Thus, a cutoff date and time specified with only the year (*YY*) would specify the last month, day, hour, minute, and second of that year. Any number of nonnumeric characters can separate the two-digit items of the *cutoff* date and time. This allows you to specify a date and time in a number of ways, as follows:

```
-c85/9/2,9:00:00
-c"85/9/2 9:00:00"
"-c85/9/2 9:00:00"
```

-e Indicates that the g-file being created is to be edited by the user applying **get**. The changes are recorded later with the **delta** command. **get -e** creates a p-file that prevents other users from issuing another **get -e** and editing a second g-file on the same SID before **delta** is run. The owner of the file can override this restriction by allowing joint editing on the same SID through the use of the **admin** command with the **-fj** flag. Other users, with permission, can obtain read-only copies by using **get** without the **-e** flag. The **get -e** command enforces SCCS file protection specified via the ceiling, floor, and authorized user list in the SCCS file (see "**admin**" on page 41).

-g Suppresses the actual retrieval of text from the SCCS file. Use the **-g** flag primarily to create an l-file or to verify the existence of a particular SID. Do not use it with the **-e** flag.

-i*list* Specifies a *list* of deltas to be included in the creation of a g-file. The ***SID list format*** consists of a combination of individual SIDs separated by commas and SID ranges indicated by two SIDs separated by a hyphen. You specify the same SIDs with both the following command lines:

```
get -e -i1.4,1.5,1.6 s.file
get -e -i1.4-1.6 s.file
```

You can specify the SCCS identification of a delta in any form shown in the **SID Specified** column of Figure 2 on page 481. **get** interprets partial SIDs as shown in the **SID Retrieved** column of the table.

-k Suppresses replacement of identification keywords in the g-file by their value (see "Identification Keywords" on page 480). The **-k** flag is implied by the **-e** flag. If you accidentally ruin the g-file created by **get** with an **-e** flag, you can recreate it by reissuing the **get** command with the **-k** flag in place of the **-e** flag.

-l[p] Writes a delta summary to an l-file. If you specify **-lp**, the delta summary is written to standard output, and **get** does not create the l-file. Use this flag to determine which deltas were used to create the g-file currently in use. See "SCCS Files" on page 478 for the format of the l-file.

-m	Writes before each line of text in the g-file the SID of the delta that inserted the line into the SCCS file. The format is: SID tab line of text
-n	Writes the value of the %M% keyword before each line of text in the g-file (see "Identification Keywords" on page 480 for information on keywords). The format is the value of %M%, followed by a horizontal tab, followed by the text line. When both the **-m** and **-n** flags are used, the format is: **%M%** value tab SID tab line of text
-p	Writes the text created from the SCCS file to standard output and does not create a g-file. **get** sends output normally sent to standard output to file descriptor 2 instead. If you specify the **-s** flag with the **-p** flag, output normally sent to standard output does not appear anywhere. Do not use **-p** with the **-e** flag.
-r*SID*	Specifies the SCCS identification string (SID) of the SCCS file version to be created. Figure 2 on page 481 shows what version of a file is created and the SID of the pending delta as functions of the SID specified.
-s	Suppresses all output normally written to standard output. Error messages (written to standard error output), remain unaffected.
-t	Accesses the most recently created delta in a given release or release and level. Without the **-r** flag, **get** accesses the most recent delta regardless of its SID.
-w*string*	Substitutes *string* for the %W% keyword in g-files not intended for editing (see "SCCS Files" on page 478 for information on g-files).
-x*list*	Excludes a *list* of deltas in the creation of a file. See the **-i** flag for the SID list format on page 482.

Examples

1. To get an SCCS file for editing:

   ```
   get  -e  s.prog.c
   ```

   This creates a file named `prog.c` that only you have permission to modify. No one else can use `prog.c` or `s.prog.c` until you use the **delta** command to indicate that you are finished.

2. To get an SCCS file for reading:

   ```
   get  s.prog.c
   ```

   This creates a file named `prog.c` that anyone can read, but that no one can modify. You can do this before searching files with the **grep** command or before compiling programs that are controlled with SCCS. If you are also using the **make** command to manage the development of a software project, **make** automatically does the **get** before compiling a program.

Related Information

The following commands: "**admin**" on page 41, "**delta**" on page 310, "**help**" on page 513, "**prs**" on page 781, and "**what**" on page 1213.

The **sccsfile** file in *AIX Operating System Technical Reference.*

The discussion of SCCS in *AIX Operating System Programming Tools and Interfaces.*

getopt

Purpose

Parses command line flags and parameters.

Syntax

```
set ——  ‾‾  —— `getopt —— opstring —— $*` ——┤ 1
```

¹This command is not entered on the command line, but is used in shell procedures.

OL805050

Description

The **getopt** command is used to break up flags and parameters in command lines for easy parsing by shell procedures and to check for valid flags. *opstring* is a string of recognized flags (see the **getopt** subroutine call in *AIX Operating System Technical Reference*). Extended characters are not permitted. If a letter within *opstring* is followed by a colon, the flag is expected to take a modifying parameter that may or may not be separated from it on the command line by one or more tabs or spaces. If you specify -- as the last flag on a command line processed by **getopt**, **getopt** recognizes it and stops its processing; otherwise, **getopt** creates the terminating --. In either case, **getopt** places it at the end of the flags.

When the output from **getopt** is passed by command substitution to the shell **set** command, **set** resets all of the shell positional parameters ($1, $2 . . .) so that each flag is preceded by a - (minus) and occupies its own positional parameter. Each parameter (for example, file names and other parameters) is also parsed into its own positional parameter.

The **getopt** command writes a message to standard error when it encounters a flag not included in *opstring*. The **set** command returns a nonzero value if a flag appears on the command line but is not specified in *opstring*. Consequently, you can test the validity of command flags by testing the value of the shell variable **$?**. If it is nonzero, the command line contains an unrecognized flag.

Example

The following shell procedure is a front end to the **ar** command. It uses **getopt** to separate the flags and parameters, then translates them into the **ar** command syntax and runs **ar** with these flags.

```
# @(#) lib: Front end to the ar command.
#
# Accepts the following flags:
#    ar flags with "-" prefixes.  See the ar command.
#    -L library   The default library is "libsubs.a".
# Note: "lib -r -b sub1.o -v -l newsub.o" performs
#        "ar rbvl sub1.o libsubs.a newsub.o".
#        The ar command DOES interpret this correctly.

set -- `getopt clsvmrua:b:i:dpqtxwL: $*`
if [ $? != 0 ]          # Test for syntax error
then
   exit 2
fi

FLAGS=  POSNAME=  LIBRARY=libsubs.a     # Default library name
while [ $1 != -- ]
do
   case $1 in
      -L)
         LIBRARY=$2
         shift; shift   # Shift past the -L and library name
         ;;
      -a|-b|-i)
         FLAGS=$FLAGS`expr "$1" : "-\(.\)"`
         POSNAME=$2
         shift; shift   # Shift past the flag and parameter
         ;;
      -*)               # Strip the "-" from the flag
         FLAGS=$FLAGS`expr "$1" : "-\(.\)"`
         shift
         ;;
   esac
done

shift                   # Shift past the "--" from getopt

FLAGS=${FLAGS:-vt}      # Default if action not specified

ar $FLAGS $POSNAME $LIBRARY $*
```

If this shell procedure is stored in a file named `lib`, then all of the following commands are equivalent:

```
lib -L mylib.a -v -r -b putfld.o getnam.o getfld.o getaddr.o
lib -Lmylib.a -v -r -bputfld.o getnam.o getfld.o getaddr.o
lib -Lmylib.a -vrbputfld.o getnam.o getfld.o getaddr.o
lib -Lmylib.a -v -rbputfld.o -- getnam.o getfld.o getaddr.o
```

In each of these cases, **getopt** breaks down the command into:

```
-L mylib.a -v -r -b putfld.o -- getnam.o getfld.o getaddr.o
```

The **getopt** command writes to its standard output. Because this command is enclosed in ` ` (grave accents), the shell takes its standard output and uses it to construct the command:

```
set -- -L mylib.a -v -r -b putfld.o -- getnam.o
getfld.o getaddr.o
```

This is called ***command substitution***. For more details, see "Command Substitution" on page 925.

The **set** command (page 933) sets the positional parameters $1, $2, $3 . . . to each of the values -L, `mylib.a`, -v . . . , respectively.

The shell procedure then uses the positional parameters to construct and run the command:

```
ar vrb putfld.o mylib.a getnam.o getfld.o getaddr.o
```

The **ar** command (page 55) accepts the flags in any order. Therefore, you can specify flags to `lib` in any order, as long as a parameter immediately follows a **-a**, **-b**, **-i**, or **-L** flag, and all the flags come before any file names. This means that:

```
lib -bputfld.o -rv -Lmylib.a getnam.o getfld.o getaddr.o
```

produces the command:

```
ar brv putfld.o mylib.a getnam.o getfld.o getaddr.o
```

which performs the same action as each of the previous commands. See "**test**" on page 1064 and "**expr**" on page 412 for more information about these commands.

Related Information

The following command: "**sh**" on page 913.

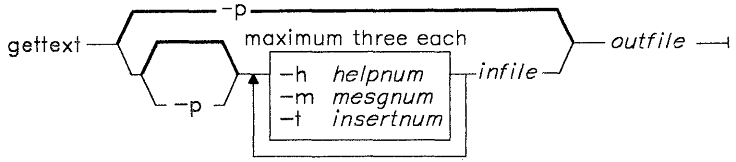
The **getopt** subroutine in *AIX Operating System Technical Reference.*

gettext

Purpose

Extracts message/insert/help descriptions.

Syntax



OL805130

Description

The **gettext** command gets message, insert, or help descriptions from *infile* and places the descriptions in *outfile*. If you specify the **-p** flag or **gettext** *outfile*, **gettext** places a message/insert/help template in *outfile*. When you have your message, insert, or help descriptions or your message/insert/help template in *outfile*; you can edit *outfile*.

The *outfile* is an AIX ASCII file that consists of a header to identify the component and a group of message/insert/help descriptions. The contents of the message/insert/help descriptions includes a delimiter, control information and message/insert/help text. See *AIX Operating System Programming Tools and Interfaces* for a description of the *outfile* format and contents.

Flags

-h *helpnum*	Extracts help information from *infile*. You specify the index value used for the desired help number with *helpnum*.
-m *mesgnum*	Extracts message information from *infile*. You specify the index value used for the desired message number with *mesgnum*.
-p	Makes a message/insert/help template for *outfile*.
-t *insertnum*	Extracts text insert information from *infile*. You specify the index value used for the desired insert number with *insertnum*.

The syntax for the *mesgnum*, *insertnum*, and *helpnum* parameters is as follows:

num-num	Retrieves index *num*bers *num* to *num*.
num,num . . .	Retrieves a list of index *num*bers specified with *num*, *num*, *num*, and so on (maximum of 50 numbers).
num-	Retrieves index numbers equal to and larger than *num*.
-*num*	Retrieves index numbers from one to *num*.

Related Information

The following command: "**puttext**" on page 796.

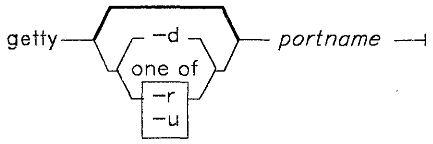
The discussion of **gettext** in *AIX Operating System Programming Tools and Interfaces*.

getty

Purpose

Sets the characteristics of ports.

Syntax



OL805333

Description

The **init** process runs the **getty** command for each *portname* enabled for login. Its primary function is to set the characteristics of the port specified by *portname.* Port characteristics include:

- Bidirectional use (tty line can be used in both directions)
- Line speed (baud rate)
- Parity
- Carriage return, tab, new-line, and form feed delays
- Character set mapping, such as lowercase to uppercase, carriage return to new-line translation, and tab expansion
- Extended character support
- Character erase and line erase editing characters
- Local or remote echo
- Screen length for paging.

The **getty** command obtains these settings by reading the port attributes specified in the **/etc/ports** configuration file and by observing the behavior of the port itself. (For details regarding the format of **/etc/ports**, see *AIX Operating System Technical Reference.* For the **logmodes** and **runmodes** parameter settings, see "**stty**" on page 1018.)

When **getty** is invoked, it opens the specified port. However, if carrier detection (modem control) is available on the port, **getty** cannot open the port until the carrier is present. Once the port is opened, **getty** sets the work station attributes according to the first parameters in the **ports** file (for example, **speed**, **logmodes**, **parity**, **erase**, and **kill**). Then **getty** writes the message **herald** to the port and reads a login name from the port.

Note: If the login name contains extended characters, these extended characters are translated to the single ASCII characters that most resemble them.

Japanese Language Support Information

Login names are limited to ASCII characters.

If a framing error occurs while reading, either because a user generates a **BREAK** signal from the work station or because the line speed is not the same as that of the transmitting work station, the port parameters are reset to the next combination specified in the **ports** file.

Once **getty** reads a login name, it does the following:

1. Resets the work station modes according to the **runmodes** parameter.
2. Turns on carriage-return-to-new-line mapping if the login name was ended by a carriage return.
3. Turns on lowercase-to-uppercase mapping if the alphabetic characters in the login name were all uppercase.
4. Executes the program specified by the **logger** parameter. That program, defaulting to **/bin/login**, runs in the same process as **getty**, not as its child.

Any additional arguments entered after the login name are passed to the **logger** program. The **login** command interprets these as shell variable settings and places them in the environment.

On dial-in ports, it is often necessary to set no parity generation or checking as a default, but to permit the user to select parity as an option. For example, the following line in the **/etc/ports** file:

```
parity = none,odd+inpck,even+inpck
```

accepts logins with any parity. However, if a user generates **BREAK** before typing a login name, **getty** sets the port to generate odd parity and to check incoming characters for odd parity, while two **BREAK**s generate and check for even parity. Similarly, the following line:

```
speed=1200,300
```

works with 1200 baud, reverting to 300 baud when a **BREAK** is received before the login name. The default **runmodes** parameter (which must appear on one line in the **ports** file), is generally satisfactory. However, for work stations that have built-in tabs to every eight character positions and do not require tab delays, eliminating the `tab3` from the default in **/etc/ports** will provide faster output with less system load.

Notes for secure Command: After you run the **secure** command, the **/etc/ports** file can contain new attributes. These attributes are used by **getty** to manage the trusted path.

The following is a list of the possible attributes:

sak Turns on secure attention key (SAK) detection for the port and causes **getty** to perform a **frevoke** system call.

synonym This attribute specifies another port that requires the same protection and ownership as the port on which **getty** is working.

shell This attribute names the terminal manager program, usually **/bin/actman**. This program is passed to the **logger** program and is executed as the user's login shell when the user logs in successfully.

Special Purpose Options

If there is a **timeout** keyword in the **ports** file, **getty** waits only the specified number of seconds for a response to the herald before advancing to the next port settings or, after all the settings are exhausted, exiting. If there is a **program** keyword for the port, then instead of displaying the herald and gathering a login name, it executes the specified program immediately. This feature is a general mechanism for supporting special service ports such as network mail daemons that need to be spawned when a connection is made from the outside world. As a special case, if you specify:

```
program = HOLD
```

the **runmodes**, **owner**, and **protection** of the port are set and **getty** holds the port open indefinitely, thereby preventing the port modes from reverting to their open-default settings. This is useful, for example, in setting the modes on serial printer ports when it is inconvenient or impossible to have the programs that use them do so.

Flags

-d Uses standard input as the work station for which parameters are to be set according to those governing *portname*. Instead of executing a logger or a program, **getty** displays the name of the program that would have been run.

-r Makes the port available for shared (bidirectional) use. With this flag, **getty** attempts to create a lock file in **/etc/locks** with the name of the device. This file can then be used by **uucp** to determine the status of the line. If the lock fails (because some other process is using the line), **getty** waits until the lock file is removed, then exits. The **init** command creates a new **getty** to attempt the locking process again.

-u Makes the port available for shared (bidirectional) use without displaying the login herald. This flag is used for direct lines or lines that have intelligent modems that need to return immediately on opening a port. This prevents **getty** from communicating with a **getty** on the remote system or modem.

Example

To test a new **/etc/ports** entry, enter the following:

```
getty -d /dev/tty5
```

This tests a new port definition for **/dev/tty5** by simulating the login sequence of this device at your work station.

Files

/bin/actman	A terminal manager program.
/etc/locks	Contains terminal devices to be locked
/etc/ports	Specifies terminal mode
/bin/login	Contains the authorization and identification program
/bin/setmaps	Contains characters for National Language Support (NLS)

Related Information

The following commands: "**login**" on page 584, "**init**" on page 521, and "**stty**" on page 1018.

The **tty** and **ports** files in *AIX Operating System Technical Reference.*

"Overview of International Character Support" in *Managing the AIX Operating System.*

The discussion of the trusted path in *Managing the AIX Operating System.*

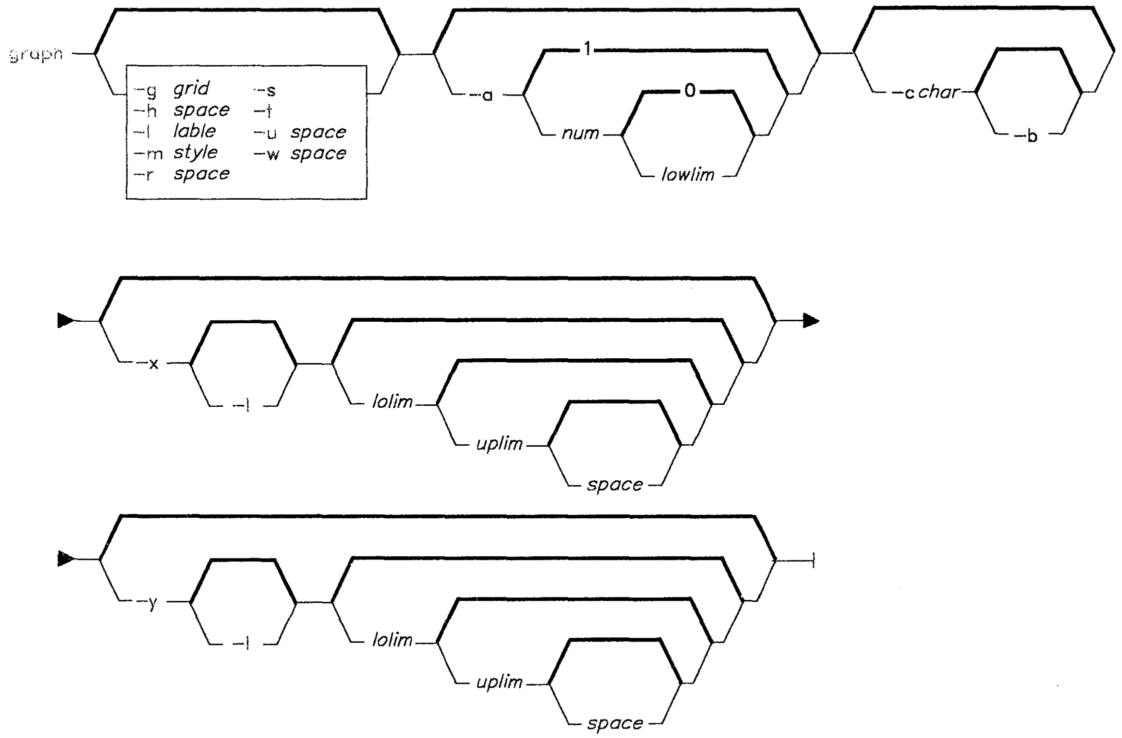
The discussion of Japanese Language Support in *Japanese Language Support User's Guide.*

graph

Purpose

Draws a graph.

Syntax



OL805429

Description

The **graph** command reads pairs of numbers from standard input, where each pair is the x and y coordinates of a point on a graph. It processes the data that allows the successive points to be connected by straight lines when printed and then writes the graph to standard output. See "**tplot**" on page 1079 for information on how to code the output for printing.

In the input, non-numeric strings following the coordinates of a point are labels. Labels begin on the point. Labels can be surrounded with " (double quotation marks), in which case they can be empty or contain blanks and numbers. Labels cannot contain new-line characters.

The **graph** command stores all points internally and drops those for which there is not room. It also drops segments that run out of bounds. The **graph** command produces a legend indicating grid range with a grid unless you specify the **-s** flag. If a specified lower limit exceeds the upper limit, **graph** reverses the axis. Note that logarithmic axes cannot be reversed.

Flags

-a [*num* [*lolim*]]	Supplies abscissas missing from the input automatically. *num* determines the spacing on the axis (the default is 1). *lolim* determines the starting point for automatic abscissas (the default is 0 or the lower limit given by **-x**[*lolim*].
-b	Breaks the graph after each label in the input.
-c *char*	Uses the character string *char* as the default label for each point.
-g *grid*	Uses *grid* as the grid style, where *grid* = 0 indicates no grid, *grid* = 1 indicates a frame with tick marks, and *grid* = 2 indicates a full grid (default).
-h *space*	Uses *space* as a fraction of space for height.
-l *"label"*	Uses *label* as a label for the graph.
-m *style*	Uses *style* as the style of connecting lines, where *style*=0 indicates disconnected lines, and *style*=1 indicates connected lines (default).
-r *space*	Uses *space* as the fraction of space to move to the right before plotting.
-s	Saves the current graphic screen image, does not erase before starting the plot.
-t	Transposes horizontal and vertical axes. (-*x* now applies to the vertical axis).
-u *space*	Uses *space* as the fraction of space to move up before plotting.
-w *space*	Uses *space* as a fraction of space for width.

-x [l] [*lolim* [*uplim* [*space*]]]
Makes the x axis logarithmic if **l** is used. Use *lolim* as the lower x axis limit and *uplim* as the upper x axis limit. Use *space* for the grid spacing on *x* axis. Normally these are determined automatically.

-y [l] [*lolim* [*uplim* [*space*]]]
Acts the same as **-x** for the y axis.

Related Information

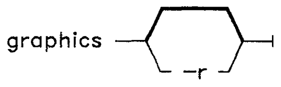
The following commands: "**spline**" on page 972 and "**tplot**" on page 1079.

graphics

Purpose

Accesses graphical and numerical commands.

Syntax



OL777038

Description

The **graphics** command appends the path name **/usr/bin/graf** to the current **$PATH** value, changes the primary shell prompt to ^, and executes a new shell. The directory **/usr/bin/graf** contains all of the graphics subsystem commands.

The command line format for a command in **graphics** is *command name* followed by *argument*(s). An *argument* may be a *filename* or an *flag string*. A *filename* is the name of any AIX Operating System file, except those beginning with -. The *filename* - is the name for the standard input. An *flag string* consists of - followed by one or more *flag*(s). A *flag* consists of a key letter possibly followed by a value. Flags may be separated by commas.

The graphical commands have been partitioned into four groups.

- Commands that manipulate and plot numerical data; see "**stat**" on page 984.
- Commands that generate tables of contents; see "**toc**" on page 1074.
- Commands that interact with graphical devices; see "**gdev**" on page 460 and "**ged**" on page 463.
- A collection of graphical utility commands; see "**gutil**" on page 508.

To produce a list of **graphics** commands, enter **whatis** in the **graphics** environment.

Flag

-r	Creates access to the graphical commands in a restricted environment; that is, it sets **$PATH** to **:/usr/bin/graf:/rbin:/usr/rbin** and invokes the restricted shell, **rsh**. To restore the environment that existed prior to issuing the **graphics** command, press **Ctrl-D** (END OF FILE). To log off of the graphics environment, enter **quit**.

Related Information

The following commands: "**gdev**" on page 460, "**ged**" on page 463, "**gend**" on page 475, "**gutil**" on page 508, "**stat**" on page 984, and "**toc**" on page 1074.

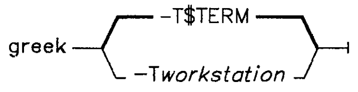
The **gps** file in *AIX Operating System Technical Reference*.

greek

Purpose

Converts output for a Teletype Model 37 work station to output for other work stations.

Syntax



OL805185

Description

The **greek** command reinterprets the Teletype Model 37 character set, including reverse and half-line motions, for display on other work stations. It simulates special characters, when possible, by overstriking. **greek** reads standard input and writes to standard output.

Flag

-T*workstation* Uses the specified *workstation*. If you omit the **-T** flag, **greek** attempts to use the work station specified in the environment variable **$TERM** (see the **environ** special facility in *AIX Operating System Technical Reference*.) *workstation* can be any one of the following:

300	DASI 300.
300-12	DASI 300 in 12-pitch.
300s	DASI 300s.
300s-12	DASI 300s in 12-pitch.
450	DASI 450.
450-12	DASI 450 in 12-pitch.
1620	Diablo 1620 (alias DASI 450).
1620-12	Diablo 1620 (alias DASI 450) in 12-pitch.
2621	Hewlett-Packard 2621, 2640, and 2645.
2640	Hewlett-Packard 2621, 2640, and 2645.
2645	Hewlett-Packard 2621, 2640, and 2645.
4014	Tektronix 4014.
hp	Hewlett-Packard 2621, 2640, and 2645.
tek	Tektronix 4014.

Files

/usr/bin/300
/usr/bin/300s
/usr/bin/4014
/usr/bin/450
/usr/bin/hp

Related Information

The following commands: "**300**" on page 1262, "**4014**" on page 1264, "**450**" on page 1265, "**eqn, neqn, checkeq**" on page 395, "**hp**" on page 514, "**mm, checkmm**" on page 663, "**tplot**" on page 1079, and "**nroff, troff**" on page 709.

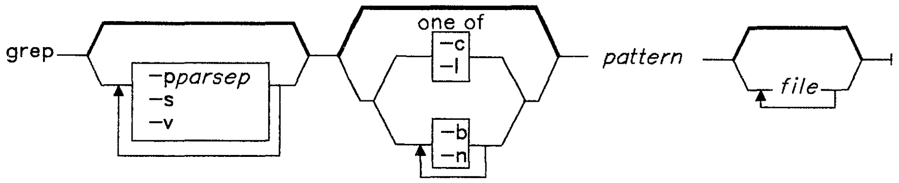
The **greek** miscellaneous facility in *AIX Operating System Technical Reference.*

grep

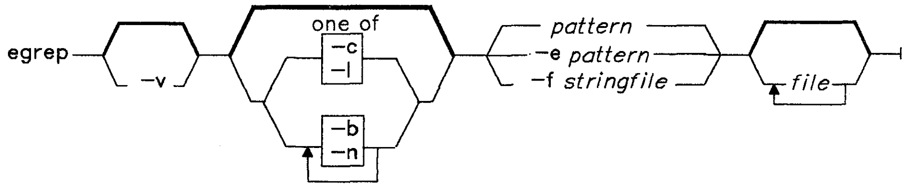
Purpose

Searches a file for a pattern.

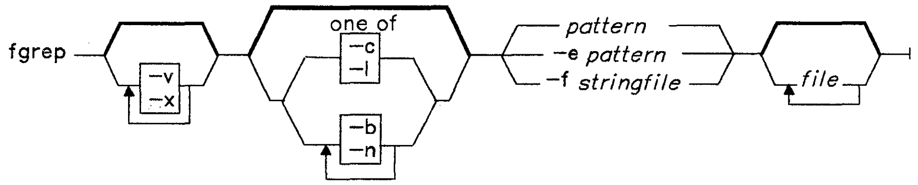
Syntax



OL805375



OL805359



OL805361

Description

Commands of the **grep** family search input *file*s (standard input by default), for lines matching a pattern. Normally, they copy each line found to standard output. Three versions of the **grep** command permit you to express the matching pattern in varying levels of complexity:

grep Searches for *pattern*s, which are limited regular expressions in the style of the **ed** command. **grep** uses a compact nondeterministic algorithm.

egrep Searches for *patterns* which are full regular expressions as in **ed**, except for \(and \) and with the addition of the following rules:

- A regular expression followed by a plus sign (+) matches one or more occurrences of the regular expression.
- A regular expression followed by a question mark (?) matches 0 or 1 occurrences of the regular expression.
- Two regular expressions separated by a vertical bar (|) or by a new-line character match strings that are matched by either.
- A regular expression may be enclosed in parentheses () for grouping.

The order of precedence of operators is [], then * ? +, then concatenation, then | and the new-line character.

The **egrep** command uses a deterministic algorithm that needs exponential space.

fgrep Searches for *patterns* that are fixed strings. It searches for lines that contain one of the strings (lines are separated by new-line characters).

All versions of **grep** display the name of the file containing the matched line if you specify more than one *file* name. Characters with special meaning to the shell ($ * [| ^ ( ) \), must be quoted when they appear in *patterns*. When *pattern* is not a simple string, you usually must enclose the entire *pattern* in single quotation marks. In an expression such as **[a-z]**, the minus means "through" according to the current collating sequence.

A collating sequence may define ***equivalence classes*** for use in character ranges. See "Overview of International Character Support" in *Managing the AIX Operating System* for more information on collating sequences and equivalence classes.

Japanese Language Support Information

A collating sequence in Japanese Language Support does not define equivalence classes for for use in character ranges. To avoid unpredictable results when using a range expression to match a class of characters, use a ***character class expression*** rather than a standard range expression. For information about character class expressions, see the discussion in "**ed**" on page 371.

The exit value of these commands is:

0 A match was found.

1 No match was found.

2 A syntax error was found or a file was inaccessible (even if matches were found).

Notes:

1. Lines are limited to 512 characters; longer lines are broken into multiple lines of 512 or fewer characters (**grep** only).
2. Paragraphs (under the **-p** flag) are currently limited to a length of 5000 characters.
3. Running **grep** on a special file produces unpredictable results and is discouraged.

Flags

-b	Precedes each line by the block number on which it was found. Use this flag to help find disk block numbers by context.
-c	Displays only a count of matching lines.
-e *pattern*	Specifies a *pattern*. This works the same as a simple *pattern* but is useful when the *pattern* begins with a - (does not work with **grep**).
-f *string file*	Specifies a file that contains patterns (**egrep**) or strings (**fgrep**).
-l	Lists just the names of files (once) with matching lines. Each file name is separated by a new-line character.
-n	Precedes each line with its relative line number in the file.
-p*parsep*	Displays the entire paragraph containing matched lines. Paragraphs are delimited by paragraph separators, *parsep*, which are patterns in the same form as the search pattern. Lines containing the paragraph separators are used only as separators; they are never included in the output. The default paragraph separator is a blank line (**grep** only).
-s	Suppresses error messages about inaccessible files (**grep** only).
-v	Displays all lines except those that match the specified pattern.
-x	Displays lines that match the pattern exactly with no additional characters (**fgrep** only).

Examples

1. To search several files for a simple string of characters:

   ```
   fgrep "strcpy" *.c
   ```

   e

   This searches for the string `strcpy` in all files in the current directory with names ending in `.c`

2. To count the number of lines that match a pattern:

   ```
   fgrep  -c  "{"  pgm.c
   fgrep  -c  "}"  pgm.c
   ```

   This displays the number of lines in `pgm.c` that contain open and close braces.

   If you do not put more than one { or } on a line in your C programs, and if the braces are properly balanced, then the two numbers displayed will be the same. If the numbers are not the same, then you can display the lines that contain braces in the order that they occur in the file with: `egrep  "{|}"  pgm.c`

3. To use a pattern that contains some of the pattern-matching characters *, ^, ?, [,], \(, \), \{, and \}:

   ```
   grep  "^[a-zA-Z]"  pgm.s
   ```

   This displays all lines in `pgm.s` that begin with a letter.

   Note that because **fgrep** does not interpret pattern-matching characters:

   ```
   fgrep  "^[a-zA-Z]"  pgm.s
   ```

   makes **fgrep** search only for the string `^[a-zA-Z]` in `pgm.s`.

4. To use an extended pattern that contains some of the pattern-matching characters +, ?, |, (, and), ::

   ```
   egrep  "\(  *([a-zA-Z]*|[0-9]*)  *\)"  my.txt
   ```

   This displays lines that contain letters in parentheses or digits in parentheses, but not parenthesized letter-digit combinations. It matches `(y)` and `(   783902)`, but not `(alpha19c)`.

   Note:

   When using **egrep**, \(and \) match parentheses in the text, but (and) are special characters that group parts of the pattern. The reverse is true for **grep**.

Japanese Language Support Information

When Japanese Language Support is installed on your system, use the following format:

```
egrep  "\( *([[:alpha:]]*|[[:digit:]]*) *\)"  my.txt
```

5. To display all lines that do *not* match a pattern:

   ```
   grep  -v  "^#"
   ```

   This displays all lines that do not begin with a # character.

6. To display the names of files that contain a pattern:

   ```
   fgrep  -l  "strcpy"  *.c
   ```

   This searches the files in the current directory that end with `.c` and displays the names of those files that contain the string `strcpy`.

Related Information

The following commands: "**ed**" on page 371, "**sed**" on page 887, and "**sh**" on page 913.

"Overview of International Character Support" in *Managing the AIX Operating System.*

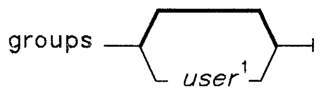
The discussion of Japanese Language Support in *Japanese Language Support User's Guide.*

groups

Purpose

Displays your group membership.

Syntax



[1]The default *user* is the person running the command.

OL805129

Description

The **groups** command writes to standard output the groups to which you or the specified *user* belong. The AIX Operating System allows you to belong to many different groups at the same time.

Your ***primary group*** is specified in the file **/etc/passwd**. Once you are logged in, you can change your active group with the **newgrp** command (see page 689). When you create a file, its group ID is that of your active group.

Other groups that you belong to are specified in the file **/etc/group**. If you belong to more than one group, you can access files belonging to any of those groups without changing your primary group ID. These are called your ***concurrent groups***.

Note: The **/etc/passwd** and **/etc/opasswd** files must be on the same node.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Files

/etc/group	Group file; contains group IDs.
/etc/ogroup	Previous version of the group file.
/etc/passwd	Password file; contains user IDs.
/etc/opasswd	Previous version of the password file.

Related Information

The following command: "**newgrp**" on page 689.

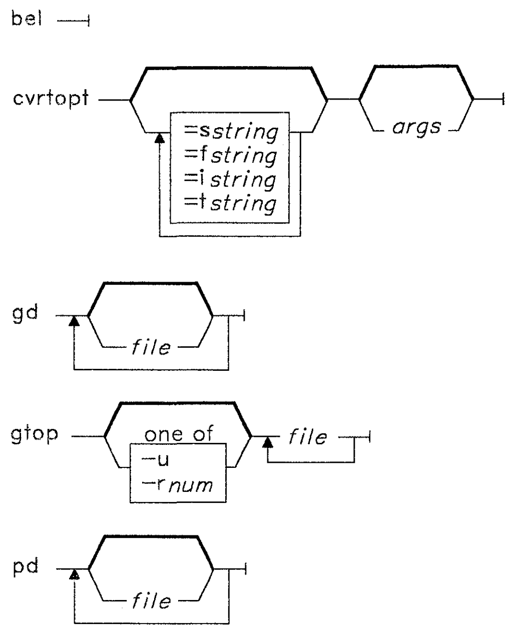
The **setgroups** system call and the **initgroups** subroutine in *AIX Operating System Technical Reference.*

gutil

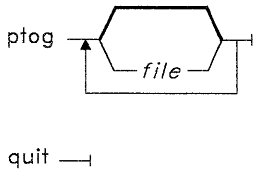
Purpose

Provides graphical utility programs.

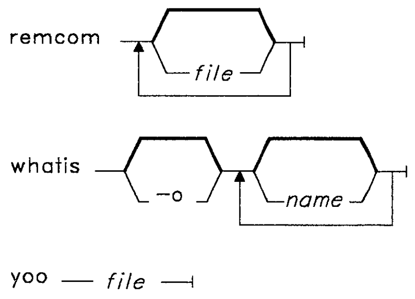
Syntax



OL777039



OL805449



OL805450

Description

The following are the miscellaneous device-independent utility commands found in the **/usr/bin/graf** directory. If you do not specify any *files*, these commands read standard input. All output is sent to standard output. Graphical data is stored in **GPS** format; see the **gps** file in *AIX Operating System Technical Reference.*

bel

Sends the ASCII BEL character to the terminal.

cvrtopt

The **cvrtopt** command reformats its arguments (usually the command line arguments of a calling shell procedure) to facilitate processing by shell procedures. An *arg* is either a file name or a flag string. A file name is either a - (minus) by itself or a string not beginning with a -. A flag string is a string of flags beginning with a - (minus). **cvrtopt** produces output of the following form:

-flag -flag . . . file . . .

All flags appear singularly in the output and precede any file names. Arguments that take values or are two letters long must be described through flags to **cvrtopt**.

The **cvrtopt** command is usually used with the **set** command as the first line of a shell procedure (see page 933 for a description of the **set** command):

```
set  -  `cvrtopt  [=flags] . . .   $@`
```

Flags

s*string*	The specified *string* accepts string values, where *string* is a one or two letter flag name.

f*string*	The specified *string* accepts floating point numbers as values, where *string* is a one- or two-letter flag name.
i*string*	The specified *string* accepts integers as values, where *string* is a one- or two-letter flag name.
t*string*	The specified *string* is a two-letter flag name that takes no value.

gd

The **gd** command produces a readable listing of a file in **GPS** format.

gtop

The **gtop** command transforms a **GPS** format into **plot** file commands displayable by **plot** filters. **GPS** objects are translated if they fall within the window that circumscribes the first *file*, unless you specify one of the following flags:

Flags

r*num*	Translates objects in **GPS** region *num*.
u	Translates all objects in the **GPS** universe.

pd

The **pd** command displays a readable listing of **plot** format graphical commands.

ptog

The **ptog** command transforms **plot** file commands into a **GPS** file.

quit

The **quit** command terminates the session.

remcom

The **remcom** command copies its input to its output with comments removed. Comments are as defined in the C language (/* comment */).

whatis

The **whatis** command displays a short description of each *name* specified. If you do not specify a *name*, then **whatis** displays the current list of description *names*. The command `whatis \*` displays every description.

Flag

-o Displays only command flags.

yoo

The **yoo** command is a piping primitive that deposits the output of a pipeline into a *file* used in the pipeline. Note that without **yoo**, this is not usually successful because it causes a read and write on the same file simultaneously.

Related Information

The following command: "**graphics**" on page 497.

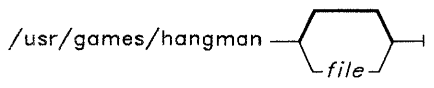
The **gps** format in *AIX Operating System Technical Reference.*

hangman

Purpose

Plays hangman, the word-guessing game.

Syntax



OL805228

Description

The **hangman** game chooses a word of at least seven letters from a standard dictionary. You try to guess the word by guessing the letters in it, one at a time. You are allowed seven mistakes. The *file* parameter specifies an alternate dictionary.

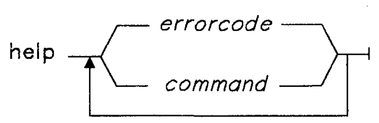
To quit the game, press INTERRUPT (**Alt-Pause**) or END OF FILE (**Ctrl-D**).

help

Purpose

Provides information about a Source Code Control System (SCCS) message or command or about certain non-SCCS commands.

Syntax



OL805054

Description

The **help** command writes to standard output information about the use of a specified SCCS *command* or about messages generated while using the commands. Each message has an associated *errorcode*, which can be supplied as a argument to the **help** command. Zero or more arguments may be supplied. If you do not supply a argument, **help** prompts for one. You may include any of the SCCS commands as arguments to **help**.

The *errorcode* consists of numbers and letters, and is found at the end of the message. For example, in the message `no id keywords (ge6)`, the error code is `ge6`.

Files

/usr/lib/help	Directory containing files of message text.
/usr/lib/help/helploc	File containing locations of help files not in **/usr/lib/help**.

Related Information

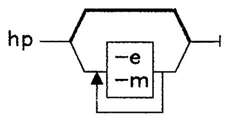
The discussion of SCCS in *AIX Operating System Programming Tools and Interfaces.*

hp

Purpose

Handles special functions for the HP2640- and HP2621-series terminals.

Syntax



OL805018

Description

The **hp** command reads standard input (usually output from **nroff**), and writes to standard output, which is usually Hewlett-Packard 2640-and 2621-series terminal displays.

If your terminal has the display enhancement feature, you can display subscripts and superscripts. With the mathematical-symbol feature, you can display Greek and other special characters the same way as the **300** command, with two exceptions: **hp** approximates the logical operator NOT with a right arrow and it only shows the top half of the integral sign.

For overstrike characters (characters followed by a backspace and another character), if either character is an underscore character, the other appears underlined or in inverse video depending on terminal enhancements.

Note: Some sequences of control characters (reverse line-feeds and backspaces) can make text disappear from the display. Tables with vertical lines generated by the **tbl** command will often be missing lines of text containing the bottom of a vertical line. You can avoid these problems by first piping the input through **col**, and then through **hp**.

Flags

-e Shows overstruck characters underlined, superscripts in half-bright, and subscripts half-bright underlined. Otherwise, all overstruck characters, subscripts, and superscripts appear in inverse video (dark-on-light). Use this flag only if your display has the display enhancements feature.

-m Produces only one blank line for any number of successive blank lines in the text.

Related Information

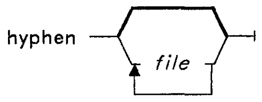
The following commands: "**300**" on page 1262, "**col**" on page 179, "**eqn**, **neqn**, **checkeq**" on page 395, "**greek**" on page 499, "**nroff, troff**" on page 709, and "**tbl**" on page 1053.

hyphen

Purpose

Finds hyphenated words.

Syntax



OL805019

Description

The **hyphen** command reads the input *file*s (standard input by default), finds all the lines ending with hyphenated words, and writes those words to standard output. A word is considered hyphenated only if the hyphen occurs at the end of a line. **hyphen** reads standard input if you do not specify any *file* names on the command line.

Note: The **hyphen** command cannot handle hyphenated italic words. It also sometimes gives unnecessary output.

Examples

1. To check the way words are hyphenated in a text file:

   ```
   hyphen  chap1
   ```

   This lists the words in `chap1` that are hyphenated at the end of a line.

2. To check the hyphenation performed by a text formatting program:

   ```
   mm  chap1  |  hyphen
   ```

   This lists the words that **nroff** decides to hyphenate across lines.

Related Information

The following commands: "**mm, checkmm**" on page 663, "**nroff, troff**" on page 709, and "**troff**" on page 710.

id

Purpose

Displays the system identity of the user issuing the command.

Syntax

```
id —┤
```

OL805131

Description

The **id** command writes a message to standard output containing the user and group IDs and corresponding names of the invoking process. When effective and real names and IDs do not match, **id** writes both.

Related Information

The following command: "**logname**" on page 589.

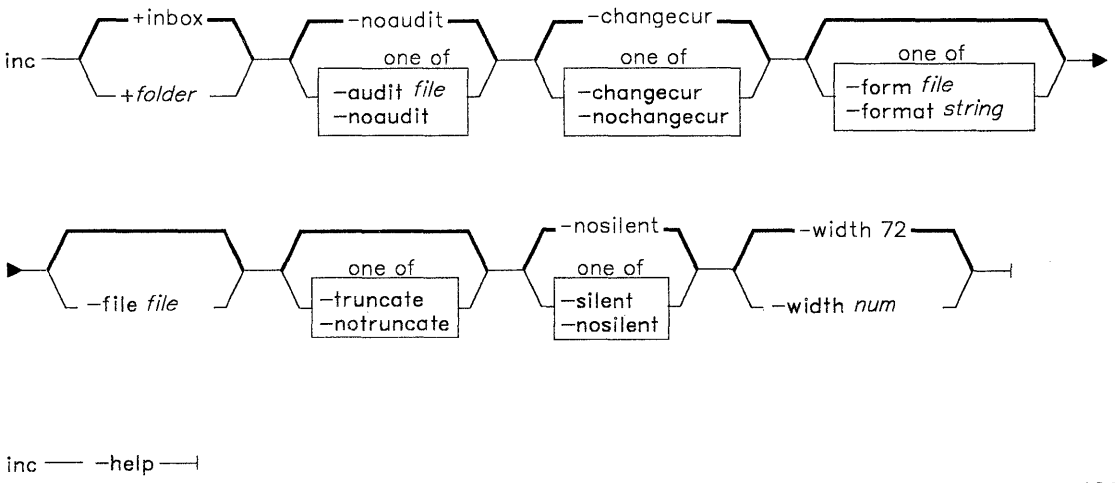
The **getuid** subroutine in *AIX Operating System Technical Reference.*

inc

Purpose

Incorporates new mail.

Syntax



AJ2FL156

Description

The **inc** command is used to incorporate incoming mail. **inc** is part of the MH (Message Handling) package and can be used with other MH and AIX commands.

The **inc** command takes all of the new messages from your mail drop and places them in the specified folder. If the specified folder does not exist, **inc** asks you if it should be created. **inc** assigns the new messages consecutive message numbers starting with the next highest number in the folder. **inc** assigns the new messages the protection code specified in the **Msg-Protect:** entry in your **.mh_profile**. If there is no **Msg-Protect:** entry, **inc** assigns the messages the protection code of 644. **inc** calls the **scan** command to display information about the new messages. If the **Unseen-Sequence:** profile entry specifies any sequences, **inc** adds the new messages to each sequence.

Flags

-audit *file*	Copies the current date to the specified file and appends the output from the **scan** command to the file.
-changecur	Sets the first new message as the current message for the specified folder. This flag is the default.
-file *file*	Incorporates messages from the specified file instead of the user's maildrop.
+*folder*	Incorporates the new messages into the specified folder. **+inbox** is the default folder.
-form *file*	Displays the **scan** command output in the alternate format described in *file.*
-format *string*	Displays the **scan** command output in the alternate format described by *string.*
-help	Displays help information for the command.
-noaudit	Does not record information about incorporating the new messages (see the **-audit** flag). This flag is the default.
-nochangecur	Does not alter the setting for the current message in the specified folder.
-nosilent	Prompts the user for any necessary information. This flag is the default.
-notruncate	Does not clear the mailbox or file from which **inc** is taking new messages. If **-file** is specified, **-notruncate** is the default.
-silent	Does not prompt you for any information. This flag is useful for running **inc** in the background.
-truncate	Clears the mailbox or file from which **inc** is taking new messages. If **-file** is not specified, **-truncate** is the default.
-width *num*	Sets the number of columns in the **scan** command output. The default is the width of the display.

Profile Entries

Alternate-Mailboxes:	Specifies your mailboxes.
Folder-Protect:	Sets the protection level for your new folder directories.
Msg-Protect:	Sets the protection level for your new message files.
Path:	Specifies your *user_mh_directory.*
Unseen-Sequence:	Specifies the sequences used to keep track of your unseen messages.

Files

$HOME/.mh_profile	The MH user profile.
/usr/lib/mh/mtstailor	The MH tailor file.
/usr/mail/$USER	The location of the mail drop.

Related Information

Other MH commands: "**mhmail**" on page 646, "**post**" on page 758, "**scan**" on page 871.

The **mh-format**, **mh-mail**, and **mh-profile** files in *AIX Operating System Technical Reference.*

"Overview of the Message Handling Package" in *Managing the AIX Operating System.*

init

Purpose

Initializes the system.

Syntax

```
init —┤ 1
```

1This command should not be entered on the command line.

OL805132

Description

After the kernel completes the basic processor initialization, it starts the **init** process, which is the ancestor of all other processes in the system. The **init** command controls the mode in which the system is running; this usually is either maintenance mode or multiuser mode. A user with superuser authority can alter **init** so that the system runs in controlled access mode. It is the program from which all loggers and most system daemons start.

When **init** starts, it determines what the startup mode should be, based on information in the **/etc/.init.state** file. If this file does not exist or is unreadable, **init** bases the startup mode on information passed to it by the kernel. The following are the usual startup modes for **init**:

maintenance	Starts a shell on the console, but does not start any other processes (single-user mode).
multiuser	Runs the command file **/etc/rc** and spawns loggers on all enabled ports.
exec-program	Runs the specified program.

Maintenance Mode

Use maintenance mode for the following:

- System installation
- Correcting problems on the file system using the **fsck** command
- All operations requiring an inactive system.

There are three ways to bring the system up in maintenance mode:

1. If the system is currently running in normal (multiuser) mode, use the **shutdown -m** command to bring the system down to maintenance mode (**shutdown** sends **init** a **SIGINT** signal).
2. Start the system from the Installation/Maintenance Diskette and specify the `Maintenance Mode` option from the End System Management menu.
3. Edit the **/etc/.init.state** file so that it consists of the character **m**. This causes the system to come up in maintenance mode each time it is started up.

Maintenance mode starts a shell program with superuser authority on the console. When you log off this shell by pressing END OF FILE (**Ctrl-D**), **init** asks you if you want to leave maintenance mode. A response beginning with `n` or `N` indicates "no," and **init** starts another shell on the console. Any processes running in the background continue to run. Any other response indicates "yes."

If the response is yes, **init** enters normal mode, as described in the following section. It also asks if the file system integrity should be checked. A response beginning with `n` or `N` indicates "no." Any other response indicates "yes." Your answer determines whether the **rc** command runs with an **m** or **d** argument.

Normal Mode

After the normal startup of the system (either from system startup or by leaving maintenance mode), **init** runs the normal initialization command file, **/etc/rc**. It passes **rc** an argument of either **m** (normal startup, clean root), or **d** (normal startup, dirty root). The latter is the default argument if the startup is from maintenance mode. **rc** is responsible for performing integrity checks, doing any necessary cleanups, mounting the normal file systems, enabling standard ports, and starting system daemons. If an error occurs during the running of this command file (indicated by a nonzero return code), **init** either forces a system restart by executing the **reboot** system call or enters maintenance mode.

Once **rc** completes successfully, **init** starts logger processes (normally **getty**) on each enabled port. Whenever someone ends a logger by logging off a port, **init** notes the logoff and starts a new logger on the port. Everything **init** knows about enabling ports is contained in the **/etc/portstatus** file, which is maintained by the **penable** command. Through this file, you can enable new ports or disable ports that were previously enabled. Whenever **init** receives a **SIGHUP** (hangup) signal, it rereads the **portstatus** file to see if any changes of port status have been requested.

The **init** commands then reads the commands in the **/etc/rc.ds** file, if that file exists. Typically, **/etc/rc.ds** contains commands to start Distributed Services. Any commands that are needed to run remote mounts should be placed in **/etc/rc.ds**.

If, at any time after the system starts up normally, **init** discovers that no ports are enabled or if **init** receives an **INTERRUPT** signal, it decides again on startup options. Generally, this means **init** will go through normal startup, assuming a dirty root.

Environments

Because **init** is the ultimate ancestor of every process on the system, its environment parameters are inherited by every process. As part of its initialization sequence, **init** reads the file **/etc/environment** and copies any assignments found in that file into the environment passed to all of its subprocesses. It treats **umask** differently. If it is assigned a reasonable octal value, **init** does a **umask** system call for the specified value, rather than passing the value in the environment. Similarly, if **filesize** is specified, **init** issues a **ulimit** call with the given size as the argument.

Files

/etc/utmp	Record of logged-in users
/usr/adm/wtmp	Permanent login accounting file
/etc/portstatus	Enabled port status file
/etc/rc	Initialization command file
/etc/environment	System environment variables

Related Information

The following commands: "**getty**" on page 490, "**pstart, penable, pshare, pdelay**" on page 791, "**rc**" on page 806, and "**shutdown**" on page 946.

The **reboot** and **umask** system calls and the **portstatus** file in *AIX Operating System Technical Reference.*

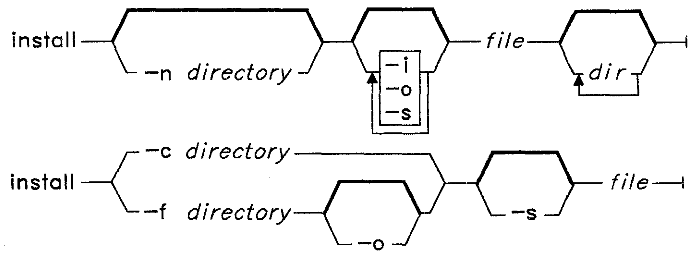
The discussion of the trusted path in *Managing the AIX Operating System* and *Using the AIX Operating System.*

install

Purpose

Installs a command.

Syntax



OL805022

Description

The **install** command installs *file* in a specific place within a file system. It is most often used in makefiles (see "**make**" on page 625). When replacing files, **install** copies each file into the appropriate directory, thereby retaining the original owner and permissions. A newly-created *file* has permission code 755, owner **bin**, and group **bin**. **install** writes a message telling you exactly which files it is replacing or creating and where they are going.

If you do not supply any arguments, **install** searches a set of default directories (**/bin**, **/usr/bin**, **/etc**, **/lib**, and **/usr/lib**, in that order) for a file with the same name as *file*. The first time it finds one, it overwrites it with *file* and issues a message indicating that it has done so. If a match is not found, **install** issues a message telling you there was no match and exits with no further action.

If any directories are specified on the command line, **install** searches them before it searches the default directories.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Flags

-c *directory*	Installs a new command file in *directory* only if that file does not already exist there. If it finds a copy of *file* there, it issues a message and exits without overwriting the file. This flag can be used alone or with **-s**.
-f *directory*	Forces installation of *file* in *directory* whether or not *file* already exists. If the file being installed does not already exist, **install** sets the permission code and owner of the new file to **755** and **bin**, respectively. This flag can be used alone or with **-o** or **-s**.
-i	Ignores the default directory list and searches only those directories specified on the command line. This flag cannot be used with **-c** or **-f**.
-n *directory*	Installs *file* in *directory* if it is not in any of the searched directories and sets the permissions and owner of the file to **755** and **bin**, respectively. This flag cannot be used with **-c** or **-f**.
-o	Saves the old copy of *file* by copying it to **OLD***file* in the directory in which it found it. This flag cannot be used with **-c**.
-s	Suppresses display of all but error messages.

Examples

1. To replace a command that already exists in one of the default directories:

   ```
   install  fixit
   ```

   This replaces `fixit` if it is found in **/bin**, **/usr/bin**, **/etc**, **/lib**, or **/usr/lib**. Otherwise, it is not installed. For example, if `/usr/bin/fixit` exists, then this file is replaced by a copy of the file `fixit` in the current directory.

2. To replace a command that already exists in a specified or default directory, and to preserve the old version:

   ```
   install  -o  fixit  /etc  /usr/games
   ```

   This replaces `fixit` if found in **/etc**, **/usr/games**, or one of the default directories. Otherwise it is not installed. If `fixit` is replaced, the old version is preserved by renaming it `OLDfixit` in the directory in which it was found (**-o**).

3. To replace a command that already exists in a specified directory:

   ```
   install  -i  fixit  /u/jim/bin  /u/joan/bin  /usr/games
   ```

   This replaces `fixit` if found in `/u/jim/bin`, `/u/joan/bin`, or **/usr/games**. Otherwise it is not installed.

4. To replace a command if found in a default directory, or install it in a specified directory if not found:

   ```
   install  -n  /usr/bin  fixit
   ```

   This replaces `fixit` if found in one of the default directories. If `fixit` is not found, it is installed as `/usr/bin/fixit` (**-n /usr/bin**).

5. To install a new command:

   ```
   install  -c  /usr/bin  fixit
   ```

   This creates a new command by installing a copy of `fixit` as `/usr/bin/fixit`, but only if this file does not already exist.

6. To install a command in a specified directory whether or not it already exists:

   ```
   install  -f  /usr/bin  -o  -s  fixit
   ```

   This forces `fixit` to be installed as `/usr/bin/fixit` whether or not it already exists. The old version, if any, is preserved by moving it to `/usr/bin/`**OLD**`fixit` (**-o**). The messages that tell where the new command was installed are suppressed (**-s**).

Related Information

The following command: "**make**" on page 625.

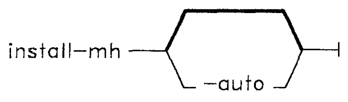
The **mk** system maintenance procedure in *AIX Operating System Technical Reference.*

install-mh

Purpose

Initializes the MH environment.

Syntax



AJ2FL230

Description

The **install-mh** command is used to set up mailbox directories. **install-mh** is not designed to be run directly by the user; it is designed to be called by other programs. The **install-mh** command is part of the MH (Message Handling) package.

The **install-mh** command is issued automatically the first time you run any MH command. **install-mh** prompts you for the name of your mail directory. If the directory does not exist, **install-mh** asks you if it should be created. **install-mh** creates the file **$HOME/.mh_profile** and places the **Path:** profile entry in it. This entry identifies the location of your mailbox.

Flag

-auto	Creates the standard MH path without prompting.

Profile Entry

Path:	Specifies your *user_mh_directory*. This entry is created by **install-mh**.

Files

$HOME/.mh_profile	The MH user profile.

Related Information

The **mh-profile** file in *AIX Operating System Technical Reference.*

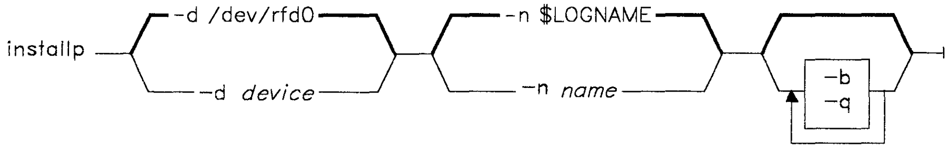
"Overview of the Message Handling Package" in *Managing the AIX Operating System.*

installp

Purpose

Installs a licensed program.

Syntax



OL805021

Description

Warning: Before you install a program, restart your system and make sure that no other users are on the system and no other programs are running.

The **installp** command installs a program. You must be a member of the system group or operating with superuser authority to run this command.

When auditing is on, an audit record of the type, **installp** is created. If your system is running in controlled access mode, this command should only be run under the **watch** command.

Because more than one program may be on a set of diskettes, **installp** asks whether or not you want to install each program. If you do, **installp** checks to see if it is an older version than the one currently installed. If the version to be installed is older than the version on the system, **installp** informs you and asks if you want to continue.

When **installp** is instructed to perform the installation of a program subset for code service clients on a system where the complete program is already installed, it checks to see if the program subset is compatible with the existing complete program. If the program subset and complete program are incompatible, **installp** warns you and then prompts you for a response.

The **installp** command makes a backup copy of the program history file before installation begins. If installation is unsuccessful, it sets the Version, Release, and Level fields of the last record of the history file to 00.00.0000 and logs the exit value in the program history file. The history file remains on the system as **/usr/lpp/***pgm-name***/lpp.hist**, where *pgm-name* is the program name.

Pressing INTERRUPT (**Alt-Pause**) will not stop the **installp** command. To stop **installp**, press QUIT WITH DUMP (**Ctrl-V**). Use this only in extreme circumstances since the system state is unpredictable and one of the following can occur:

- The write-verify feature may be left on for all minidisks
- All terminals other than the console may be disabled
- Some install control files may need to be deleted.

Notes:

1. Only ordinary files with the prefix **lpp.** remain in **/usr/lpp/***pgm-name* after completion of **installp**. All other ordinary files are removed.
2. Installing a complete program on a client where a program subset already exists causes the history file information for the program subset to be destroyed.

Flags

-b	Runs the **bffcreate** command to create an installation file in backup format from the distribution media. Then tells **installp** to use the file as the distribution media for the install. A server in a code service environment uses this flag to install.
-d *device*	Installs the program from the specified *device*. The default *device* is **/dev/rfd0**.
-n *name*	Logs the first eight nonblank characters of *name* in the program history file. The default *name* is the value of the environment variable **$LOGNAME**.
-q	Runs in quiet mode suppressing most of the interactive queries.

The **installp** command runs a program-provided installation procedure **instal**. Each installation procedure returns one of the following exit values to **installp**:

0	Installation completed. Take no action.
2	Update superblocks, i-nodes, and delayed block I/O (sync), then restart the AIX Operating System.
3	Build the kernel, then update the superblocks, i-nodes, and delayed block I/O (sync) and shut down the AIX Operating System.
4	Build the kernel, then update the superblocks, i-nodes, and delayed block I/O (sync) and restart the kernel.
5	Installation cancelled without errors.
6	Update superblocks, i-nodes, and delayed block I/O (sync), then shut down the AIX Operating System.

Any other return value indicates that installation failed.

Internal Commands

Install procedures can use the internal install commands. These commands provide common code for the save and recovery functions frequently needed by most program-provided procedures. They do a minimum validation of input parameters and return exit values like subcommands. You can, however, receive messages from system commands that they call. C Language procedures that call these commands can use the **/usr/include/inu21.h** file to define return codes.

inusave

The **inusave** command saves some or all of the files and archive files that will change during a program install or update procedure. It uses the following syntax:

> **inusave** *listfile pgm-name*

The *pgm-name* parameter specifies the program to be installed or updated. *pgm-name* can be a maximum of eight characters. *listfile*, which must be a full path name, contains a list of relative path names (relative to the root) for all of the files that need to be saved. *listfile* must be in the format of an ***apply list***. (See *AIX Operating System Programming Tools and Interfaces* for a discussion of the format of an apply list.)

The **inusave** command creates the ***save directory*** (**/usr/lpp/***pgm-name***/inst_updt.save**). This is the directory in which the install and update procedures store saved files and the control list that correlates the local file names with their full path names. **inusave** uses *listfile* as a basis to determine which files need to be temporarily saved.

If the file named in *listfile* already exists, **inusave** copies that file to **/usr/lpp/***pgm-name***/inst_updt.save/update.***n*, where *n* is an integer assigned by **inusave**. If the file does not exist, **inusave** assumes that this entry in *listfile* represents either a new file or a file to be archived or processed by the archive procedure. **inusave** maintains a list of saved files in **/usr/lpp/***pgm-name***/inst_updt.save/update.list**. The format of each entry in the list is:

> **update.***n*   *file*

where **update.***n* is the name of the saved file and *file* is the full path name of the file.

An archived constituent file is saved if there is a valid archive control file, **/usr/lpp/***pgm-name***/lpp.acf**, for the program. If this file exists, **inusave** compares each of the file names in *listfile* to the constituent file names in **/usr/lpp/***pgm-name***/lpp.acf**. When it finds a match, **inusave** uses the **ar** command to extract the constituent file from its associated archive file.

It then moves it to **/usr/lpp/***pgm-name***/inst_updt.save/archive.***n*, where *n* is an integer selected by **inusave**. **inusave** maintains a list of the extracted files that have been saved in the file **/usr/lpp/***pgm-name***/inst_updt.save/archive.list**. The format of each entry in the list is:

archive.*n* *cfile* *afile*

where **archive.***n* is the name of the saved file and *cfile* and *afile* are the constituent and archive files defined in the archive control file.

The **inusave** command returns the following exit values:

0	No error conditions occurred.
105	Failure occurred trying to create a save directory.
107	Copy of a file from one directory to another failed. This implies that the update apply has not yet begun and that the old level of the program is still usable.
202	One or more parameters missing.
204	Too many parameters were entered.
207	Could not access the apply list.

inurecv

The **inurecv** command recovers all files and archive-constituent files saved from the previous **inusave**. **inurecv** uses the following syntax:

inurecv *pgm-name* *reject-flag*

It uses the control lists from the **/usr/lpp/***pgm-name***/inst_updt.save** directory to recover the files. **inusave** creates the **/usr/lpp/***pgm-name***/inst_updt.save** directory and control lists. **inurecv** also recovers files that may have been saved by the program-provided install or update procedure (see *AIX Operating System Programming Tools and Interfaces* for details).

The **inurecv** command has to distinguish between an immediate recovery that occurs because of an error condition during an install or update and an update rejection that occurs because a user rejects an update (`updatep -r`). If the *reject-flag* argument is **yes**, **inurecv** assumes that it is being run because of an update rejection. If the argument is **no** or if no flag is specified, **inurecv** assumes that it is being run because of an immediate recovery.

The **inurecv** command returns the following exit status values:

0	No error conditions occurred.
101	The save directory does not exist.
102	A copy of a file from one directory to another failed. This implies that the program could not be recovered and that it must be reinstalled and any updates reapplied.
104	A file that was saved in the save directory was not found.
205	Replacement of a constituent file in an archive file failed while attempting to recover a program. This implies that the program is no longer useable and should be reinstalled and any updates reapplied.

inurest

The **inurest** command does simple restores and archives. It does not do any additional processing or user interaction. **inurest** uses the following syntax:

inurest -**d***device*] [-**q**] *listfile pgm-name*

The *listfile* is the full path name of a file containing the relative directory target path name (relative to the root), of files that a program needs to restore. It must be in the format of an apply list. **inurest** restores all files in the list relative to the root directory. *pgm-name* specifies the name of the program to be installed or updated. It can be a maximum of eight characters.

To archive a file, there must be an archive control file, **/usr/lpp/***pgm-name***/lpp.acf**. If it exists, **inurest** compares each of the target names in *listfile* to the component files listed in there. Whenever **inurest** finds a match, it archives the restored file into the corresponding archive file and deletes the restored file.

Flags

The following flags modify the action of **inurest**:

-d *device*	Specifies the input *device*. The default device is **/dev/rfd0**.
-q	Prohibits **restore** from displaying the `insert volume 1` prompt.

The **inurest** command returns the following exit status values:

0	No error conditions occurred.
106	Failed trying to restore an updated version of files.
201	An invalid flag was specified.
202	One or more parameters missing.
204	Too many parameters were entered.
206	Failed trying to replace file in an archive file.
208	Could not access the apply list.

ckprereq

The **ckprereq** command determines whether the system level is compatible with the program to be installed or updated. It uses the following syntax:

ckprereq [-**v**] [-**f** *prerequisites*]

You can run **ckprereq** only if you are a member of the system group or are operating with superuser authority. *prerequisites* is a program prerequisite list file. Each record in this file contains the name of a prerequisite program and describes the version, release, and level requirements. There is one record for each prerequisite program. The default *prerequisites* file is **prereq**. See *AIX Operating System Programming Tools and Interfaces* for details on the format of **ckprereq** file entries.

The **ckprereq** command tests the current version, release, and level found in the history file and marks each **prereq state** field of the **prereq** file with one of the following codes if the test fails:

l	The test is false for level.
f	The history file format is not fixed 80.
n	The history file was not found.
r	The test is false for release.
s	There is a syntax error in the **prereq** file.
u	The history file is in an unknown state.
v	The test is false for version.

A blank **prereq state** field indicates that the test was true. The exit value of **ckprereq** is the number of records that did not test true. If all records test true, the exit value is 0.

Note: If a program is installed on a local node and executed on a remote node, the remote node must have file trees that have all necessary prerequisite files available. When auditing is on, an audit record of the type, **ckprereq** is created.

Flags

-f *prerequisites*	Specifies the *prerequisites* file to use in place of **prereq**.
-v	Sends a descriptive message to standard error for each failure in the prerequisite program test. The messages give the same information as the **prereq state** field of the **prereq** file.

mvmd

The **mvmd** command updates the VRM minidisk. It uses the following syntax:

mvmd -a *file* -D *VRM-dir* [-fp [*file*]] -l *pgm-name*
mvmd -c *VRM-file permissions* -l *pgm-name*
mvmd -d *VRM-file* -l *pgm-name*
mvmd -m *VRM-file* [-fp [*file*]] -l *pgm-name*
mvmd -r *file* -D *VRM-dir* -l *pgm-name*

You must be a member of the system group or operating with superuser authority to run **mvmd**. When auditing is on, an audit record of the type, **mvmd** is created.

Flags

-a *file*	Adds the specified *file* to the VRM minidisk. Use the **-D** flag to specify the destination VRM directory. *file* must not already exist in the specified directory. By default, **mvmd** adds the file to the first unused position in the VRM directory. To specify a position, use the **-f** or **-p** flag.

-c *VRM-file permissions*
Changes the permission code of the specified *VRM-file* to the octal value, *permissions*. The *VRM-file* parameter must be a full path name. Valid combinations of permission bits are as follows:

0700	The loadlist processor loads, runs, and deletes this module.
0450	The loadlist processor transfers control to this module after all loadlist directory entries have been processed.
0440	The loadlist processor loads this module.
0410	This module is a virtual machine.
0040	If the system startup device is a diskette, the loadlist processor is to load the module. If the system startup device is a fixed disk, the loadlist processor does not load the module. Instead, it maps the module.

The loadlist processor ignores any module that does not have the load bit set. For more information about these permission bits, see *Virtual Resource Manager Technical Reference.*

-d *VRM-file* Deletes the specified file from the VRM minidisk. The *VRM-file* parameter must be a full path name.

-D *VRM-dir* Specifies the full path name of the VRM directory.

-f [*file*] Specifies the position following *file* in the directory list or, if you do not specify *file*, the bottom of the directory list. Use this positioning flag with the **-a** or **-m** flags.

-l *pgm-name* Specifies the name of a program that is modifying the VRM minidisk. The *pgm-name*, the date, the user name, and a descriptive title are place in a record appended to the VRM history file. If you do not specify this flag, then a record with the name `UNKNOWN` is appended to the VRM history file.

-m *VRM-file* Moves the specified file within its VRM directory. By default, **mvmd** moves the file to the first unused position. To specify a position, use the **-f** or **-p** flag.

-p [*file*] Specifies the position prior to *file* in the directory list or, if you do not specify *file*, the top of the directory list. Use this positioning flag with the **-a** or **-m** flags.

-r *file* Replaces the specified *file* on the VRM minidisk. Use the **-D** flag to select the VRM directory of the file to be replaced. Both the replacement file and the file to be replaced must have the same name.

The **mvmd** command returns an exit status of 0 if no errors occurred. A nonzero return indicates that an error occurred.

Files

/usr/include/inu21.h	Defines error codes for internal commands
/usr/lpp/*pgm-name*/instal	Program installation procedure
/usr/lpp/*pgm-name*/inst_updt.save	Directory for saved files
/usr/lpp/*pgm-name*/inst_updt/inu*PID*temp*n*	Temporary files
/usr/lpp/*pgm-name*/liblpp.a	Central archive file
/usr/lpp/*pgm-name*/lpp.acf	Archive control file
/usr/lpp/*pgm-name*/lpp.hist	Program history file
/usr/lpp/*pgm-name*/prereq	Program prerequisite list file
/usr/lpp/*pgm-name*/inst_updt.save/archive*n*	File containing saved, archived, constituent files
usr/lpp/*pgm-name*/inst_updt.save/archive.list	List of the saved, extracted, archived files

Related Information

The following commands: "**updatep**" on page 1122 and "**bffcreate**" on page 108.

The discussion of code service in *Managing the AIX Operating System.*

The **fork** and **exec** system calls and the **lpp.hist** file in *AIX Operating System Technical Reference.*

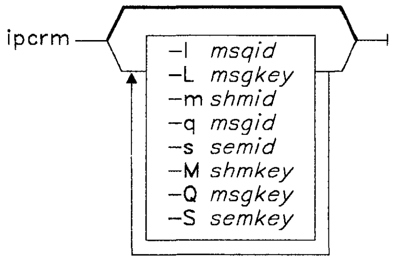
The discussion of installing programs in *AIX Operating System Programming Tools and Interfaces.*

ipcrm

Purpose

Removes message queue, semaphore set or shared memory identifiers.

Syntax



OL805135

Description

The **ipcrm** command removes one or more message queue, semaphore set, or shared memory identifiers.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Flags

-l*msqid*	Removes local information about the remote queue *msqid* without removing the remote queue.
-L*msgkey*	Removes local information about the remote queue *msgkey* without removing the remote queue.
-m *shmid*	Removes the shared memory identifier *shmid*. The shared memory segment and data structure associated with *shmid* are also removed after the last detach.

-M *shmkey*	Removes the shared memory identifier, created with key *shmkey*. The shared memory segment and data structure associated with it are also removed after the last detach.
-q *msqid*	Removes the message queue identifier *msqid* and the message queue and data structure associated with it.
-Q *msgkey*	Removes the message queue identifier, created with key *msgkey*, and the message queue and data structure associated with it.
-s *semid*	Removes the semaphore identifier *semid* and the set of semaphores and data structure associated with it.
-S *semkey*	Removes the semaphore identifier, created with key *semkey*, and the set of semaphores and data structure associated with it.

The details of the remove operations are described in **msgctl**, **shmctl**, and **semctl** in the *AIX Operating System Technical Reference.* The identifiers and keys can be found by using the **ipcs** command.

Related Information

The following command: "**ipcs**" on page 539.

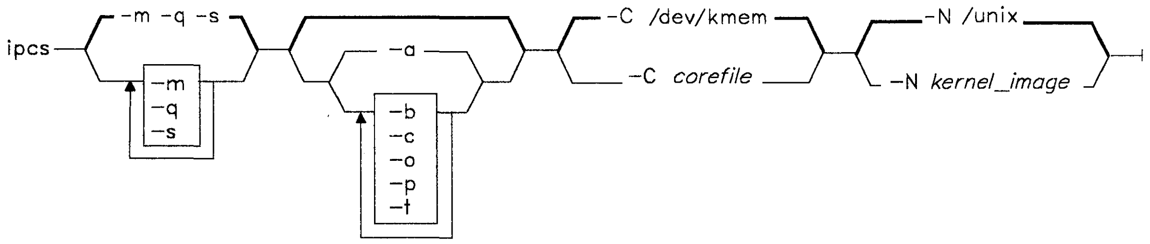
The **msgctl**, **msgget**, **msgrcv**, **msgsnd**, **semctl**, **semget**, **semop**, **shmctl**, **shmget**, and **shmop** system calls in *AIX Operating System Technical Reference.*

ipcs

Purpose

Reports interprocess communication facility status.

Syntax



OL805432

Description

The **ipcs** command 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, semaphores, remote queues, and local queue headers.

The column headings and the meaning of the columns in an **ipcs** listing follow. The letters in parentheses indicate the flags that cause the corresponding heading to appear. **all** means that the heading always appears. These flags only determine what information is provided for each facility. They do not determine which facilities will be listed.

T (**all**) Type of facility:

q	message queue
Q	message queue resides on a remote node
m	shared memory segment
s	semaphore.

ID (**all**) The identifier for the facility entry.

KEY (**all**) The key used as a parameter to **msgget**, **semget**, or **shemget** to make the facility entry.

Note: 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.

MODE	**(all)** The facility access modes and flags. The mode consists of 11 characters that are interpreted as follows:

The first two characters can be:

R	if a process is waiting on a **msgrcv**
S	if a process is waiting on a **msgsnd**
D	if the associated shared memory segment has been removed. It disappears when the last process attached to the segment detaches it.
C	if the associated shared memory segment is to be cleared when the first attach is run
-	if the corresponding special flag is not set.

The next 9 characters are interpreted as three sets of 3 bits 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 of the owner of the facility entry.
GROUP	**(all)** The name of the group that owns the facility entry.
CREATOR	**(a,c)** The login name of the creator of the facility entry.
CGROUP	**(a,c)** The group name of the group of the creator of the facility entry.

Note: For the **OWNER**, **GROUP**, **CREATOR**, and **CGROUP**, the user and group IDs display instead of the login names.

CBYTES	**(a,o)** The number of bytes in messages currently outstanding on the associated message queue.
QNUM	**(a,o)** The number of messages currently outstanding on the associated message queue.
QBYTES	**(a,b)** The maximum number of bytes allowed in messages outstanding on the associated message queue.
LSPID	**(a,p)** The ID of the last process that sent a message to the associated queue. If the last message sent was from a process in a node other than the node which holds the queue, then **LSPID** is the PID of the kernel process which actually placed the message on the queue, not the PID of the sending process.

LRPID	(**a**,**p**) The ID of the last process that received a message from the associated queue. If the last message received was from a process in a node other than the node which holds the queue, then **LRPID** is the PID of the kernel process which actually received the message on the queue, not the PID of the receiving process.
STIME	(**a**,**t**) The time when the last message was sent to the associated queue. For remote queues, this is the server time. No attempt is made to compensate for time-zone differences between the local clock and the server clock.
RTIME	(**a**,**t**) The time when the last message was received from the associated queue. For remote queues, this is the server time. No attempt is made to compensate for any clock skew between the local clock and the server clock.
CTIME	(**a**,**t**) The time when the associated entry was created or changed. For remote queues, this is the server time. No attempt is made to compensate for any clock skew between the local clock and the server clock.
NATTCH	(**a**,**o**) The number of processes attached to the associated shared memory segment.
SEGSZ	(**a**,**b**) The size of the associated shared memory segment.
CPID	(**a**,**p**) The process ID of the creator of the shared memory entry.
LPID	(**a**,**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.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Flags

-a	Uses the **-b**, **-c**, **-o**, **-p** 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.
-C*corefile*	Uses the file *corefile* in place of **/dev/kmem**. *corefile* is a memory image file produced by the **Ctrl-**(left)**Alt-End** key sequence.
-m	Writes information about active shared memory segments.
-N*kernel-image*	Uses the specified *kernel-image* (**/unix** is the default).
-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 last **msgrcv** on message queues • Time of the last **shmat** and last **shmdt** on shared memory • Time of the last **semop** on semaphore sets.

Files

/unix	System kernel image
/dev/kmem	Memory
/etc/passwd	User names
/etc/group	Group names

Related Information

The **ipcs**, **msgrcv**, **msgsnd**, **semop**, **shmat**, and **shmdt** system calls in *AIX Operating System Technical Reference.*

The discussion of generating core files in *Problem Determination Guide.*

ipctable

Purpose

Creates, displays, or changes the Distributed Services IPC Queues Table.

Syntax

```
ipctable —|
```

OL805468

Description

The **ipctable** command lets you build, examine, or modify the Distributed Services IPC Queues Table. Only members of the system group or users operating with superuser authority can use **ipctable** to change the state of the Distributed Services IPC Queues Tables (see "**su**" on page 1026). Other user can use **ipctable** to browse the IPC Queues Table.

Related Information

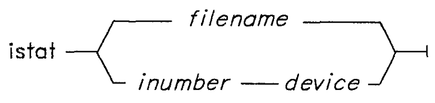
"Getting Started With Distributed Services Configuration Menus" in *Managing the AIX Operating System.*

istat

Purpose

Examines i-nodes.

Syntax



OL805138

Description

The **istat** command writes information about the i-nodes specified with *inumber* to standard output. Use the **istat** command to write information about the i-node for a specified *filename*, or to write the contents of a specified i-node, *inumber* on an arbitrary file system.

If you specify *filename*, **istat** writes the following information about the file:

- The device where the file resides.
- The i-node number of the file, on that device.
- The file type (normal, directory, block device, and so on).
- What protection is on the file.
- The name and identification number of the owner and group.

  Note: The owner and group names for remote files are taken from the local **/etc/passwd** file.
- The number of links to the file.
- If the i-node is for a normal file, the length of the file.
- If i-node is for a device, the major and minor device designations.
- The date of the last time the i-node was updated.
- The date of the last time the file was modified.
- The date of the last time the file was referenced.

If you specify *inumber* and *device*, **istat** also displays, in long decimal values, the block numbers recorded in the i-node. You can specify the *device* as either a device name or as a mounted-file-system name.

Note: *inumber* and *device* cannot specify a remote device.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Examples

1. To display the information stored in a file i-node:

   ```
   istat  /bin/sh
   ```

   This displays the i-node information for the file /bin/sh. The information looks something like this:

   ```
   Inode  34  on  device  0/10  File
   Protection:  rwxr-xr-x   Sticky
   Owner:  0(su)           Group: 0(system)
   Link count:   1          Length 54240 bytes

   Last updated:   Tue Dec 18 01:07:36 1984
   Last modified:  Sat Jun 30 18:11:47 1984
   Last accessed:  Wed Feb 13 11:06:37 1985
   ```

2. To display i-node information if given a file i-number:

   ```
   istat  34  /dev/hd0
   ```

   This displays the information contained in i-node number 34 on the /dev/hd0 device. In addition to the information shown in Example 1, this displays:

   ```
   Block pointers:
        219   220   221   222   223   224   225   226
        227   228   229    0     0
   ```

   These numbers are addresses of the disk blocks that contain the data in the file.

Related Information

The following command: "**fsdb**" on page 450.

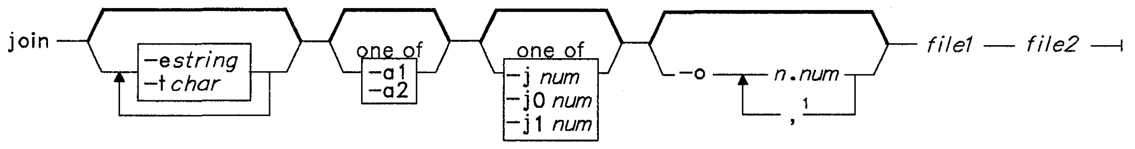
The **stat** system call and the **filesystems** and **fs** files in *AIX Operating System Technical Reference.*

join

Purpose

Joins data fields of two files.

Syntax



¹Do not put a blank on either side of the comma.

OL805371

Description

The **join** command reads *file1* and *file2*, joins lines in the files according to the flags, and writes the results to standard output. Both files must be sorted according to the collating sequence specified by the **NLCTAB** environment variable, if set, for the fields on which they are to be joined (normally the first field in each line).

One line appears in the output for each identical ***join field*** appearing in both *file1* and *file2*. The join field is the field in the input files that **join** looks at to determine what will be included in the output. The output line consists of the join field, the rest of the line from *file1*, then the rest of the line from *file2*. You can specify standard input in place of *file1* by substituting a - (minus) for the name.

Both input files must be sorted in increasing ASCII collating sequence on the fields on which they are to be joined (the join field, normally the first field in each line).

Fields are normally separated by a blank, a tab character, or a new-line character. In this case, **join** treats consecutive separators as one, and discards leading separators.

Flags

-a*num*	When *num* is **1**, **join** produces an output line for each line found in *file1* but not in *file2*. When *num* is **2**, **join** produces an output line for each line found in the *file2* but not in *file1*.
-e *string*	Replaces empty output fields with *string*.
-j[*n*] *num*	Joins the two files on the *num*th field of file *n*. *n* is **0** or **1**. If you do not specify *n*, **join** uses the *num*th field in each file.
-o *n.num*[,*n.num* . . .]	Makes each output line consist of the fields specified in *list*, in which each element has the form *n.num*, where *n* is a file number and *num* is a field number.
-t*char*	Uses *char* as the field separator character in the input and the output. Every appearance of *char* in a line is significant. The default separator is a blank. With default field separation, the collating sequence is that of **sort -b**. If you specify **-t**, the sequence is that of a plain sort. To specify a tab character, enclose it in single quotation marks ('').

Examples

Note: The vertical alignment shown in these examples may not be consistent with your output.

1. To perform a simple join operation on two files whose first fields are the same:

```
join  phonedir  names
```

If `phonedir` contains the following telephone directory:	and `names` contains these names and departments numbers:	then **join** displays:

```
Adams     A.  555-6235
Dickerson B.  555-1842
Erwin     G.  555-1234
Jackson   J.  555-0256
Lewis     B.  555-3237
Norwood   M.  555-5341
Smartt    D.  555-1540
Wright    M.  555-1234
Xandy     G.  555-5015
```
	```
	Erwin     Dept.  389
	Frost     Dept.  217
	Nicholson Dept.  311
	Norwood   Dept.  454
	Wright    Dept.  520
	Xandy     Dept.  999
	```
		```
		Erwin   G. 555-1234 Dept. 389
		Norwood M. 555-5341 Dept. 454
		Wright  M. 555-1234 Dept. 520
		Xandy   G. 555-5015 Dept. 999
		```

Each line consists of the join field (the last name), followed by the rest of the line found in `phonedir` and the rest of the line in `names`.

2. To display unmatched lines with the command:

```
join  -a2  phonedir  names
```

If phonedir contains:	and names contains:	then **join** displays:
Adams A. 555-6235	Erwin Dept. 389	Erwin G. 555-1234 Dept. 389
Dickerson B. 555-1842	Frost Dept. 217	Frost Dept. 217
Erwin G. 555-1234	Nicholson Dept. 311	Nicholson Dept. 311
Jackson J. 555-0256	Norwood Dept. 454	Norwood M. 555-5341 Dept. 454
Lewis B. 555-3237	Wright Dept. 520	Wright M. 555-1234 Dept. 520
Norwood M. 555-5341	Xandy Dept. 999	Xandy G. 555-5015 Dept. 999
Smartt D. 555-1540		
Wright M. 555-1234		
Xandy G. 555-5015		

This performs the same join operation as in Example 1, and also lists the lines of names that have no match in phonedir. Frost and Nicholson are included in the listing, since they do not have entries in phonedir.

3. To display selected fields:

```
join  -o 2.3 2.1 1.2 1.3  phonedir names
```

This displays the following fields in the order given:

Field 3 of names	(Department Number)
Field 1 of names	(Last Name)
Field 2 of phonedir	(First Initial)
Field 3 of phonedir	(Telephone Number)

If phonedir contains:	and names contains:	then **join** displays:
Adams A. 555-6235	Erwin Dept. 389	389 Erwin G. 555-1234
Dickerson B. 555-1842	Frost Dept. 217	454 Norwood M. 555-5341
Erwin G. 555-1234	Nicholson Dept. 311	520 Wright M. 555-1234
Jackson J. 555-0256	Norwood Dept. 454	999 Xandy G. 555-5015
Lewis B. 555-3237	Wright Dept. 520	
Norwood M. 555-5341	Xandy Dept. 999	
Smartt D. 555-1540		
Wright M. 555-1234		
Xandy G. 555-5015		

4. To perform the join operation on a field other than the first:

```
sort  +2 -3  phonedir ¦ join -jl 3  - numbers
```

This combines the lines in phonedir and numbers, comparing the third field of phonedir to the first field of numbers.

First, this sorts phonedir by the third field, because both files must be sorted by their join fields. The output of **sort** is then piped to **join**. The - (minus sign) by itself causes the **join** command to use this output as its first file. The -jl 3 defines the third field of the sorted phonedir as the join field. This is compared to the first field of numbers because its join field is not specified with a **-j** flag.

If numbers contains:	then this command displays the names listed in phonedir for each telephone number:
555-0256 555-1234 555-5555 555-7358	555-0256 Jackson J. 555-1234 Erwin G. 555-1234 Wright M.

Note that **join** lists all the matches for a given field. In this case, **join** lists both Erwin G. and Wright M. as having the telephone number 555-1234. The number 555-5555 is not listed because it does not appear in phonedir.

Related Information

The following commands: "**awk**" on page 81, "**comm**" on page 183, "**sort**" on page 958, "**cut**" on page 269, and "**paste**" on page 736.

The **environment** miscellaneous facility in *AIX Operating System Technical Reference.*

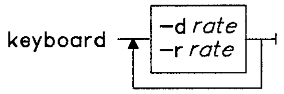
"Overview of International Character Support" in *Managing the AIX Operating System.*

keyboard

Purpose

Controls the delay and repetition rates of the keyboard.

Syntax



OL805443

Description

The **keyboard** command changes the keyboard delay and repetition rates. These rates are initially set at system startup to 500 milliseconds and 14 characters per second, respectively.

Flags

-d*rate*	Sets the delay rate to the specified value. *rate* can be **300**, **400**, **500**, or **600** milliseconds.
-r*rate*	Sets the rate of repetition to the specified value. This *rate* can be an integer from **2** to **40**, inclusive.

Example

To change both the delay and repetition rates:

```
keyboard -d300 -r40
```

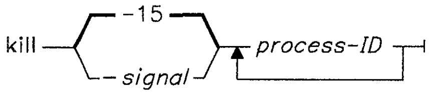
This sets the keyboard to a delay of 300 milliseconds and the repetition rate to 40 characters per second.

kill

Purpose

Sends a signal to a running process.

Syntax



OL805139

Description

The **kill** command sends a *signal* to a running process, by default signal 15 (**SOFTWARE TERMINATE**). This default action normally kills processes that do not catch or ignore the signal. You specify a process by giving its *process-ID* (process identification number, or ***PID***). The shell reports the PID of each process that is running in the background (unless you start more than one process in a pipeline, in which case the shell reports the number of the last process). You can also use the **ps** command to find the process ID number of commands.

In addition, there are special *process-IDs* that cause the following special actions:

0	The *signal* is sent to all processes having a process-group ID equal to the process-group ID of the sender (except those with PID's 0 and 1).
-1	If the effective user ID of the sender is not 0 (root), *signal* is sent to all processes with a process-group ID equal to the effective user ID of the sender. (except those with PID's 0 and 1). If the effective user ID of the sender is 0 (root), *signal* is sent to all processes, excluding numbers 0 and 1.
-process-ID	The *signal* is sent to all processes whose process-group number is equal to the absolute value of *process-ID*. Note that when you specify a minus PID, you must also specify the *signal* to be sent, even signal 15.

See the **kill** system call in *AIX Operating System Technical Reference* for a complete discussion of **kill**. For a list of signal numbers, see the **signal** systems call in *AIX Operating System Technical Reference.*

Unless you are operating with superuser authority, the process you wish to stop must belong to you. When operating with superuser authority, you can stop any process.

Examples

1. To stop a given process:

   ```
   kill  1095
   ```

   This stops process 1095 by sending it the default signal, 15 (also called **SIGTERM**). Note that process 1095 might not actually stop if it has made special arrangements to ignore or override signal 15.

2. To stop several processes that ignore the default signal:

   ```
   kill  -9  1034  1095
   ```

   This sends signal 9 (**SIGKILL**) to processes 1034 and 1095. Signal 9 is a special signal that normally cannot be ignored or overridden.

3. To stop all of your background processes:

   ```
   kill  0
   ```

   This sends signal 15 to all members of the shell process group. This includes all background processes started with &. (See page 914 about running background processes.) Although the signal is sent to the shell, it has no effect because the shell ignores signal 15.

4. To stop all of your processes and log yourself off:

   ```
   kill  -9  0
   ```

   This sends signal 9 to all members of the shell process group. Because the shell cannot ignore signal 9, this also stops the login shell and logs you off. If you are using multiple windows on a high-function terminal, then this closes the active window.

5. To stop all processes that you own:

   ```
   kill  -9  -1
   ```

   This sends signal 9 to all processes owned by the effective user, even those started at other work stations and that belong to other process groups. If you are using multiple windows on a high-function terminal, then this closes all of the windows. If a listing that you requested is being printed, then it is also stopped.

   Note: To send signal 15 with this form of the **kill** command, you must specify -15 explicitly:

   ```
   kill  -15  -1
   ```

6. To send a different signal code to a process:

   ```
   kill  -16  1103
   ```

   This sends signal 16 (**SIGUSR1**) to process 1103.

   The name of the **kill** command is misleading because many signals, including 16, do not stop processes. The action taken on signal 16 is defined by the particular application you are running.

Related Information

The following commands: "**csh**" on page 225, "**ps**" on page 786, and "**sh**" on page 913.

Note: The **csh** command contains a built-in subcommand named **kill**. The command and subcommand do not necessarily work the same way. For information on the subcommand, see the **csh** command.

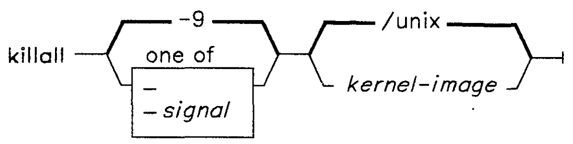
The **kill** and **signal** system calls in *AIX Operating System Technical Reference.*

killall

Purpose

Cancels all processes except the calling process.

Syntax



OL805140

Description

The **killall** command cancels all processes that you started, except those producing the **killall** process. This command provides a convenient means of canceling all processes created by the shell that you control. When started by a user operating with superuser authority, **killall** cancels all cancelable processes except those that started it.

The *kernel-image* parameter specifies the name of the system load module (by default, **/unix**).

Flags

-	Sends a **SIGTERM** signal initially and then sends a **SIGKILL** (kill) signal to all processes that survive for 30 seconds after receipt of the signal first sent. This gives processes that catch **SIGTERM** signal an opportunity to clean up. (For more information, see the **signal** system call in *AIX Operating System Technical Reference.*)
-*signal*	Sends the specified *signal* number. (For information about signal numbers, see the **signal** system call in *AIX Operating System Technical Reference.*)

Examples

1. To stop all background processes that have started:

   ```
   killall
   ```

   This sends all background processes the kill signal 9 (also called **SIGKILL**).

2. To stop all background processes, giving them a chance to clean up:

   ```
   killall -
   ```

   This sends signal 15 (**SIGTERM**), waits 30 seconds, and then sends signal 9 (**SIGKILL**).

3. To send a specific signal to the background processes:

   ```
   killall -2
   ```

   This sends signal 2 (**SIGINT**) to the background processes.

Files

/unix	System kernel image.
/dev/mem	Used for reading the process table.

Related Information

The following command: "**kill**" on page 552.

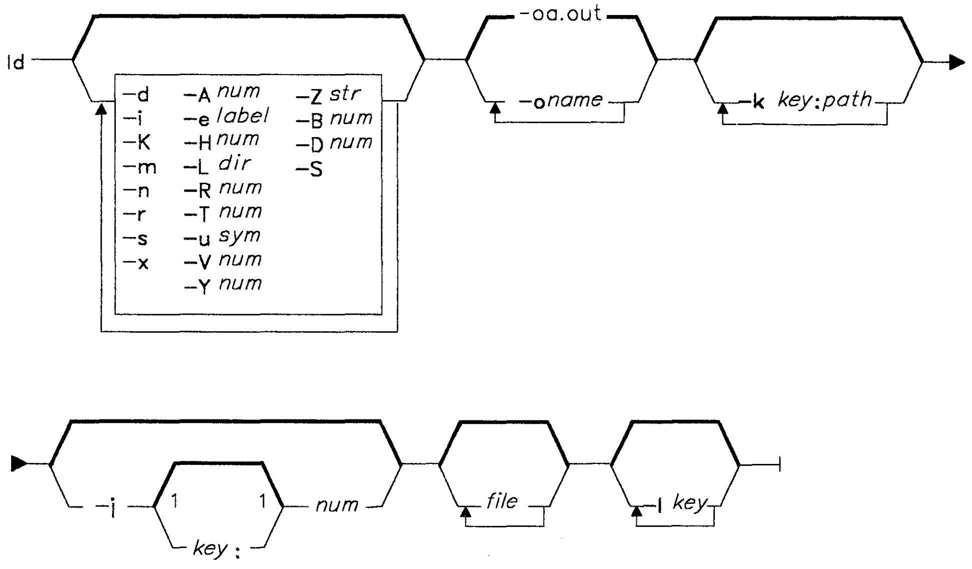
The **signal** system call in *AIX Operating System Technical Reference.*

ld

Purpose

Links object files.

Syntax



OL805362

1 Do not put a blank between these items.

OL805308

Description

The **ld** command (the ***linkage editor***) combines the specified object *file*s into one file, resolving external references and searching libraries. It produces an object module that can be run or that can become a *file* parameter in another call to **ld**. In the latter case, you must use the **-r** flag to preserve the relocation bits. **ld** places its output in a file named **a.out**. It makes this file executable if no errors occur during the link and if the **-r** flag is not specified.

The linkage editor links object files and searches object libraries in the order specified. It links object modules unconditionally, but links from the library only those files that define an unresolved external reference. If a routine from a library calls another routine in that library, the called routine must follow the calling routine.

Unless you use the **-e** flag to specify another entry point, the first byte of the first nonnull text segment (or the first byte of the data segment if all text segments are null) becomes the entry point of the output file.

The reserved symbols **_text**, **_data**, **_sdata**, **_etext**, **_edata**, and **_end** (in C, **text**, **data**, **sdata**, **etext**, **edata**, and **end**) are set to the first location of the program, the first location of the data, the segment number of the data, the first location above the program, the first location above initialized data, and the first location above all data, respectively. You cannot define these symbols.

Because you can use **ld** to link modules intended to run on other machines, some of its action depends upon the architecture of the computer system on which you intend to run the module. **ld** recognizes that architecture automatically from the input modules and modifies its action accordingly. You can use some of its flags to alter the default behavior of **ld** for a particular architecture.

Flags

The **ld** command recognizes several flags. Except for **-l** entries, which are really abbreviations for file names, the order in which you specify flags does not affect the way they work. You can specify numeric values in either decimal, octal (with a leading **0**), or hexadecimal (with a leading **0x** or **0X**) format.

-A*num*	Stores *num* in the **a_misc** field of the output file header. This field indicates the size of memory, in bytes, allocated to the process which runs the file. On many systems, the stand-alone loader or kernel uses this value to set the base of the run-time stack pointer.
-B*num*	Makes *num* the starting address for the uninitialized data (bss) segment of the output file. The default starting address is the first storage unit after the end of the data segment. Not all architectures support the separation of data and bss segments.
-d	Defines common storage, even if you have specified the **-r** flag.
-D*num*	Makes *num* the starting address for the initialized data segment of the output file. The default starting address begins at location 0 (if **-i** is in effect), at the first storage unit after the end of the text segment, or, if **-n** is in effect, at the next page or segment boundary.
-e*label*	Makes *label* the entry point of the executable output file.
-H*num*	Makes *num* the boundary, usually the page size, to which the text segment must be padded if it has a different protection than does the data segment. Specify this parameter only to override the default value for the given architecture.

-i	Assigns text and data segments to separate address spaces in memory, with the text segment read-only—if the architecture supports read-only memory—and shared among all users. The data segment starts at location zero unless set with **-D**. If the architecture does not support separate instruction and data space, this flag is treated as if it were **-n**.
-j[*key*:]*num*	Assigns the shared library image *key* to location *num*. If you do not specify *key*, do not use location *num* when you assign the run-time location of the shared library text images. The exact interpretation of *num* depends on the target architecture. On the RT work station, *num* refers to the segment register, one of 4 through 13. You can specify **-j** once for each shared library image that has an assigned location.
-k*key*:*path*	Maps any reference to the shared library image with the shared library *key* into *path*. Instead of adding the shared library *key* to the run-time table, add *path*. You can specify **-k** once for each shared library image with a remapped *key*.
-K	Loads the **a.out** header into the first bytes of the text segment, followed by the text segments from the object modules. This flag causes pages of executable files to be aligned on pages in the file system so that they can be paged upon demand on systems that support paging. This flag provides mapped file support for the text and data segments.
-l*key*	Searches the specified library file, where *key* selects the file **lib***key***.a**. **ld** searches for this file in the directory specified by an **-L** flag, then in **/lib** and **/usr/lib**. It searches library files in the order that you list them on the command line.
-L*dir*	Looks in *dir* for files specified by **-l** keys. If it does not find the file in *dir*, **ld** searches the standard directories.
-m	Lists on standard output the names of all files and archive members used to create the output file.
-n	Makes the text segment read-only—if the architecture supports read-only memory—and shared among all users running the file. The data segment starts at the first segment boundary following the end of the text unless set with **-D**. On architectures which only permit read-only text with separate text and data spaces, the **-n** flag is treated as if it were the **-i** flag.
-o *name*	Assigns *name* rather than **a.out** to the output file.
-r	Writes relocation bits in the output file so that it can serve as a file parameter in another **ld** call. This flag also prevents common symbols from being assigned final definitions and suppresses the `undefined symbol` diagnostic messages.
-R*num*	Makes *num* bytes the allocation unit for objects manipulated by **ld**, such as segments or common objects. Typically this value ranges from 1 to 8. Specify this parameter only to override the default value for the given architecture.

-s	Strips the symbol table, line number information, and relocation information from the output. This saves space but impairs the usefulness of the debugger. Using the **strip** command has the same effect. This flag is turned off if there are any undefined symbols.
-S *num*	Makes *num* the maximum size the user stack is allowed to grow. This value represents the number of bytes allowed. If you do not specify this argument, the system assumes a default limit of 1 MB.
-T*num*	Makes *num* the starting address for the text segment of the output file. If not specified, the text segment begins at location zero.
-u *sym*	Enters *sym* into the symbol table as an undefined symbol. This is useful when linking from only a library, since initially the symbol table is empty and an unresolved reference is needed to force the linking of the first routine.
-V*num*	Stores *num* in the **a_version** field of the output file header; *num* must be in the range 0 to 32767.
-x	Does not enter local symbols in the output symbol table; enters only external symbols. This flag saves some space in the output file.
-Y*num*	In a segmented system, makes *num* the boundary to which the text segment should be padded if it has a protection different from that of the data segment. If *num* is zero, the padding is either that selected by the **-H** flag or the default value associated with that flag. Specify this parameter only to override the default value for the given architecture.
-Z*str*	Prefixes with *str* the names specified by the **-l** key. For example, with **-Z/test** and **-lxyz**, **ld** looks for the file **/test/lib/llbxyz.a** or, if that file does not exist, **/test/usr/lib/libxyz.a**. The ordinary directories will not be searched. This flag is most useful when cross-compiling.

Examples

1. To link several object files and produce an **a.out** file to run under the AIX Operating System without the Floating-Point Accelerator:

   ```
   ld -n -t0x10000000 -K /lib/crt0.o pgm.o subs1.o subs2.o -lrts -lc
   ```

   A simpler way to accomplish this is to use the **cc** command to link the files as follows:

   ```
   cc  pgm.o  subs1.o  subs2.o
   ```

   Since the **cc** command automatically uses the link options and necessary support libraries, you do not need to specify them on the command line (it gets this information from the configuration file **cc.cfg**). For this reason, you should use **cc** to link files when you are producing programs that run under the AIX Operating System.

2. To specify the name of the output file:

   ```
   cc  -o pgm  pgm.o  subs1.o  subs2.o
   ```

   This stores the linked output in the file pgm.

3. To conditionally link library subroutines:

   ```
   cc  pgm.o  subs1.o  subs2.o  mylib.a  -ltools
   ```

   This links the object modules pgm.o, subs1.o, and subs2.o unconditionally. It then links the subroutines from mylib.a that are used by the preceding modules. (This is often called ***conditional linking***.) Then **ld** conditionally links subroutines from the library specified by -ltools. (This means /lib/libtools.a, if it exists. If **ld** does not find this file, then it looks for /usr/lib/libtools.a.)

   Note: Always list libraries and **-l** flags at the end of the **ld** or **cc** command lines.

Files

/lib/lib*.a	Libraries.
/usr/lib/lib*.a	Libraries.
a.out	Output file.

Related Information

The following commands: "**ar**" on page 55, "**as**" on page 61, "**cc**" on page 140, and "**shlib**" on page 939.

The **a.out** file in *AIX Operating System Technical Reference.*

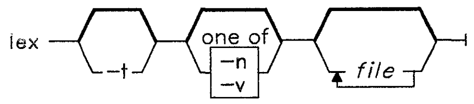
The discussion of **ld** in *AIX Operating System Programming Tools and Interfaces* and in *Assembler Language Reference.*

lex

Purpose

Generates a C Language program that matches patterns for simple lexical analysis of an input stream.

Syntax



OL805025

Description

The **lex** command reads *file* or standard input, generates a C Language program, and writes it to a file named **lex.yy.c**. This file, **lex.yy.c**, is a compilable C Language program.

The **lex** command uses *rules* and *actions* contained in *file* to generate a program, **lex.yy.c**, which can be compiled with the **cc** command. It can then receive input, break the input into the logical pieces defined by the *rules* in *file*, and run program fragments contained in the *actions* in *file*. For a more detailed discussion of **lex** and its operation, see *AIX Operating System Programming Tools and Interfaces.*

The generated program is a C Language function called **yylex**. **lex** stores **yylex** in a file named **lex.yy.c**. You can use **yylex** alone to recognize simple, one-word input, or you can use it with other C Language programs to perform more difficult input analysis functions. For example, you can use **lex** to generate a program that simplifies an input stream before sending it to a parser program generated by the **yacc** command.

The function **yylex** analyzes the input stream using a program structure called a ***finite state machine***. This structure allows the program to exist in only one state (or condition) at a time. There is a finite number of states allowed. The rules in *file* determine how the program moves from one state to another.

If you do not specify a *file*, **lex** reads standard input. It treats multiple files as a single file.

Note: Since **lex** uses fixed names for intermediate and output files, you can have only one **lex**-generated program in a given directory.

Input File Format (*file*)

The input file can contain three sections: definitions, rules, and user subroutines. Each section must be separated from the others by a line containing only the delimiter, %%. The format is:

definitions
%%
rules
%%
user subroutines

The purpose and format of each are described in the following sections.

Definitions

If you want to use variables in your rules, you must define them in this section. The variables make up the left column, and their definitions make up the right column. For example, if you want to define **D** as a numerical digit, you would write;

```
    D    [0-9]
```

You can use a defined variable in the rules section by enclosing the variable name in braces (`{D}`).

In the definitions section, you can set table sizes for the resulting finite state machine. The default sizes are large enough for small programs. You may want to set larger sizes for more complex programs.

%p *n*	Number of positions is *n* (default 2000)
%n *n*	Number of states is *n* (default 500)
%t *n*	Number of parse tree nodes is *n* (default 1000)
%a *n*	Number of transitions is *n* (default 3000)

If extended characters appear in regular expression strings, you may need to reset the output array size with the **%o** parameter (possibly to array sizes in the range 10,000 to 20,000). This reset reflects the much larger number of characters relative to the number of ASCII characters.

Rules

Once you have defined your terms, you can write the rules section. It contains strings and expressions to be matched in *file* to **yylex**, and C commands to execute when a match is made. This section is required, and it must be preceded by the delimiter **%%**, whether or not you have a definitions section. The **lex** command does not recognize your rules without this delimiter.

In this section, the left column contains the pattern to be recognized in an input file to **yylex**. The right column contains the C program fragment executed when that pattern is recognized.

Patterns can include extended characters with one exception: these characters cannot appear in range specifications within character class expressions surrounded by square brackets. The columns are separated by a tab. For example, if you want to search files for the keyword KEY, you might write:

```
(KEY)
printf("found KEY");
```

Japanese Language Support Information

Patterns can include SJIS characters, with the same exception as previously noted: the SJIS characters cannot appear in range specifications within character class expressions enclosed in square brackets.

If you include this rule in *file*, the lexical analyzer **yylex** matches the pattern KEY and runs the **printf** command.

Each pattern can have a corresponding action, that is, a C command to execute when the pattern is matched. Each statement must end with a semicolon. If you use more than one statement in an action, you must enclose all of them in braces. A second delimiter, **%%**, must follow the rules section if you have a user subroutine section.

When **yylex** matches a string in the input stream, it copies the matched file to an external character array, **yytext**, before it executes any commands in the rules section.

You can use the following operators to form patterns that you want to match:

x	Matches the character written. *x* matches the literal character x.
[ ]	Matches any one character in the enclosed range ([.-.]) or the enclosed list ([...]). **[a,b,c,x-z]** matches a,b,c,x,y,or z.
" "	Matches the enclosed character or string even if it is an operator. *"$"* prevents **lex** from interpreting the character $ as an operator.

\	Acts the same as " ". \$ also prevents the shell from interpreting the character $ as an operator.
*	Matches zero or more occurrences of the character immediately preceding it. **x*** matches zero or more repeated.
+	Matches one or more occurrences of the character immediately preceding it.
?	Matches either zero or one occurrences of the character immediately preceding it.
^	Matches the character only at the beginning of a line. `^x` matches an x at the beginning of a line.
[^]	Matches any character except the one following the ^. [`^x`] matches any character except x.

Japanese Language Support Information

The character following the ^ *cannot* be a 2-byte character.

.	Matches any character except the new-line character.
$	Matches the end of a line.
¦	Matches either of two characters. `x ¦ y` matches either x or y.
/	Matches one character only when followed by a second character. It reads only the first character into **yytext**. **x/y** matches x when it is followed by y, and reads x into **yytext**.
( )	Matches the pattern in the parentheses. This is used for grouping. It reads the whole pattern into **yytext**. A group in parentheses can be used in place of any single character in any other pattern. `(xyz123)` matches the pattern `xyz123` and reads the whole string into **yytext**.
{}	Matches the character as you defined it in the definitions section. If you defined D to be numerical digits, `{D}` matches all numerical digits.
{*m*,*n*}	Matches *m* to *n* occurrences of the character. **x{2,4}** matches 2, 3, or 4 occurrences of **x**.

If a line begins with only a blank, **lex** copies it to the output file, **lex.yy.c**. If the line is in the declarations section of *file*, **lex** copies it to the declarations section of **lex.yy.c**. If the line is in the rules section, **lex** copies it to the program code section of **lex.yy.c**.

User Subroutines

The **lex** library has three subroutines defined as macros, which you can use in the rules.

input( )	Reads a character from **yyin**.
unput( )	Replaces a character after it has been read.
output( )	Writes an output character to **yyout**.

You can override these three macros by writing your own code for these routines in the user subroutines section. But if you write your own, you must undefine these macros in the definition section as follows:

```
%{
#undef input
#undef unput
#undef output
}%
```

There is no **main( )** in **lex.yy.c** because the **lex** library contains the **main( )** that calls **yylex**. Therefore, if you do not include **main( )** in the user subroutines section, when you compile **lex.yy.c**, you must enter **cc -ll lex.yy.c**, where **ll** will call the **lex** library.

External names generated by **lex** all begin with the preface **yy**, as in **yyin**, **yyout**, **yylex**, and **yytext**.

Flags

-n	Suppresses the statistics summary. When you set your own table sizes for the finite state machine (see page 563), the **lex** automatically produces this summary if you do not select this flag.
-t	Writes **lex.yy.c** to standard output instead of to a file.
-v	Provides a one-line summary of the generated finite-state-machine statistics.

Files

/usr/lib/libl.a	Run-time library.

Related Information

The following command: "**yacc**" on page 1237.

The description of **lex** in *AIX Operating System Programming Tools and Interfaces.*

"Overview of International Character Support" in *Managing the AIX Operating System.*

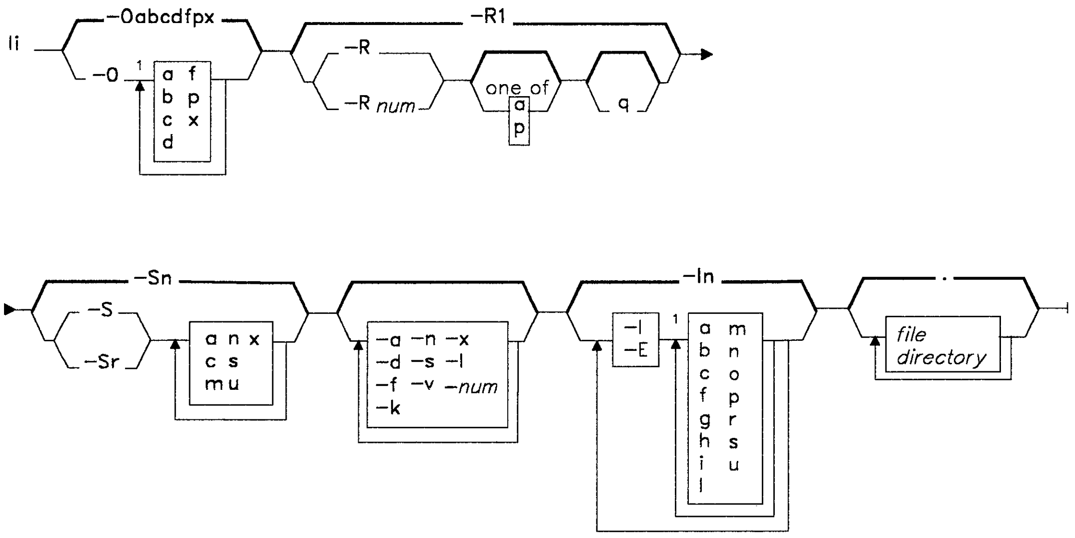
The discussion of Japanese Language Support in *Japanese Language Support User's Guide.*

li

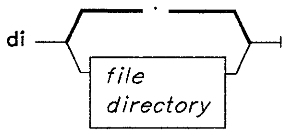
Purpose

Lists the contents of a directory.

Syntax



OL805372



OL805346

¹ Do not put a blank between these items.

OL805308

Description

The **li** command lists the contents of each named *directory* or archive *file* on standard output. For each nonarchive *file* named, **li** displays the file name and any information requested. If you do not specify a *file* or *directory*, **li** lists the current directory.

By default, **li** sorts the output alphabetically and lists it in multiple columns. The collating sequence is determined by the **NLCTAB** environment variable (see "**ctab**" on page 257). It displays control characters in file names in expanded form (for example, `^D`, `\177`). When you specify more than one file or directory, **li** sorts them appropriately, but files appear before directories and their contents. When the date and time appear, the **NLLDATE** and **NLTIME** environment variables control their format. The **NLSMONTH** environment variable controls the short names for months.

The **di** command is equivalent to `li -Ialmops`.

Permissions Field

The permissions field displayed with the **-Ip** flag contains 11 characters. The first character is:

d	The entry is a directory.
b	The entry is a block-type special file.
c	The entry is a character-type special file.
l	The entry is a symbolic link.
p	The entry is a pipe (FIFO).
-	The entry is an ordinary file.
D	The entry is a remote directory.
F	The entry is a remote ordinary file.
B	The entry is a remote block special file.
C	The entry is a remote character special file.
L	The entry is a remote symbolic link.
P	The entry is a remote first-in first-out (FIFO) special file.

The next nine characters are interpreted as three sets of 3 bits each. The first set refers to owner permissions, the next to permissions for others in the same group, and the last to all others. Each of the three characters within each set indicate, respectively, permission to read, write, or execute the file. For a directory, execute permission is interpreted as permission to search the directory for a specified file. These permissions are indicated as follows:

r	If the file is readable.
w	If the file is writable.
x	If the file is executable.
-	If the corresponding permission is not granted.

The group-execute permission is given as **s** if the file has set-group-ID mode. The user-execute permission character is given as **s** if the file has set-user-ID mode. (For a discussion of these modes, see "**chmod**" on page 160.)

The last character of the field is normally blank, but is displayed as **t** if the 1000 bit of the mode is on. (See the **chmod** system call in *AIX Operating System Technical Reference* for the current meaning of this mode.)

Note: Some combinations of flags do not work well together. For example, `li -vRa` looks unusual, and `li -RSx` and `li -Sx *` are both nearly unintelligible if there are subdirectories contained in the current directory, due to confusion about what level is being listed.

Flags

Flags are grouped into five classes, four of which are always introduced by an uppercase letter: fields (**I** or **E**), restrictions (**O**), recursion (**R**), sort orders (**S**), and miscellaneous. The following flags modify the action of **li**:

-I [hiplogcsmaunrfb]
-E [hiplogcsmaunrfb]
Requests the inclusion (**-I**) or exclusion (**-E**) of certain fields. These fields are selected by the flags in the subset **hiplogcsmaunrfb**. **-I** includes and **-E** excludes the selected fields in the order in which they appear in the argument list. For example, `-l -Ep` excludes the protections field, while `-Ep -l` includes it, since `-l` (the equivalent of `-Icglmop`) follows `-Ep`.

The only field included by default is the name (**n**) field. If you include any other fields, **li** lists the output in single-column rather than multiple-column format. **li** lists the following fields in the following order:

h	Headers
i	I-number
p	Protections
l	Link count
o	Local owner (name or UID)
g	Local group (name or GID)
c	Character count
s	Size in blocks
m	Modified time
a	Accessed time
u	Updated (i-node modified) time
n	Name
r	Node where the entry resides
f	Raw UID of the entry's owner
b	Raw GID of the entry's group.

If the file is a special file, the size (**s**) field contains the major-and minor-device numbers. If you select the **c** (character count) or **s** (size in blocks) flags, **li** writes a total number of blocks for each directory and a grand total when appropriate.

For remote files and directories, the local owner and local group are obtained by using inverse IDs. If there is no inverse ID or if **li** cannot determine the inverse ID, a **-** (minus sign) displays in the corresponding field. If possible, remote nodes are identified with nicknames. Otherwise, they are identified by their NID displayed in hexadecimal. (See "Distributed Services Concepts" in *Managing the AIX Operating System.*)

For local files and directories that do not have a nickname defined for the local node ID, the node ID field displays as a **-** (minus sign), and the raw UID (GID) field contains the local owner UID (group GID).

-L Follows a symbolic link and reports on the file at the end of the link.

-O [abcdfpx]
Requests that the listing be restricted to files of certain types. These types are selected from the subset **abcdfpx**. The possible types are:

a	Archives
b	Block devices
c	Character devices
d	Directories
f	Files (normal, not special)
p	Pipes (FIFOs)
x	Executable files (any file with execute permission)

-R[*num*]**apq**
Lists recursively to *num* levels deep. The default depth is infinite. This normally displays a single column, with a two-column indentation for each level of the directory structure. When **li** reaches a directory with no subdirectories, it lists the contents of that directory in multiple-column form. Specifying either **-Ra** or **-Rp** suppresses the indentation and multiple-column display. These flags display either the full (**-Ra**) or relative (**-Rp**) path names of each file found. The **-Rq** flag also lists the contents of archive files. When using the **-Rq** flag to list the contents of remote archive files, the user and group fields display as a **-** (minus sign) unless the **-k** flag is specified. With the **-k** flag, the user and group fields for archive entries display as raw as found in the archive. (See the archive file format in *AIX Operating System Technical Reference.*)

-S [acmnrsux]
Describes the order in which the listing is to be displayed. The default order is by name (**n**). The **-Sx** flag specifies no sorting. Choosing a flag from the subset **acmnsu** selects which field the listing will be sorted by:

a	Accessed time, latest first
c	Character count, largest first

m	Modified time, latest first
n	Name
s	Size (same as character count)
u	Updated time, latest first

If you include the **r** flag with any of these, **li** reverses the order of the sort.

The miscellaneous flags are:

-a Lists all entries, including those beginning with **.** (dot).

-d Lists only the name, not the contents, of directories.

-f Forces **li** to interpret each *file* as a directory and to list the name found in each slot. All flags requiring information not found in directory entries are turned off and the **-a** flag is turned on. Names are listed in the order that they appear in the directory.

-k Provides a listing that is equivalent to `li -Ibcfmpr`. That is, it lists the permission code, node ID, remote UID, remote GID, time of last modification, character count, and file name for remote entries.

-l Uses a listing that is equivalent to `li -Icglmop` (the long form listing). That is, it lists the permission code, link count, owner, group, character count, time of last modification, time of last access, and name of each file.

> **Note:** A symbolically linked file is followed by an → and then the contents of the symbolic link.

-n Inhibits the interpretation of control characters in file names. This flag is useful for generating lists of file names for program input or for editing into per-file commands.

-s Provides a listing similar to that of the **-v** flag, except that the distinguishing marks for file types do not affect sorting (a sortable verbose list). Subdirectories appear in the listing as *name/*, files with execute permission as *name**, special files as *name?*, and symbolic links as *name@*.

-v Lists files in a way that visually differentiates file types (a verbose visual listing). With this flag, **li** lists subdirectories as [*name*], files with execute permission as <*name*>, special files as **name**, and symbolic links as @*name*@. This differentiation occurs before the **-S** sort. Thus, different types of files are sorted into different parts of the listing.

-x Displays every available field except headers (an extended form listing). This is equivalent to specifying `li -Iabcfglimoprsu`.

-num Lists with a maximum of *num* columns. If *num* is unreasonable, **li** picks its own *num*. This flag can be used as in `li -1` to make shell files or `li-Io9` to force **li** to display its output in multiple columns. A number appearing in any flag argument is assumed to be the number of columns unless it follows the **-R** flag.

Examples

1. To list the files in the current directory in alphabetical order:

   ```
   li
   ```

2. To list all files in the current directory, including those with names beginning with a . (dot):

   ```
   li -a
   ```

3. To display detailed information:

   ```
   li -l chapl .profile
   ```

   This displays a long listing with detailed information about `chapl` and **.profile**. It lists all the information that you probably need to see. However, **li** can supply even more information with the **-x** flag.

4. To display detailed information about a directory:

   ```
   li -d -l . manual manual/chapl
   ```

   This displays a long listing for the directories . and `manual`, and for the file `manual/chapl`. **-d** flag, this would list the files in . and `manual` instead of the detailed information about the directories themselves.

5. To list the files in order of modification time:

   ```
   li -Sm -l
   ```

   This displays a long listing of the files that were modified most recently, followed by the older files.

6. To include extra information in the listing:

   ```
   li -Ichil
   ```

   In addition to the file name, this lists the character count (**-Ic**), i-number (**-Ii**), and link count (**-Il**) for each file in the current directory. The **-Ih** tells **li** to write a heading at the top of each column of information.

7. To list the contents of each directory in a tree:

   ```
   li -R manual
   ```

   This lists the names in each subdirectory of the tree that starts with `manual`.

Files

/etc/passwd	Contains user names for `li -Io`.
/etc/group	Contains group names for `li -Ig`.

Related Information

The following commands: "**ctab**" on page 257 and "**ls**" on page 595.

The **chmod** system call and the **environment** miscellaneous facility in *AIX Operating System Technical Reference.*

"Overview of International Character Support" in *Managing the AIX Operating System.*

"Distributed Services Concepts" in *Managing the AIX Operating System.*

line

Purpose

Reads one line from the standard input.

Syntax

```
line ──┤
```

OL805142

Description

The **line** command copies one line from standard input and writes it to standard output. It returns an exit value of 1 on an end-of-file and always writes at least a new-line character. Use this command within a shell command file to read from your work station.

Example

To read a line from the keyboard and append it to a file:

```
echo 'Enter comments for the log:'
echo ': \c'
line >>log
```

This shell procedure displays the message:

```
Enter comments for the log:
```

then reads a line of text from the work station keyboard and adds it to the end of `log`. The `echo ': \c'` command displays a colon prompt. See "**echo**" on page 369 for information about the **\c** escape sequence.

Related Information

The following command: "**sh**" on page 913.

The **read** system call in *AIX Operating System Technical Reference.*

link, unlink

Purpose

Performs a link or unlink system call.

Syntax

```
link — file1 — file2 —|
```

OL805143

```
unlink — file —|
```

OL805227

Description

The **link** and **unlink** commands perform the corresponding system calls of the same name on the specified file, abandoning all error checking. These commands can be run only by a user operating with superuser authority (see "**su**" on page 1026). You should be familiar with the **link** and **unlink** system calls described in *AIX Operating System Technical Reference.*

The **link** and **unlink** commands do not issue error messages when the associated system call fails; you must check the exit value to determine if the command completed normally. Each returns a 0 if it succeeds, a 1 if you specify too few or too many parameters, and a 2 if its system call fails.

Warning: The **link** and **unlink** commands allow the superuser to deal with unusual problems, such as moving an entire directory to a different part of the directory tree. They also permit you to create directories that cannot be reached or escaped from. Be careful to preserve directory structure.

To preserve directory structure observe the following rules:

- Be certain every directory has a **.** (dot) link to itself.
- Be certain every directory has a **..** (dot dot) link to its parent directory.
- Be certain every directory has no more than one link to it.
- Be certain every directory is accessible from the root of its file system.

Note: If the . (dot) entry has been destroyed and **fsck** is unable to repair it (a rare occurrence), you can use the **link** command to restore the . (dot) entry of the damaged directory with the command: **link** *dir dir***/.** where *dir* is the name of the damaged directory. However, use this only as a last resort when the directory is destroyed and **fsck** is unable to fix it.

Related Information

The following commands: "**ln**" on page 581 and "**fsck**, **dfsck**" on page 445.

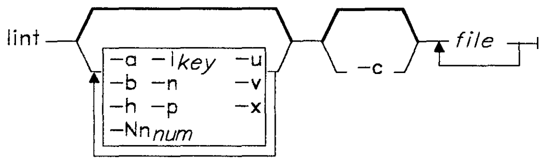
The **link** and **unlink** system calls in *AIX Operating System Technical Reference*.

lint

Purpose

Checks C programs for potential problems.

Syntax



OL805433

Description

The **lint** program checks C language source code for coding and syntax errors and for inefficient or nonportable code. You can use this program to:

- Identify source code and library incompatibility
- Enforce type checking rules more strictly than does the compiler
- Identify potential problems with variables
- Identify potential problems with functions
- Identify problems with flow control
- Identify legal constructions that may produce errors or be inefficient
- Identify possibly nonportable code.

The **lint** command assumes that *file* names ending in **.c** are C Language source files. It assumes that those ending in **.ln** are the result of an earlier running of **lint** with either the **-c** or the **-o** flag used. These **.ln** files are analogous to the **.o** (object) files produced by the **cc** command when given a **.c** file as input. **lint** warns you about files with other suffixes and ignores them.

The **lint** command takes all the **.c** and **.ln** files and the libraries specified by **-l** flags and processes them in the order that they appear on the command line. By default, it adds the standard **lint** library (**llib-lc.ln**) to the end of the list of files. However, when you select the **-p** flag, **lint** uses the portable library **llib-port.ln**. By default, the second pass of **lint** checks this list of files for mutual compatibility; however, if you specify the **-c** flag, **lint** ignores the **.ln** and **lib-l***x* files.

The **-c** and **-o** flags allow for incremental use of **lint** on a set of C Language source files. Generally, you use **lint** once for each source file with the **-c** flag. Each of these runs produces a **.ln** file that corresponds to the **.c** file and writes all messages that are about just that source file. After you have run all source files separately through **lint**, you run it once more, without the **-c** flag, listing all the **.ln** files with the needed **-l** arguments. This writes all interfile inconsistencies. This procedure works well with the **make** command, allowing it to run **lint** on only those source files that have been modified since the last time that set of source files was checked.

The following comments in a C source program change the way that **lint** operates when checking the source program:

/*NOTREACHED*/	Suppresses comments about unreachable code.
/*VARARGS*n/**	Suppresses checking the following function declaration for varying numbers of arguments but does check the data type of the first *n* arguments. If you do not include a value for *n*, **lint** checks no arguments ($n = 0$).
/*ARGSUSED*/	Turns on the **-v** flag for the next function.
/*LINTLIBRARY*/	If you place this comment at the beginning of a file, **lint** does not identify unused functions in the file.

The **lint** command first writes messages about each source file as it processes the file. It collects messages about included files and writes those after it has gone through all the source files. Finally, if you have not specified the **-c** flag, it collects information gathered from all input files and checks it for consistency. At this point, if it is not clear whether a message stems from a given source file or from one of its included files, **lint** displays the source file name followed by a question mark.

Flags

-a	Suppresses messages about assignments of long values to variables that are not long.
-b	Suppresses messages about unreachable break statements.
-h	Does not try to detect bugs, improve style, or reduce waste.
-c	Causes **lint** to produce a **.ln** file for every **.c** file on the command line. These **.ln** files are the product of the first pass of **lint** only and are not checked for interfunction compatibility.
-l*key*	Includes the additional **lint** library **llib-l***key***.ln**. You can include a **lint** version of the math library **llib-lm.ln** by specifying **-lm** on the command line or **llib-ldos.ln** by specifying **-ldos** on the command line. Use this flag to include local **lint** libraries when checking files that are part of a project having a large number of files. This flag does not prevent **lint** from using the **llib-lc.ln** library.

-n	Does not check for compatibility with either the standard or the portable **lint** libraries.
-Nn*num*	Increases the size of the symbol table. The default size is 1500.
-o *lib*	Causes **lint** to create a lint library with the name **llib-l***lib***.ln**. The **-c** flag nullifies any use of the **-o** flag. The lint library produced is the input that is given to the second pass of **lint**. The **-o** flag simply causes this file to be saved in the named lint library. To produce a **llib-l***lib***.ln** without extraneous messages, use the **-x** flag. The **-v** flag is useful if the source files for the lint library are just external interfaces (for example, the way the file **llib-lc** is written). These flag settings are also available through the use of lint comment lines.
-p	Checks for portability to other C dialects.
-u	Suppresses messages about functions and external variables that are either used and not defined or defined and not used. Use this flag to run **lint** on a subset of files of a larger program.
-v	Suppresses messages about function parameters that are not used.
-x	Suppresses messages about variables that have external declarations but are never used.

In addition, **lint** recognizes the following flags of the **cpp** command (macro preprocessor):

-Dname[=*def*]	Defines the **name**, as if by **#define**. The default *def* is **1**.
-I*dir*	Adds *dir* to the list of directories in which **lint** searches for **#include** files.
-Uname	Removes any initial definition of **name**, where **name** is a reserved symbol that is predefined by the particular preprocessor.

Examples

1. To check a C program for errors:

```
lint  program.c
```

2. To suppress some of the messages:

```
lint  -v  -x  program.c
```

   This checks `program.c`, but does not display error messages about unused function parameters (**-v**) or unused externals (**-x**).

3. To check the program against an additional lint library:

   ```
   lint  -lsubs  program.c
   ```

   This checks `program.c` against both the standard lint library (**/usr/lib/llib-lc.ln**) and **/usr/lib/llib-lsubs.ln**.

4. To check against the portable library and an additional library:

   ```
   lint -lsubs -p program.c
   ```

   This checks `program.c` against both the portable lint library (**/usr/lib/llib-port.ln**) and **/usr/lib/llib-lsubs.ln**.

5. To check against a nonstandard library only:

   ```
   lint -lsubs -n program.c
   ```

   This checks `program.c` against only **/usr/lib/llib-lsubs.ln**.

Files

/usr/lib/lint[12]	Programs.
/usr/lib/llib-lc.ln	Declarations for standard functions (binary format).
/usr/lib/llib-lc	Declarations for standard functions (source).
/usr/lib/llib-port.ln	Declarations for portable functions (binary format).
/usr/lib/llib-port	Declarations for portable functions (source).
/usr/lib/llib-lm.ln	Declarations for standard math functions (binary format)
/usr/lib/llib-lm	Declarations for standard math functions (source)
/usr/lib/llib-ldos.ln	Declarations for **shell** functions (binary format).
/usr/lib/llib-ldos	Declarations for **shell** functions (source).
/usr/tmp/*lint*	Temporary files.

Related Information

The following command: "**cc**" on page 140.

The topic "Checking C Programs" in *AIX Operating System Programming Tools and Interfaces.*

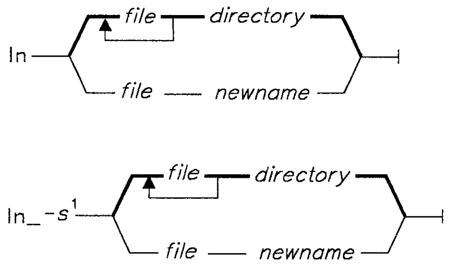
"Overview of International Character Support" in *Managing the AIX Operating System.*

ln

Purpose

Links files.

Syntax



[1]This flag is not supported by Japanese Language Support

OL805028

Description

The **ln** command links *file* to *newname* (in the current directory), or to the same name (*file*) in another existing *directory*. You can link directories, provided the two directories have the same parent.

If you are linking a file to a new name, you can list only one *file*. If you are linking to a *directory*, you can list more than one *file*.

Note: You can only link files across file systems using symbolic links.

Flags

-s Creates a symbolic link to a file or directory. Specifying the complete path name of the file or directory is recommended.

Examples

1. To create another name (also called an alias) for a file:

```
ln  chapl  intro
```

This links `chap1` to the new name `intro`. If `intro` does not already exist, the file name is created. If `intro` does exist, the file is replaced by a link to `chap1`. Now `chap1` and `intro` are two file names that refer to the same file. Any changes made to one also appear in the other. If one name is deleted with **del** or **rm**, the file is not actually deleted, but remains under the other name.

2. To link a file to the same name in another directory:

   ```
   ln  index  manual
   ```

   This links `index` to the new name `manual/index`.

Note the difference: `intro` in Example 1 is the name of a file; `manual` in Example 2 is a directory that already exists.

3. To link several files to names in another directory:

   ```
   ln  chap2  jim/chap3  /u/manual
   ```

   This links `chap2` to the new name `/u/manual/chap2` and `jim/chap3` to `/u/manual/chap3`.

4. To use **ln** with pattern-matching characters:

   ```
   ln  manual/*  .
   ```

   This links all files in the directory `manual` into the current directory (.), giving them the same names they have in `manual`. Note that you must type a space between the asterisk and the period.

Related Information

The following commands: "**rm**" on page 833, "**mv**" on page 679, and "**cp**" on page 202.

The **chmod** and **link** system calls in *AIX Operating System Technical Reference.*

The symbolic link section in *Using the AIX Operating System.*

locator

Purpose

Controls the sample rate of the locator.

Syntax

```
locator —— -rrate ——|
```

OL805444

Description

The **locator** command sets the *rate* at which the system checks, per second, the cursor position controlled by the mouse. You can specify any of the following *rate*s: **10**, **20**, **40**, **60**, **80**, or **100**. Initially, at system startup, this rate is set at **60**.

Note: You can run the **locator** command only from the system console.

Flag

-r*rate* Sets the sampling *rate* to the specified value.

Example

To set the locator rate to **40**:

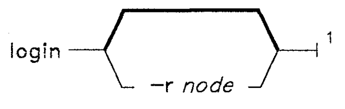
```
locator  -r40
```

login

Purpose

Allows you to sign on to the system and performs user identification and authentication.

Syntax



[1] This command is not normally entered on the command line

OL805005

Description

The **login** program logs you in to the system and performs user authentication. Its primary functions are the following:

- Identify the user and validates the user's password
- Make the required audit, accounting, and log entries
- Execute **loginx** or **passwd**.

A ***logger process***, initially running the **getty** program, is started for each enabled port. The **getty** command reads a login name and sets work station modes (see "**getty**" on page 490). Then it runs **login**, which may ask for a password. If you do not have a password, press the **Enter** key.

Your login attempt might fail for the following reasons:

- Your login name/password pair does not match an entry in the password file.
- Your password has expired. This can happen if your system requires that you change your password after a set number of days. In this case, **login** runs the **passwd** command instead of letting you log in. (For more information, see "**passwd**" on page 735.) After you change your password, you can attempt to log in again.
- The system has reached the limit of simultaneously logged-in users. Each AIX kernel sets a limit on the number of concurrent logins by nonprivileged users; this limit may be one. A ***privileged*** user is one that has a user ID from 0 to 20. A privileged user can log in at any time.

- Your account is no longer active. The administrator of the system has invalidated your account by specifying the **nouse** value for the **restrictions** attribute in the **/etc/security/passwd** file. This failure results in the message: `You cannot login with this account..`
- You are not allowed to login with this account. The administrator of the system has prevented logins to your account by specifying the **nologin** value for the **restrictions** attribute in the **/etc/security/passwd** file. This failure results in the message: `You cannot login with this account..`

In one special case, **login** does not ask for a user name and password pair. When the login port is the console and the file **/etc/autolog** contains a valid user name, **login** creates a login session for that user automatically. Other processing by **login** proceeds normally.

When a user logs in successfully, the **login** program makes entries in **/etc/utmp**, the record of users logged in to the system, and in **/usr/adm/wtmp** (if it exists), for use in accounting. On invalid login attempts (due to incorrect login names or passwords), **login** makes entries in the **/etc/.ilog** file.

When you log in as user **root** or **su** and the **/etc/.ilog** file is not empty, you see a message advising you to check the **/etc/.ilog** file for a record of unsuccessful login attempts.

Environment variables inherited from **getty** and **init** (such as those specified in **/etc/environment**) are kept. You may expand or modify the environment by supplying additional parameters to **login** when it requests your login name. These may take the form *xxx* or *xxx*=*yyy*. Parameters without an equal sign are placed in the environment as L*num*=*xxx*, where *num* is a number starting at 0 and incremented each time a new variable name is required. Parameters containing an equal sign are placed into the environment without modification. If they already exist, the new assignment replaces the older value. However, you cannot change the shell variables **PATH** and **SHELL**. (This restriction prevents people who log in to restricted environments from creating unrestricted secondary shells.)

Flags

-r*node*	Identifies the login as a remote login and specifies the node requesting the login.

Files

/etc/utmp	Accounting file.
/usr/adm/wtmp	Accounting file.
/etc/.ilog	Accounting file.
/etc/autolog	Login ID for automatic login.
/etc/passwd	Password file.
/etc/security/config	File containing security-relevant information.
/etc/security/passwd	File containing password information.

Related Information

The following commands: "**users, adduser**" on page 1129, "**csh**" on page 225, "**getty**" on page 490, "**init**" on page 521, "**passwd**" on page 735, and "**pstart, penable, pshare, pdelay**" on page 791.

Note: The **csh** command contains a built-in subcommand named **login**. The command and subcommand do not necessarily work the same way. For information on the subcommand, see the **csh** command.

The **passwd** and **utmp** files in *AIX Operating System Technical Reference.*

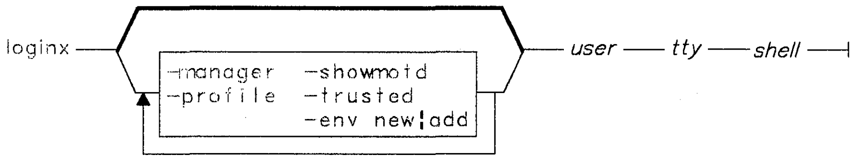
The discussion of login sessions in *Managing the AIX Operating System.*

loginx

Purpose

Sets up a user's execution environment.

Syntax



A5ACG022

Description

This command is called by **login** and **su**. The **loginx** command sets up your execution environment and enhances the environment according to the flags you select. It then runs the specified shell on the specified terminal.

This program always sets the uid and gid to that of the specified user. It sets the user's audit classes as defined in **/etc/security/passwd** as well as the file size limit for this user. In addition, it sets the HOME, LOGNAME, PATH, MAIL, and SHELL environment variables.

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Flags

-manager	The **-manager** flag causes this command to read the **/etc/ports** file for the **shell** attribute and executes that value. The **shell** given on the command line becomes the first parameter to the attribute value.
-profile	The **-profile** flag causes **loginx** to change the current directory to the user's home directory as specified in **/etc/passwd**. When the user's shell is run, its name is preceded by a - (minus) sign.
-showmotd	The **-showmotd** flag causes **/etc/motd** to be displayed on the screen.

-trusted	The **-trusted** flag causes the terminal to become untrusted and the terminal mode and ownership to change. The mode changes to 0600, and the ownership changes to that of the specified user.
-env new¦add	The **-env** flag must be followed by either **new** or **add** If **new** is specified, the current shell environment variables are cleared. If **add** is specified, the current shell environment variables are preserved.

Files

/etc/ports	Contains the names and characteristics of the system terminal ports.
/etc/security/passwd	Contains password information necessary for security.

Related Information

The following commands: "**login**" on page 584, "**su**" on page 1026, "**shell**" on page 938, "**tty**" on page 1105, and "**users, adduser**" on page 1129

The **/etc/passwd** and **/etc/ports** files in *AIX Operating System Technical Reference.*

The discussion of login sessions in *Managing the AIX Operating System.*

logname

Purpose

Displays your login name.

Syntax

```
logname ──┤
```

OL805145

Description

The **logname** command writes to standard output the name you used to log in to the system. It is the contents of the environment variable **$LOGNAME**, which is set when you log in to the system.

Files

/etc/profile	System profile.

Related Information

The following commands: "**env**" on page 393 and "**login**" on page 584.

The **logname** subroutine *AIX Operating System Technical Reference.*

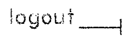
The **environ** special facility in *AIX Operating System Technical Reference.*

logout

Purpose

Stops all processes on a port, returning it to a dead state.

Syntax



OL805485

Description

The **logout** command provides a thorough method of logging off your system. The command verifies that the user invoking **logout** logged in at the same port. If the **login** user and the **logout** user do not match, **logout** permission is denied, and the command stops.

Note: Only the superuser can redirect standard input for this command. The **logout** command is a setuid-root program. It sends a **SIGKILL** to the process group leader running on the port from which the **logout** command is entered. This command also issues the **revoke()** system call for the **logout** port and any synonym of that port.

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Files

/etc/utmp	Contains a record of logged-in users.
/etc/ports	Specifies the port synonym.

Related Information

The following commands: "**login**" on page 584 and "**tsh**" on page 1100.

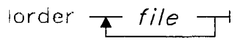
The **/etc/ports** and the **/etc/utmp** file formats in *AIX Operating System Technical Reference.*

lorder

Purpose

Finds the best order for member files in an object library.

Syntax



OL805029

Description

The **lorder** command reads one or more object or library archive *file*s, looking for external references and writing a list of paired file names to standard output. The first of each paired files contains references to identifiers that are defined in the second file. You can send this list to the **tsort** command to find an ordering of a library member file suitable for one-pass access by **ld**.

If object files do not end with **.o**, **lorder** overlooks them and attributes their global symbols and references to some other file.

Example

To create a subroutine library:

```
lorder charin.o scanfld.o scan.o scanln.o ¦ tsort ¦ xargs ar qv libsubs.a
```

This creates a subroutine library named `libsubs.a` that contains `charin.o`, `scanfld.o`, `scan.o`, and `scanln.o`. The ordering of the object modules in the library is important. The **ld** command requires each module to precede all the other modules that it calls or references. The **lorder** and **tsort** commands together add the subroutines to the library in the proper order.

Suppose that `scan.o` calls `scanfld.o` and `scanln.o`. `scanfld.o` also calls `charin.o`. First, the **lorder** command creates a list of pairs that shows these dependencies:

```
charin.o charin.o
scanfld.o scanfld.o
scan.o scan.o
scanln.o scanln.o
scanfld.o charin.o
scanln.o charin.o
scan.o scanfld.o
```

Next, the ¦ (vertical bar) sends this list to the **tsort** command, which converts it into the ordering we need:

```
scan.o
scanfld.o
scanln.o
charin.o
```

Note that each module precedes the module it calls. `charin.o`, which does not call another module, is last.

The second ¦ then sends this list to **xargs**, which constructs and runs the following **ar** command:

```
ar  qv  libsubs.a  scan.o  scanfld.o  scanln.o  charin.o
```

This **ar** command creates the properly ordered library.

Files

/tmp/sym*	Temporary files.

Related Information

The following commands: "**ar**" on page 55, "**ld**" on page 557, "**nm**" on page 705, "**tsort**" on page 1102, and "**xargs**" on page 1232.

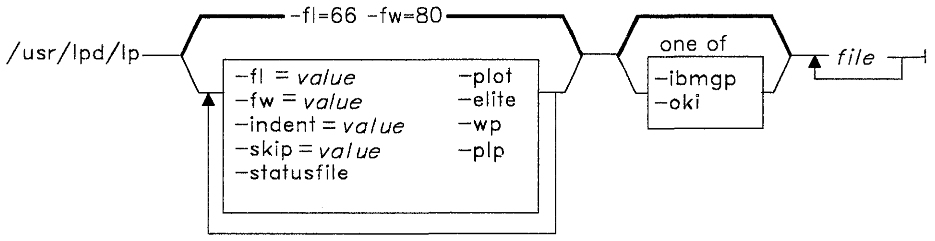
The **ar** file in *AIX Operating System Technical Reference.*

lp

Purpose

Prints a file in a format suitable for sending to a line printer.

Syntax



OL805396

Description

The **lp** command prints *file* on its standard output in a form that is suitable for a line printer. The **lp** command is normally invoked by the **qdaemon** command. **qdaemon** directs the output from **lp** to the appropriate device.

Flags are passed to **lp** in the following ways:

- Flags specified in the **qconfig** structure are passed each time that **lp** is invoked. The **-plp**, **-ibmgp**, **-oki**, and **-statusfile** flags most likely appear in **qconfig**.
- Flags that are not recognized by the **print** command are assumed to be for **lp** and are passed to **lp** with the requested job.

Flags

-elite	Prints the text at 12 characters per inch instead of 10 characters per inch. This flag changes the default forms width to 96 characters.
-fl=*value*	Sets the forms length equal to *value*. The default length is 66 lines.
-fw=*value*	Sets the forms width equal to *value*. The default width is 80 columns. Lines that are wider than *value* are truncated. If you set *value* to 0, no truncation is performed.
-ibmgp	Specifies an IBM Graphic Printer.

-indent=*value*	Indents the printed output the number of spaces specified with *value*.
-oki	Specifies an Okidata Model 92 or 93.
-plot	Passes text directly to the printer without processing. This is useful when using the printer as a plotter. Normally, lines that contain backspaces and carriage-return characters are processed so that they print with minimum print head motion. The sequence **ESC-9** maps to half-line feeds.
-plp	Sets and resets printer port parameters, if the printer is attached with a parallel interface.
-skip=*value*	Does not print the first *value* blank lines in the file.
-statusfile	Updates the status information in the status file that is open on file descriptor 3. The status information is passed from **qdaemon**.
-wp	Sets the printer, if possible, to the Word Processing mode.

Related Information

The following commands: "**print**" on page 767 and "**qdaemon**" on page 802.

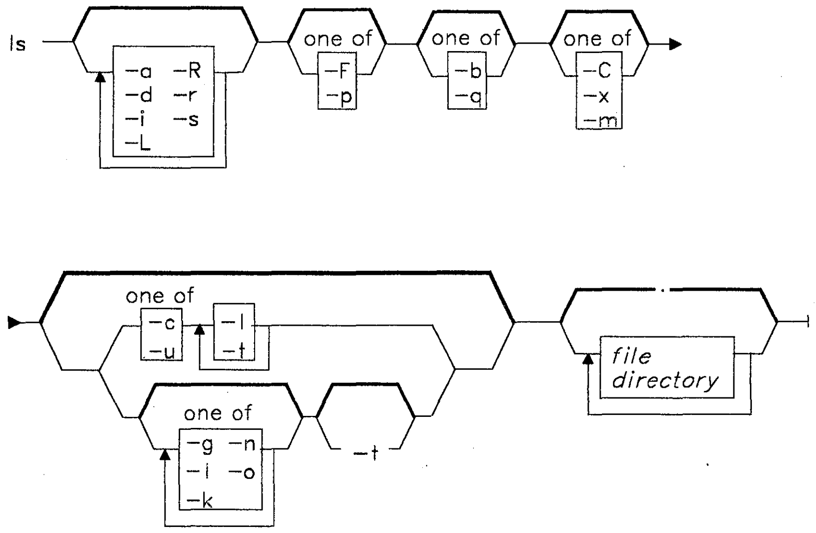
The **qconfig** file in *AIX Operating System Technical Reference.*

ls

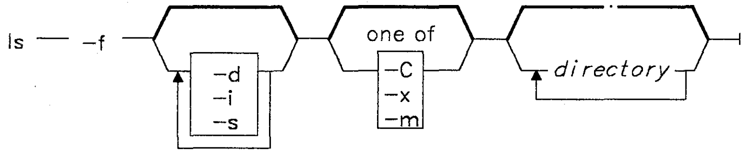
Purpose

Displays the contents of a directory.

Syntax



OL805030



OL805243

Description

The **ls** command writes to standard output the contents of each specified *directory* or the name of each specified *file*, along with any other information you ask for with the flags. If you do not specify a *file* or *directory*, **ls** displays the contents of the current directory.

By default, **ls** displays all information in alphabetic order by file name. The collating sequence is determined by the **NLCTAB** environment variable (see "**ctab**" on page 257). Individual file names are listed before directory names.

There are three main ways to format the output:

- List one entry per line. This is the default format.
- List entries in multiple columns by specifying either the **-C** or **-x** flags.
- List entries in a comma-separated series by specifying the **-m** flag.

To determine the number of character positions in the output line, **ls** uses the environment variable **COLUMNS**. If this variable is not set, it reads the **terminfo** file. If **ls** cannot determine the number of character positions by either of these methods, it uses a default value of 80.

The mode displayed with the **-l** flag is interpreted as follows:

If the first character is:

d	The entry is a directory.
b	The entry is a block special file.
c	The entry is a character special file.
l	The entry is a symbolic link.
p	The entry is a first-in first-out (FIFO) special file.
-	The entry is an ordinary file.
D	The entry is a remote directory.
F	The entry is a remote ordinary file.
B	The entry is a remote block special file.
C	The entry is a remote character special file.
L	The entry is a remote symbolic link.
P	The entry is a remote first-in first-out (FIFO) special file.

The next nine characters are divided into three sets of three characters each. The first three characters show the owner's permission. The next set of three characters show the permission of the other users in the group. The last set of three characters show the permission of any one else with access to the file. The three characters in each set show read, write, and execute permission of the file. Execute permission of a directory lets you search a directory for a specified file.

Permissions are indicated as follows:

r	You can read the file.
w	You can edit (write) the file.
x	You can search the file.
-	You do not have permission to access the file.

The group-execute permission character is *s* if the file has set-group ID mode. The user-execute permission character is *s* if the file has set-user-ID mode. The last character of the mode (normally *x* or -) is *t* if the 1000 (octal) bit of the mode is set; see "**chmod**" on page 160 for the meaning of this mode. The indications of set-ID and 1000 bit of the mode are capitalized (*S* and *T* respectively) if the corresponding execute permission is not set.

When the size of the files in a directory are listed, the **ls** command displays a total count of blocks, including indirect blocks.

The environment variables **NLLDATE** and **NLTIME** control the format of the date and time. The environment variable **NLSMONTH** controls the short names of months.

Flags

-a Lists all entries in the directory including the entries that begin with a **.** (dot).

-b Displays nonprintable characters in an octal \nnn notation.

-c Uses the time of last modification of the i-node (file created, mode changed, and so on) for sorting (when used with **-t**) or for displaying (when used with **-l**). This flag has no effect when not used with either **-t** or **-l** or both.

-C Sorts output vertically in a multicolumn format.

-d Displays only the information for the directory named. This is useful with the **-l** flag to get the status of a directory.

-f Lists the name in each slot for each named *directory*. This flag turns off **-l**, **-t**, **-s**, and **-r**, and turns on **-a**; the order is the order in which entries appear in the directory.

-F Puts a / (slash) after each file name if the file is a directory, an * (asterisk) after each file name if the file can be executed, and an @ sign after each file name if the file is a symbolic link.

-g Displays the same information as with **-l**, except for the owner.

-i Displays the i-number in the first column of the report for each file.

-k Displays the permission codes, node ID, remote UID, remote GID, time of last modification, size (in bytes), and file name for remote entries.

For remote files and directories, the local owner and local group are obtained by using inverse IDs. If there is no inverse ID or if **ls** cannot determine the inverse ID, a **-** (minus sign) displays in the corresponding field. If possible, remote nodes are identified with nicknames. Otherwise, they are identified by their NID displayed in hexadecimal. (See "Distributed Services Concepts" in *Managing the AIX Operating System.*)

For local files and directories that do not have a nickname defined for the local node ID, the node ID field displays as a **-** (minus sign), and the raw UID (GID) field contains the local owner UID (group GID).

-l Displays the mode, number of links, owner, group, size (in bytes), and time of last modification for each file. If the file is a special file, the size field will instead contain the major and minor device numbers.

> **Note:** A symbolically linked file is followed by an → and then the contents of the symbolic link.

-L Follows a symbolic link and reports on the file at the end of the link.

-m Uses stream output format (a comma-separated series).

-n Displays the same information as with **-l**, except that it displays the user and the group IDs instead of the user and group names.

-o Displays the same information as with **-l**, except for the group.

-p Puts a slash after each file name if that file is a directory. This is useful when you pipe the output of **ls** to the **pr** command as follows:

```
ls -p | pr -5 -t -w80
```

-q Displays nonprintable characters in file names as the character ?.

-r Reverses the order of the sort, giving reverse alphabetic or the oldest first, as appropriate.

-R Lists all subdirectories recursively.

-s Gives size in blocks (including indirect blocks) for each entry.

-t Sorts by time of last modification (latest first) instead of by name.

-u Uses the time of the last access instead of time of the last modification for sorting (when used with **-t**) or for displaying (when used with **-l**). This flag has no effect when not used with either **-t** or **-l** or both.

-x Sorts output horizontally in a multicolumn format.

Examples

1. To list all files in the current directory:

   ```
   ls  -a
   ```

   This lists all files, including . (dot), .. (dot-dot), and other files with names beginning with a dot.

2. To display detailed information:

   ```
   ls  -l  chap1  .profile
   ```

   This displays a long listing with detailed information about `chap1` and **.profile**.

3. To display detailed information about a directory:

   ```
   ls  -d  -l  .  manual  manual/chapl
   ```

   This displays a long listing for the directories . and `manual`, and for the file `manual/chapl`. Without the **-d** flag, this would list the files in . and `manual` instead of the detailed information about the directories themselves.

4. To list the files in order of modification time:

   ```
   ls  -l  -t
   ```

   This displays a long listing of the files that were modified most recently, followed by the older files.

Files

/etc/passwd	Contains user IDs.
/etc/group	Contains group IDs.
/usr/lib/terminfo/*	Contains terminal information.

Related Information

The following commands: "**chmod**" on page 160, "**ctab**" on page 257, and "**find**" on page 422.

The **environment** miscellaneous facility in *AIX Operating System Technical Reference.*

"Overview of International Character Support" in *Managing the AIX Operating System.*

"Distributed Services Concepts" in *Managing the AIX Operating System.*

The symbolic link section in *Using the AIX Operating System.*

Task Index

This index groups commands by task. Each command listing includes a command, a page reference, and a description of the command. Commands are grouped under the following tasks:

Managing the System TASK-3
- Installing and Maintaining Programs TASK-3
- Configuring the System TASK-3
- Backing Up and Restoring System Files TASK-4
- Managing File Systems TASK-4
- Analyzing System Activity TASK-5
- Performing System Accounting Functions TASK-6

Controlling System Security TASK-6
- Managing System Auditing TASK-6
- Managing the Secure System TASK-6
- Managing the Trusted Path TASK-7
- Managing Access Permissions and Ownerships TASK-7

Using the System TASK-8
- Starting and Stopping the System TASK-8
- Using Shells and Interfaces TASK-8
- Displaying System Statistics and Information TASK-8
- Controlling System Processes TASK-9
- Using Disks and Diskettes TASK-10
- Using Tape TASK-10
- Working with Work Stations TASK-10

Working with Files and Directories TASK-11
- Working with Directories TASK-11
- Creating and Editing Files TASK-12
- Printing and Displaying Files TASK-12
- Copying and Moving Files TASK-12
- Deleting Files TASK-13
- Comparing Files TASK-13
- Scanning Files TASK-13
- Sorting Files TASK-14
- Merging and Splitting Files TASK-14
- Working with Remote Files TASK-14
- Formatting Text TASK-15
- Working with Graphics TASK-15
- Protecting Files with File Permissions TASK-16
- Backing Up and Restoring Files TASK-16

Working with Data TASK-16
- Using Data Tools TASK-16

  Performing Calculator Functions TASK-17
Communicating on the System TASK-17
  Sending Messages and Notices TASK-17
  Using Mailboxes TASK-17
  Using the MH (Message Handling) Package TASK-17
  Communicating with Other Systems TASK-19
Developing Programs TASK-19
  Programming in Assembler TASK-19
  Programming in C TASK-19
  Programming in Miscellaneous Languages TASK-20
  Programming in Shell TASK-20
  Working with Messages TASK-21
  Debugging Programs TASK-21
  Managing Source Programs Using the Source Code Control System (SCCS) TASK-21
  Managing Object Files TASK-22
Playing Games TASK-22

Managing the System

The following groups of commands are important for managing various aspects of the system.

Installing and Maintaining Programs

cvid	272	Creates a VRM install diskette for backup purposes.
install	524	Installs a command.
installp	529	Installs a licensed program.
make	625	Maintains up-to-date versions of programs.
mdrc	640	Allows you to reinstall a user-created minidisk after you have reinstalled AIX.
ndtable	685	Accesses the Distributed Services Node Table.
pwtable	801	Accesses the Distributed Services Node Security Table.
updatep	1122	Updates one or more programs.

Configuring the System

bffcreate	108	Creates files in backup format for complete or subset programs in a code service environment.
biodd_cfg	115	Configures the block I/O AIX device driver.
chkcomp	158	Checks compatibility between a code server and an active-service client.
chngstate	164	Changes the state of a code service client to either active-service or stand-alone.
chparm	171	Changes or examines system parameters.
config	194	Extracts configuration information from configuration files.
devices	315	Adds, deletes, changes, and displays device information.
defkey	306	Defines keyboard key assignments.
display	332	Selects the physical display that an existing or new virtual terminal uses and sets colors and fonts.
dsipc	354	Installs the Interprocess Communication key mapping in the kernel.
dsldxprof	355	Loads translate information into the UID/GID translate profiles.
dsxlate	363	Installs Distributed Services UID/GID translate tables into the kernel.
env	393	Sets the environment for execution of a command.
getty	490	Sets the characteristics of ports.
init	521	Initializes the system.
ipctable	544	Creates, displays, or changes the Distributed Services IPC Queues Table.
keyboard	551	Controls the delay and repetition rates of the keyboard.
locator	583	Controls the sample rate of the locator.
minidisks	650	Adds, deletes, changes, and displays minidisks.

mknod	661	Creates a special file.
pdisable, phold	741	Kills the logger running on the specified port.
pstart, penable, pshare, pdelay	791	Enables or reports the availability of login ports.
rc	806	Performs normal startup initialization.
setdma	910	Sets the DMA channel of the specified adapter.
sound	967	Controls the volume and click of the keyboard speaker.
splp	975	Changes or displays printer driver settings.
stty	1018	Sets, resets, or reports work station operating parameters.
termdef	1062	Queries terminal characteristics.
trdiag	1097	Starts diagnostics on the Token-Ring Network.
ugtable	1109	Creates, displays, and changes the Distributed Services Network Users/Groups Table.
users, adduser	1129	Adds, deletes, and changes user and group information.
varyon	1180	Makes an external disk drive and any minidisks or file systems defined on it available for use.
varyoff	1177	Removes an external disk drive from the operating system configuration.
verify	1186	Turns write verification on or off for a particular minidisk.
vrmconfig	1206	Installs peripheral devices.

Backing Up and Restoring System Files

backup	88	Backs up files.
pack	730	Compresses files.
restore	826	Copies back files created by the **backup** command.
tapechk	1047	Performs consistency checking of the streaming tape device.
tar	1048	Manipulates archives.
tctl	1058	Gives commands to streaming tape.

Managing File Systems

basename	95	Returns the base name of a string parameter.
chroot	172	Changes the root directory of a command.
clri	175	Clears the specified i-node.
cpio	205	Copies files into and out of archive storage and directories.
crash	215	Examines system images.

dcopy	299	Copies file systems for the best access time.
devnm	316	Names a device.
df	318	Reports number of available disk blocks.
env	393	Sets the environment for execution of a command.
ff	417	Lists the file names and statistics for a file system.
fsck	445	Checks file system consistency and interactively repairs the file system.
fsdb	450	Debugs file systems.
istat	545	Examines i-nodes.
link	575	Performs a link or unlink system call.
mkfs	658	Makes a file system.
mknod	661	Creates a special file.
mount	669	Makes a file system available for use.
ncheck	683	Generates path names from i-numbers.
proto	780	Constructs a prototype file for a file system.
setmnt	911	Creates mount table.
skulker	951	Cleans up file systems by removing unwanted files.
sync	1030	Updates the superblock and writes buffered files to the fixed disk.
umount, unmount	1112	Unmounts a previously mounted file system, directory, or file.
varyon	1180	Makes an external disk drive and any minidisks or file systems defined on it available for use.
varyoff	1177	Removes an external disk drive from the operating system configuration.

Analyzing System Activity

du	364	Summarizes disk usage.
dumpfmt	368	Formats the VRM dump file.
errdead	397	Extracts error records from dump.
errdemon	398	Starts the error-logging daemon.
errpt	400	Processes a report of logged errors.
errstop	404	Terminates the error-logging daemon.
errupdate	405	Updates an error report template.
fuser	455	Identifies processes using a file or file structure.
profiler	775	Profiles the operating system.
time	1068	Times the execution of a command.
trace	1086	Starts the trace function.
trcrpt	1091	Formats a report from the trace log file.
trcstop	1093	Stops the trace function.
trcupdate	1094	Updates trace format templates.

Performing System Accounting Functions

acct/*	13	Provides accounting shell procedures.
acctcms	18	Produces command usage summaries from accounting records.
acctcom	20	Displays selected process accounting record summaries.
acctcon	24	Performs connect-time accounting.
acctdisk, acctdusg	26	Performs disk-usage accounting.
acctmerg	28	Merges total accounting files.
acctprc	30	Performs process accounting.
diskusg	330	Generates disk accounting data by user ID.
fwtmp	457	Manipulates connect accounting records.
runacct	848	Runs daily accounting.
sadc	863	Provides a system activity report package.

Controlling System Security

The following groups of commands are important to ensure the security of the system.

Managing System Auditing

audit	67	Controls system auditing.
auditapp	69	Adds an audit bin file to the end of the audit trail file.
auditbin	71	Manages bins of audit information.
auditpr	73	Displays audit trail files.
auditselect	76	Selects audit records.
auditstream	78	Creates a channel for the reading of audit records.
auditwrite	80	Generates an audit record at the command level.

Managing the Secure System

init	521	Initializes the system.
login	584	Allows you to sign on to the system and performs user identification and authentication.
loginx	587	Sets up a user's execution environment.
passwd	735	Changes login password.
print	767	Enqueues a file.
secure	885	Establishes a more secure system configuration
shell	938	Executes a shell in a user's login environment.

sysck	1031	Verifies the secure system state.
tsh	1100	Interprets commands in a trusted shell.
tvi	1108	Acts as trusted editor for system administration.
watch	1209	Observes and reports security-relevant actions on the system.

Managing the Trusted Path

actman	32	Permits interaction with multiple virtual terminals.
getty	490	Sets the characteristics of ports.
init	521	Initializes the system.
login	584	Allows you to sign on to the system and performs user identification and authentication.
loginx	587	Sets up a user's execution environment.
logout	590	Stops all processes on a port, returning it to a dead state.
shell	938	Executes a shell in a user's login environment.
tsh	1100	Interprets commands in a trusted shell.
tvi	1108	Acts as trusted editor for system administration

Managing Access Permissions and Ownerships

chgrp	156	Changes the group ownership of a file or directory.
chmod	160	Changes permission codes.
chown	169	Changes the owner of files or directories.
chtcb	174	Sets or queries the tcb attribute of a file.
groups	506	Displays your group membership.
id	517	Displays the system identity of the user issuing the command.
logname	589	Displays your login name.
login	584	Allows you to sign on to the system and performs user identification and authentication.
makekey	634	Generates an encryption key.
newgrp	689	Changes your primary group identification.
passwd	735	Changes login password.
pwck	798	Checks the password and group files for inconsistencies.
su	1026	Obtains the privileges of another user, including superuser authority.
umask	1110	Displays and sets file-creation permission code mask.
users, adduser	1129	Adds, deletes, and changes user and group information.

Using the System

The following groups of commands help you use the various functions of the system.

Starting and Stopping the System

actman	32	Permits interaction with multiple virtual terminals.
login	584	Allows you to sign on to the system and performs user identification and authentication.
open	728	Opens a virtual terminal.
passwd	735	Changes login password.
shutdown	946	Ends system operation.

Using Shells and Interfaces

actman	32	Permits interaction with multiple virtual terminals.
csh	225	Interprets commands read from a file or entered from the keyboard.
dos	341	Starts **shell.**
sh	913	Interprets commands read from a file or entered at the keyboard.
tsh	1100	Interprets commands in a trusted shell.

Displaying System Statistics and Information

date	281	Displays or sets the date.
devices	315	Adds, deletes, changes, and displays device information.
diskusg	330	Generates disk accounting data by user ID.
dsstate	361	Sets the state of the Distributed Services kernel logic.
errpt	400	Processes a report of logged errors.
errupdate	405	Updates an error report template.
file	420	Determines file type.
fptype	444	Displays the floating point configuration of the system.
fuser	455	Identifies processes using a file or file structure.
groups	506	Displays your group membership.
help	513	Provides information about a Source Code Control System (SCCS) message or command or about certain non-SCCS commands.
id	517	Displays the system identity of the user issuing the command.
ipcs	539	Reports interprocess communication facility status.
ipctable	544	Creates, displays, or changes the Distributed Services IPC Queues Table.
istat	545	Examines i-nodes.

logname	589	Displays your login name.
man	635	Displays manual entries online.
minidisks	650	Adds, deletes, changes, and displays minidisks.
ncheck	683	Generates path names from i-numbers.
ndtable	685	Accesses the Distributed Services Node Table.
news	691	Writes system news items to standard output.
od	723	Writes the contents of storage to the standard output.
profiler	775	Profiles the operating system.
PostScript	786	Reports process status.
pstart, penable, pshare, pdelay	791	Enables or reports the availability of login ports.
pwck	798	Checks the password and group files for inconsistencies.
pwd	800	Displays the path name of the working directory.
pwtable	801	Accesses the Distributed Services Node Security Table.
rstatd	847	Returns NFS performance statistics from the kernel.
rup	854	Displays the host status of local machines.
rusers	856	Identifies users logged in on network hosts.
sact	862	Displays current Source Code Control System (SCCS) file editing status.
sadc	863	Provides a system activity report package.
sag	865	Displays a graph of system activity.
sar	867	Collects, reports, or saves system activity information.
splp	975	Changes or displays printer driver settings.
stty	1018	Sets, resets, or reports work station operating parameters.
sum	1029	Displays the checksum and block count of a file.
time	1068	Times the execution of a command.
timex	1069	Times a command, and reports process data and system activity.
tty	1105	Writes to standard output the full path name of your work station.
ugtable	1109	Creates, displays, and changes the Distributed Services Network Users/Groups Table.
uname	1114	Displays the name of the current operating system.
uustat	1158	Reports the status of and provides rudimentary job control for BNU commands.
who	1219	Identifies the users currently logged in.

Controlling System Processes

at	63	Runs commands at a later time.
cron	220	Runs commands automatically.
crontab	222	Submits a schedule of commands to **cron**.

errdemon	398	Starts the error-logging daemon.
errstop	404	Terminates the error-logging daemon.
kill	552	Sends a signal to a running process.
killall	555	Cancels all processes except the calling process.
nice	699	Runs a command at a different priority.
nohup	707	Runs a command without hangups and quits.
open	728	Opens a virtual terminal.
qdaemon	802	Schedules jobs enqueued by the **print** command.
sleep	952	Suspends execution for an interval.
syslogd	1037	Reads and logs messages.
tlog	1071	Stops or restarts sending of terminal I/O to a daemon.
tlogger	1072	Gathers I/O from a terminal and writes it to a log file.
writesrv	1230	Allows Distributed Services users to send messages to and receive messages from a remote system.

Using Disks and Diskettes

format	436	Formats diskettes.
mdrc	640	Allows you to reinstall a user-created minidisk after you have reinstalled AIX.
minidisks	650	Adds, deletes, changes, and displays minidisks.
mount	669	Makes a file system available for use.
umount, unmount	1112	Unmounts a previously mounted file system, directory, or file.
varyon	1180	Makes an external disk drive and any minidisks or file systems defined on it available for use.
varyoff	1177	Removes an external disk drive from the operating system configuration.
verify	1186	Turns write verification on or off for a particular minidisk.

Using Tape

tapechk	1047	Performs consistency checking of the streaming tape device.
tar	1048	Manipulates archives.
tctl	1058	Gives commands to streaming tape.

Working with Work Stations

defkey	306	Defines keyboard key assignments.
display	332	Selects the physical display that an existing or new virtual terminal uses and sets colors and fonts.

echo	369	Writes its arguments to standard output.
hp	514	Handles special functions for the HP2640- and HP2621-series terminals.
keyboard	551	Controls the delay and repetition rates of the keyboard.
locator	583	Controls the sample rate of the locator.
pdisable, phold	741	Kills the logger running on the specified port.
pstart, penable, pshare, pdelay	791	Enables or reports the availability of login ports.
stty	1018	Sets, resets, or reports work station operating parameters.
tabs	1041	Sets tab stops on work stations.
termdef	1062	Queries terminal characteristics.
tic	1067	Translates **terminfo** files from source to compiled format.
tput	1081	Queries the **terminfo** file.
tty	1105	Writes to standard output the full path name of your work station.
300	1262	Handles special line-motion functions for DASI 300/300s work stations.
4014	1264	Formats a full page 66-line screen display for a Tektronix 4014 work station.
450	1265	Handles special line-motion functions for the DASI 450 work station.

Working with Files and Directories

The following groups of commands allow you to create and manipulate files and directories.

Working with Directories

cd	150	Changes the current directory.
chroot	172	Changes the root directory of a command.
dircmp	328	Compares two directories and the contents of their common files.
dosdir	346	Lists the directory for DOS files.
find	422	Finds files matching expression.
li	567	Lists the contents of a directory.
ls	595	Displays the contents of a directory.
mkdir	657	Makes a directory.
mvdir	682	Moves (renames) a directory.
pwd	800	Displays the path name of the working directory.
rm	833	Removes files or directories.
rmdir	838	Removes a directory.

Creating and Editing Files

admin	41	Creates and initializes SCCS files.
cdc	152	Changes the comments in a Source Code Control System (SCCS) delta.
ed	371	Edits text by line.
edit	387	Provides a simple line editor for the new user.
ex	407	Edits lines interactively, with screen display.
get	477	Creates a specified version of a Source Code Control System (SCCS) file.
mknod	661	Creates a special file.
sed	887	Provides a stream editor.
spell	969	Finds spelling errors.
tab	1040	Changes spaces into tabs or tabs into spaces.
uniq	1118	Deletes repeated lines in a file.
vi	1187	Edits files with a full screen display.

Printing and Displaying Files

cat	137	Concatenates or displays files.
cut	269	Writes out selected fields from each line of a file.
lp	593	Prints a file in a format suitable for sending to a line printer.
nl	701	Numbers lines in a file.
od	723	Writes the contents of storage to the standard output.
pg	744	Formats files to the work station.
piobe	753	Writes a file to standard output in a format suitable for sending to a line printer.
pr	761	Writes a file to standard output.
prs	781	Displays a Source Code Control System (SCCS) file.
print	767	Enqueues a file.
qdaemon	802	Schedules jobs enqueued by the **print** command.
splp	975	Changes or displays printer driver settings.
tail	1044	Writes a file to standard output, beginning at a specified point.
vc	1182	Substitutes assigned values in place of keywords.

Copying and Moving Files

cat	137	Concatenates or displays files.
cp	202	Copies files.
Cvt	274	Moves old UUCP files into new BNU directories.
dd	301	Converts and copies a file.
dosread	348	Copies a DOS file.

doswrite	350	Copies AIX files to DOS files.
ln	581	Links files.
mv	679	Moves files.
uucp	1144	Copies files from one AIX system to another AIX system.
uuto	1162	Copies public files from one AIX system to another AIX system, with local system control of file access.

Deleting Files

del	308	Deletes files if the request is confirmed.
dosdel	345	Deletes DOS files.
rm	833	Removes files or directories.
uniq	1118	Deletes repeated lines in a file.
uucleanup	1141	Deletes selected files older than a specified number of hours from the BNU spool directory or a named directory.

Comparing Files

bdiff	102	Uses **diff** to find differences in very large files.
cmp	177	Compares two files.
comm	183	Selects or rejects lines common to two sorted files.
diff	320	Compares text files.
diff3	323	Compares three files.
diffmk	326	Marks differences between files.
dircmp	328	Compares two directories and the contents of their common files.
sdiff	883	Compares two files and displays the differences in a side by side format.
sccsdiff	874	Compares two versions of a Source Code Control System (SCCS) file.

Scanning Files

awk	81	Finds lines in files matching specified patterns and performs specified actions on them.
bfs	110	Scans files.
file	420	Determines file type.
find	422	Finds files matching expression.
grep	501	Searches a file for a pattern.
hyphen	516	Finds hyphenated words.
wc	1211	Counts the number of lines, words, and characters in a file.
what	1213	Displays identifying information in files.

Sorting Files

lorder	591	Finds the best order for member files in an object library.
sort	958	Sorts or merges files.
tsort	1102	Sorts an unordered list of ordered pairs (a topological sort).

Merging and Splitting Files

csplit	252	Splits files by context.
join	547	Joins data fields of two files.
paste	736	Merges the lines of several files or subsequent lines in one file.
sort	958	Sorts or merges files.
split	974	Splits a file into pieces.

Working with Remote Files

biod	114	Starts daemons that handle NFS block I/O requests.
domainname	340	Displays or sets YP domain name.
makedbm	632	Makes a Yellow Pages **dbm** map.
mountd	674	Answers NFS mount requests.
nfsd	696	Starts NFS client request daemons.
nfsstat	697	Displays NFS statistics.
on	726	Executes a command remotely via NFS.
pcnfs	739	Serves PC-NFS client requests.
portmap	757	Maps RPC programs to the servicing ports on RPC servers.
rexd	832	Executes remote programs.
rpcgen	843	Compiles a Remote Procedure Call program.
rpcinfo	845	Reports Remote Procedure Call status information.
rstatd	847	Returns NFS performance statistics from the kernel.
rup	854	Displays the host status of local machines.
rusers	856	Identifies users logged in on network hosts.
rusersd	858	Displays list of active NFS users.
rwall	859	Writes to all network users.
rwalld	861	NFS daemon for **rwall** and **shutdown**.
showmount	945	Displays list of remotely mounted file systems.
spray	981	Sprays packets to host when NFS is installed.
sprayd	983	Receives packets sent by the **spray** command.
ypbind	1239	Allows Yellow Pages client processes to communicate with the YP server.
ypcat	1241	Displays values in a Yellow Pages data base.
ypinit	1243	Builds and installs the Yellow Pages data base.

ypmatch	1245	Displays the values of one or more keys from a Yellow Pages map.
yppasswd	1247	Changes Yellow Pages passwords
yppasswdd	1249	Handles Yellow Pages password change requests
yppoll	1251	Displays the version of a Yellow Pages map located at a Yellow Pages host.
yppush	1252	**yppush**.
ypserv	1256	Looks up information in a local Yellow Pages data base.
ypset	1254	Directs the **ypbind** daemon to a particular server.
ypwhich	1258	Displays the name of the host machine acting as the Yellow Pages server or as a map YP master server.
ypxfr	1260	Transfers a Yellow Pages map to the local host machine.

Formatting Text

col	179	Processes text having reverse linefeeds and forward/reverse half-linefeeds for output to standard output.
cw	275	Prepares constant-width text for **troff**.
deroff	313	Removes **nroff, troff**, **troff**, **tbl**, and **eqn** constructs from files.
eqn	395	Formats mathematical text for the **nroff, troff** and **troff** commands.
fmt	428	Formats mail messages prior to sending.
greek	499	Converts output for a Teletype Model 37 work station to output for other work stations.
hp	514	Handles special functions for the HP2640- and HP2621-series terminals.
mm	663	Displays or checks documents formatted with Memorandum Macros.
mmt	666	Typesets documents, manual pages, view graphs, and slides.
newform	686	Changes the format of a text file.
nl	701	Numbers lines in a file.
nroff, troff	709	Formats text for printing devices.
ptx	794	Generates a permuted index.
tbl	1053	Formats tables for the **nroff, troff** and **troff** commands.
tc	1056	Simulates phototypesetter output for a Tektronix 4014 work station.
vrm2rtfont	1205	Converts a standard AIX font file to RT rtx font format.
300	1262	Handles special line-motion functions for DASI 300/300s work stations.
4014	1264	Formats a full page 66-line screen display for a Tektronix 4014 work station.
450	1265	Handles special line-motion functions for the DASI 450 work station.

Working with Graphics

gdev	460	Provides graphical device routines and filters.
ged	463	Displays, makes, and edits graphical files on Tektronix 4010 terminals.
gend	475	Provides a general graphics device backend.

graph	494	Draws a graph.
graphics	497	Accesses graphical and numerical commands.
gutil	508	Provides graphical utility programs.
spline	972	Interpolates smooth curve.
stat	984	Provides tools for analyzing numerical data.
toc	1074	Provides graphical table of contents routines.
tplot	1079	Produces plotting instructions for a particular work station.

Protecting Files with File Permissions

chgrp	156	Changes the group ownership of a file or directory.
chmod	160	Changes permission codes.
chown	169	Changes the owner of files or directories.
groups	506	Displays your group membership.
li	567	Lists the contents of a directory.
ls	595	Displays the contents of a directory.
umask	1110	Displays and sets file-creation permission code mask.

Backing Up and Restoring Files

ar	55	Maintains portable libraries used by the linkage editor.
backup	88	Backs up files.
cpio	205	Copies files into and out of archive storage and directories.
lorder	591	Finds the best order for member files in an object library.
pack	730	Compresses files.
restore	826	Copies back files created by the **backup** command.
shlib	939	Creates a shared library.
tar	1048	Manipulates archives.

Working with Data

The following groups of commands allow you to use various data tools.

Using Data Tools

banner	94	Writes character strings in large letters to standard output.
cal	132	Displays a calendar.
calendar	134	Writes reminder messages to standard output.
ctab	257	Produces a collating table.

echo	369	Writes its arguments to standard output.
tr	1083	Translates characters.
units	1119	Converts units in one measure to equivalent units in another measure.

Performing Calculator Functions

bc	97	Provides an interpreter for arbitrary-precision arithmetic language.
dc	295	Provides an interactive desk calculator for doing arbitrary-precision integer arithmetic.
factor	416	Factors a number.

Communicating on the System

The following groups of commands allow you to use mail and message facilities on the system.

Sending Messages and Notices

confer	189	Provides an online conferencing system.
mesg	642	Permits or refuses **write** messages.
news	691	Writes system news items to standard output.
wall	1208	Writes a message to all logged-in users.
write	1225	Sends messages to other users on the system.

Using Mailboxes

bellmail	104	Sends messages to system users and displays messages from system users.
fmt	428	Formats mail messages prior to sending.
mail, Mail	608	Sends and receives mail.
mailstats	623	Displays statistics regarding mail traffic.
sendmail	897	Routes mail for local or network delivery.

Using the MH (Message Handling) Package

ali	48	Lists mail aliases and their addresses.
anno	50	Annotates messages.
ap	53	Parses and reformats addresses.
burst	129	Explodes digests into messages.
comp	185	Composes a message.

conflict	196	Searches for alias and password conflicts.
dist	336	Redistributes a message to additional addresses.
dp	352	Parses and reformats dates.
folder	429	Selects and lists folders and messages.
folders	433	Lists folders and messages.
forw	438	Forwards messages.
inc	518	Incorporates new mail.
install-mh	527	Initializes the MH environment.
mark	637	Creates, modifies, and displays message sequences.
mhl	643	Produces formatted listings of messages.
mhmail	646	Sends or receives mail.
mhpath	648	Prints full path names of messages and folders.
msgchk	675	Checks for messages.
msh	677	Creates an MH shell.
next	694	Shows the next message.
packf	733	Compresses the contents of a folder into a file.
pick	748	Selects messages by content, and creates and modifies sequences.
post	758	Delivers a message.
prev	765	Shows the previous message.
prompter	778	Invokes a prompting editor.
rcvdist	808	Sends a copy of incoming messages to additional recipients.
rcvpack	810	Saves incoming messages in a packed file.
rcvstore	812	Incorporates new mail from standard input into a folder.
rcvtty	815	Notifies the user of incoming messages.
refile	817	Files messages in other folders.
repl	821	Replies to a message.
rmf	839	Removes a folder.
rmm	841	Removes messages.
scan	871	Produces a one line per message scan listing.
send	893	Sends a message.
show	942	Shows messages.
slocal	954	Processes incoming mail.
sortm	965	Sorts messages.
spost	978	Delivers a message.
vmh	1203	Invokes a visual interface for use with MH commands.
whatnow	1215	Invokes a prompting interface for draft disposition.
whom	1222	Lists the addresses of the proposed recipients of a message and verifies the addresses.

Communicating with Other Systems

connect	198	Establishes a connection to a remote system.
ct	254	Dials an attached terminal and issues a login process.
cu	263	Connects directly or indirectly to another UNIX system.
Cvt	274	Moves old UUCP files into new BNU directories.
sum	1029	Displays the checksum and block count of a file.
uucheck	1137	Checks for files and directories required by BNU.
uucheck	1137	Checks for files and directories required by BNU.
uucico	1139	File transport program for the BNU facility.
uucleanup	1141	Deletes selected files older than a specified number of hours from the BNU spool directory or a named directory.
uucp	1144	Copies files from one AIX system to another AIX system.
uulog	1149	Provides information about **uucp** and **uux** activities on a system.
uuname	1151	Provides information about other systems accessible to the local system.
uucpadm	1133	Enters basic **uucp** configuration information.
uusched	1156	Schedules work for the BNU file transport program.
uustat	1158	Reports the status of and provides rudimentary job control for BNU commands.
uuto	1162	Copies public files from one AIX system to another AIX system, with local system control of file access.
uutry, Uutry, uukick	1164	Contacts a remote system with debugging turned on.
uux	1166	Runs a command on another AIX system.
uuxqt	1172	Executes remote command requests.

Developing Programs

The following groups of commands are for use in programming.

Programming in Assembler

as	61	Assembles a source file.
sdb	875	Provides a symbolic debugger for C and assembler programs.

Programming in C

cb	139.	Puts C source code into a form that is easily read.

cc	140	Compiles C programs.
cflow	154	Generates a C flow graph of external references.
cpp	210	Performs file inclusion and macro substitution on C Language source files.
cxref	279	Creates a C program cross-reference listing.
factor	416	Factors a number.
ipcrm	537	Removes message queue, semaphore set or shared memory identifiers.
m4	603	Preprocesses files, expanding macro definitions.
lex	562	Generates a C Language program that matches patterns for simple lexical analysis of an input stream.
lint	577	Checks C programs for potential problems.
regcmp	820	Compiles patterns.
sdb	875	Provides a symbolic debugger for C and assembler programs.
tic	1067	Translates **terminfo** files from source to compiled format.
yacc	1237	Generates a LR(1) parsing program from input consisting of a context-free grammar specification.

Programming in Miscellaneous Languages

bc	97	Provides an interpreter for arbitrary-precision arithmetic language.
bs	118	Compiles and interprets modest-sized programs.
ctags	261	Makes a file of tags to help locate objects in source files.
m4	603	Preprocesses files, expanding macro definitions.
sno	956	Provides a SNOBOL interpreter.

Programming in Shell

basename	95	Returns the base name of a string parameter.
cron	220	Runs commands automatically.
crontab	222	Submits a schedule of commands to **cron**.
csh	225	Interprets commands read from a file or entered from the keyboard.
echo	369	Writes its arguments to standard output.
env	393	Sets the environment for execution of a command.
expr	412	Evaluates arguments as expressions.
find	422	Finds files matching expression.
getopt	485	Parses command line flags and parameters.
line	574	Reads one line from the standard input.
nice	699	Runs a command at a different priority.
nohup	707	Runs a command without hangups and quits.
open	728	Opens a virtual terminal.
sh	913	Interprets commands read from a file or entered at the keyboard.

sleep	952	Suspends execution for an interval.
tee	1060	Displays the output of a program and copies it into a file.
test	1064	Evaluates conditional expressions.
time	1068	Times the execution of a command.
true	1099	Returns an exit value of zero.
xargs	1232	Constructs argument lists and runs commands.

Working with Messages

dspcat	357	Displays all or part of a message catalog.
dspmsg	359	Displays a selected message from a message catalog.
gencat	470	Creates and modifies a message catalog.
gettext	488	Extracts message/insert/help descriptions.
mkcatdefs	651	Preprocesses a message source file.
puttext	796	Updates an output file that contains message/insert/help descriptions.
runcat	852	Pipes data from **mkcatdefs** to **runcat**.

Debugging Programs

crash	215	Examines system images.
dbx	284	Provides a tool to debug and run programs under AIX.
dump	366	Dumps selected parts of an object file.
dumpfmt	368	Formats the VRM dump file.
od	723	Writes the contents of storage to the standard output.
prof	773	Displays program profile data.
adb	33	Provides a general purpose debugger.
sdb	875	Provides a symbolic debugger for C and assembler programs.
time	1068	Times the execution of a command.
timex	1069	Times a command, and reports process data and system activity.
xdbx	1236	Provides an overlaying X-Window application for the dbx symbolic debugger.

Managing Source Programs Using the Source Code Control System (SCCS)

admin	41	Creates and initializes SCCS files.
cdc	152	Changes the comments in a Source Code Control System (SCCS) delta.
comb	181	Combines SCCS deltas.
delta	310	Creates a delta in a Source Code Control System file.
get	477	Creates a specified version of a Source Code Control System (SCCS) file.
help	513	Provides information about a Source Code Control System (SCCS) message or command or about certain non-SCCS commands.

prs	781	Displays a Source Code Control System (SCCS) file.
rmdel	837	Removes a delta from a Source Code Control System (SCCS) file.
sact	862	Displays current Source Code Control System (SCCS) file editing status.
sccsdiff	874	Compares two versions of a Source Code Control System (SCCS) file.
unget	1116	Cancels a previous **get** command.
val	1175	Validates Source Code Control System (SCCS) files.
what	1213	Displays identifying information in files.

Managing Object Files

ar	55	Maintains portable libraries used by the linkage editor.
as	61	Assembles a source file.
dump	366	Dumps selected parts of an object file.
ld	557	Links object files.
lorder	591	Finds the best order for member files in an object library.
make	625	Maintains up-to-date versions of programs.
nm	705	Displays the symbol table of an object file.
prof	773	Displays program profile data.
size	949	Displays the section sizes of common object files.
strip	1017	Removes symbol and line number information from a common object file.
touch	1077	Updates the access and modification times of a file.
tsort	1102	Sorts an unordered list of ordered pairs (a topological sort).

Playing Games

The following groups of commands allow you to play games on the system.

arithmetic	59	Tests arithmetic skills.
back	87	Plays backgammon.
bj	117	Plays blackjack.
craps	214	Plays craps.
fish	427	Plays the card game Go Fish.
fortune	437	Tells a fortune.
hangman	512	Plays hangman, the word-guessing game.
moo	668	Plays a number-guessing game.
number	721	Displays the written form of a number.
quiz	803	Tests your knowledge.
ttt	1104	Plays tic-tac-toe.
turnon	1107	Turns on execute permission for games.
wump	1231	Plays the game Hunt the Wumpus.

Index

Special Characters

$- 925
$! 924
$$ 924
$? 924
$# 924

A

abs command 988
access times of a file, changing 1077
accounting
    ASCII format 29
    ASCII summary format 19
    billing summary file 848
    binary summary format 19
    combining total accounting files 28
    connect 24
    daily 848
    disk 14
    line-usage summary 25
    login 15
    merging total accounting files 28
    monthly reports 15
    process 14, 30
    reports 15
    session 15
    shell procedures 13
    start 16
    turn off process 16
    usage summaries 18
accounting commands
    acctcms 18
    acctcom 20
    acctcon1 24
    acctcon2 25
    acctdisk 26
    acctdusg 26
    acctmerg 28
    accton 31
    acctprc1 30
    acctprc2 31
    acctwtmp 458
    chargefee 14
    ckpacct 14
    dodisk 14
    fwtmp 457
    lastlogin 15
    monacct 15
    nulladm 15
    prctmp 15
    prdaily 15
    prtacct 16
    runacct 848
    shutacct 16
    startup 16
    turnacct 16
    wtmpfix 458
accounting file 15
accounting files
    /usr/adm/acct/fiscal 15
    /usr/adm/acct/nite/active 848
    /usr/adm/acct/nite/ctmp 15
    /usr/adm/acct/nite/lastdate 848
    /usr/adm/acct/nite/lock 848
    /usr/adm/acct/nite/lock1 848
    /usr/adm/acct/nite/statefile 848
    /usr/adm/acct/sum 15
    /usr/adm/acct/sum/loginlog 15
    /usr/adm/acct/sum/rprt 15
    /usr/adm/fee 14
    /usr/adm/pacct 14
        acctcom 20
        accton 31
        ckpacct 14
        turnacct 16

/usr/adm/wtmp
    acctcon1 24
  billing summary file 848
  creating 15
accounting records
  ASCII 30
  ASCII format 24
  converting ASCII to binary 457
  converting binary to ASCII 457
  display 20
  examining connect records 457
  login session 24
  repairing wtmp records 458
  session 24
  total accounting login session 25
accounting report, process 21
accounting, disk usage 26
acctcms command 18-19
acctcom command 20-23
acctcon1 command 24-25
acctcon2 command 25
acctdisk command 26
acctdusg command 26-27
acctmerg command 28-29
accton command 31
acctprc2 command 31
acctwtmp command 458
activity graph, system 865
activity manager 32
activity reporter, system 863, 867
actman command 32
adb command 33-40
adding
  audit bin file 69
  devices 315
  groups 1129
  header flags, SCCS 45
  users 1129
  users, SCCS 45
adduser command 1129-1132
admin command 41-47
Advanced Floating-Point Accelerator 143
Advanced Processor Card 143
af command 989
AIX device driver 115
AIX-KJ commands
  awk command 83, 84
  bs command 125
  cal command 132
  character class expressions 413, 423, 502, 916
  cpio command 206
  csh command 225, 232
  ctab command 257, 260
  cut command 270
  dd command 302
  delta command 312
  df command 319
  ed command 375
  expr command 413
  file command 420
  find command 423
  getty command 491
  grep command 502, 504
  lex command 564
  mkfs command 660
  od command 723
  pattern matching 375, 564
  sh command 916
  sort command 959
  tail command 1044
  tar command 1048
  tr command 1083
  translating characters 1083
  uname command 1114
  xargs command 1233
ali command 48-49
anno command 50-52
ap command 53-54
ar command 55-58
arbitrary precision arithmetic 97
arithmetic game 59-60
arithmetic, shell variable 412
as command 61-62, 143
assembler 61
assembling
  source code
    as 61
    asm 61
    cc 141
    masm 61
at command 63-66

audit command 67-68, 77
auditappend command 69-70, 79
auditbin command 71
auditd command 72
auditing commands
  audit 67
  auditapp 69
  auditbin 71
  auditpr 73
  auditselect 76
  auditstream 78
  auditwrite 80
auditpr command 73-75
auditselect command 76
auditstream command 78
auditwrite command 80
awk command 81-86

B

back game 87
backing up files 88
backup command 88-93
banner command 94
bar command 1009
basename command 95-96
batch command 63-66
bc command 97-101
bdiff command 102-103
bel command 509
bellmail command 104
belonging to different groups 506
bffcreate command 108-109
bfs command 110-113
billing summary file, accounting 848
biod command 114
biodd_cfg command 115-116
bj game 117
blackjack game 117
block count of a file, display 1029
block I/O device driver 115
branching from nonleaf deltas 481
break command 931
bs command 118-128
bucket command 1001
burst command 129-131

C

C Language programming
  See also managing programs
  See also programming
  assembling source code 141
  commands
    ar 55
    as 61
    cb 139
    cc 140
    cflow 154
    cpp 210
    fcc 141
    lint 577
    vcc 141
    vrmfmt 141
  cross-reference listing 279
  files
    a.out 141
  formatting source code 139
  linking object files 141
  maintaining linkage libraries 55
  preprocessing source code 141
  syntax checking 577
cal command 132-133
calculating
  CPU factor 22
  CPU time 21
  hog factor 22
calculator program 97
calculator, desk 295
calendar command 134-136
calprog program 136
case command 930
case in Japanese Language Support 257
cat command 137-138
cb command 139
cc command 140-149, 173
cd command 150-151, 931
cdc command 152-153

CDPATH 922
ceil command 991
cflow command 154-155
changing
  ASCII accounting records to binary 457
  binary accounting records to ASCII 457
  changing permission codes 160
  current directory 150
  devices 315
  files, SCCS 310
  format of a file 686
  group identification 689
  group ownership 156
  groups 1129
  login environment 689
  owner-ID of files or directories 169
  password 735
  primary group 689
  root directory 172
  SCCS delta comments 152
  system parameters 171
  users 1129
changing Distributed Services ipc queues table 544
changing Distributed Services network Users/Groups table 1109
changing Distributed Services node table 685
changing password table 801
changing state values 361
character class expressions (Japanese Language Support) 84, 125, 206, 257, 413, 423, 502, 916
  equivalence classes
    ctab command 257
character classes 4
chargefee command 14
charting
  external references 154
checkcw command 277-278
checkeq command 395-396
checking process accounting files 14
checking programs
  awk command 81
  bs 118
  cvid 272
  managing programs
    awk 81
    make 625
  messages
  miscellaneous languages
    ar 55
    awk 81
checkmm command 663
checksum of a file, display 1029
chgrp command 156-157
  changing
    group ownership 156
chkcomp command 158-159
chmod command 160-163
chng command 168
chngstate command 164
chown command 169-170
chparm command 171
chroot command 172-173
chtcb command 174
ckpacct command 14
ckprereq command 533
clearing an i-node 175
clri command 175-176
cmp command 177-178
col command 179-180
collating sequence
  collating sequence
    csh command 231
    ctab command 257
    sh command 916
    sort command 958
  csh command 231
  ctab command 257
  li command 568
  ls command 596
  NLCTAB environment variable 923
  sh command 916
  sort command 958
colors
  setting active display palette 332
  setting background display 332
  setting foreground display 332
comb command 181-182
combining
  deltas, SCCS 181
  total accounting files 28
comm command 183-184

command execution environment 393
command line flag parsing 485
command usage summary 18
commands
  See accounting commands
  See AIX-KJ commands
  See auditing commands
  See C Language programming
  See communication commands
  See controlled access mode commands
  See editors
  See filter commands
  See graphics commands
  See maintenance commands
  See managing programs, commands
  See Multi-User Services commands
  See programming
  See reading standard input
  See system group commands
  See text processing commands
  See writing to standard output
comments in SCCS files, including kanji characters 312
communication commands
  confer 189
  connect 198
  ct 254
  cu 263
  Cvt 274
  edconfig 385
  fmt 428
  mail, Mail 608
  mailstats 623
  mesg 642
  news 691
  rmail 836
  sendmail 897
  uucheck 1137
  uucico 1139
  uucleanup 1141
  uucp 1144
  uulog 1149
  uuname 1151
  uupick 1153
  uusched 1156
  uustat 1158
  uuto 1162
  uutry 1164
  uux 1166
  uuxqt 1172
  wall 1208
  who 1219
  300 1262
  4014 1264
  450 1265
communication, interprocess status 539
comp command 185-188
comparing
  directories
    dircmp 328
  files 883
    bdiff 102
    cmp 177
    diff 320
    diffmk 326
    diff3 323
    dircmp 328
  SCCS files 874
compilers
  bs 118
  cc 140
  sno 956
compress program 971
compressing files 730
concatenate files 137
concurrent groups 506
conditional expressions, evaluating 1064
confer command 189-193
config command 194-195
configuration information 194
configuring
  block I/O device driver 115
conflict command 196-197
connect accounting 24
connect command 198-201
consistency check and repair of files
  dfsck command 447
  fsck command 445
constant-width text 275
constructing a file system 658
contents of directory, listing 346, 567
context split 252

continue command 931
controlled access mode commands
  audit 67
  auditapp 69
  auditbin 71
  auditpr 73
  auditselect 76
  auditstream 78
  auditwrite 80
  chtcb 174
  getty 491
  login 584
  logout 590
  tsh 1100
  watch 1209
conversion routines
  fromnls 302
  fromsjis (Japanese Language Support) 302
  tonls 302
  tosjis (Japanese Language Support) 302
converting
  ASCII accounting records to binary 457
  binary accounting records to ASCII 457
  kanji characters 302
copy command 202-204
copying
  AIX files
    copy 202
    cp 202
  DOS files
    dosread 348
    doswrite 350
cor command 1002
cp command 202-204
cpio command 205-209
cpp command 143, 210-213
CPU factor computation 22
CPU time computation 21
craps game 214
crash command 215-219
creating
  C program cross-reference listing 279
  delta, SCCS 310
  mount table 911
  SCCS files 41
  special file 661
  specified version of an SCCS file 477
cron command 220-221, 848
  used with the sa1 command 864
  used with the sa2 command 864
crontab command 222-224
cross-reference listing, C program 279
csh command 225-251
csplit command 252-253
ct command 254-256
ctab command 257-260
ctags command 261-262
cu command 263-268
current directory
current directory, changing 150
cusum command 991
cut command 269-271
cvid command 272-273
cvrtopt command 509
Cvt command 274
cw command 275-278
  used in pipeline with nroff 710
cxref command 279-280

D

daemon, error-logging 398
daemon, error-logging termination 404
daemon, terminal logging 1072
daemon, tlogger 1071
daily accounting 848
database operator 547
date command 281-283, 356, 363
dbx 1236
dbx command 284-294
dc command 295-298
dcopy command 299-300
dd command 301-305
debugger, file system 450
defining shell functions 931
defkey command 306-307
del command 308-309
deleting
  delta from SCCS file 837
  devices 315

directories
  rm 833
  rmdir 838
DOS files 345
files
  del 308
  rm 833
  skulker 951
groups 1129
repeated words 1118
users 1129
users, SCCS 45

delta command 310-312
delta summary of SCCS file 482
deltas, branching from nonleaf 481
deroff command 313-314
description file, make command 630
desk calculator 295
device (special) files
  /dev/audit 78
  /dev/null
    acctcom 20
    standard input assigned to 20
  adding 315
  changing 315
  creating 661
  deleting 315
device driver
  block I/O 115
device name 316
devices
devices command 315
devnm command 316-317
df command 318
dfsck command 447-449
di command 567-573
diagnostic
  token-ring 1097
  trdiag 1097
diff command 177, 320-322
diffmk command 326-327
diff3 command 323-325
dircmp command 328-329
directories
  changing
    owner-ID 169
  comparing
    dircmp 328
  listing contents
    di 567
    DOS directories 346
    li 567
    ls 595
  removing
    rm 833
    rmdir 838
directory 682
  change root 172
  changing
  changing current 150
  create 657
  moving 682
  renaming 682
  return path name 95
directory contents, listing 346, 567
dirname command 95-96
disk usage accounting 26
disk usage summary 364
diskusg command 330-331
display command 332-335
display station
  changing DMA pinned page 333
  setting active color palette 332
  setting background colors 332
  setting fonts 332
  setting foreground colors 332
displaying
  a calendar 132
  accounting report 15
  audit record
    auditpr 73
    auditselect 76
  audit trail 73
  compressed files 730
  connect accounting records 457
  contents of i-nodes 545
  corresponding group names and IDs 517
  corresponding user names and IDs 517
  current directory 800
  date 281
  documents formatted with the Memorandum Macros 663

file checksum 1029
files 110, 137
formatted files 744, 761
login name 589
news items 691
packed files 730
process accounting records 20
process status 786
profile data 773
SCCS file editing activity 862
session record 15
squeezed files 730
system images 215
system parameters 171
total accounting report 16
dist command 336-339
Distributed Services
dsldxprof 355
dividing a file into pieces 974
DMA channel, setting 910
dodisk command 14
dos command 341-344
dosdel command 345
dosdir command 346-347
dosread command 348-349
doswrite 350-351
dp command 352-353
drill in arithmetic skills 59
dsipc command 354
dsldxprof command 355
dspcat command 357-358
dspmsg command 359-360
dsstate command 361-362
dsxlate command 363
dtoc command 1074
du command 364-365
dump command 366-367
dump, extracting error records 397
dump, octal 723
dumpfmt command 368

E

echo command 369-370
ed command 371-384
edconfig command 385-386
edit command 387-392
editors
ed 371
edit 387
ex 407
ged 463
red 371
sed 887
tvi 1108
vedit 1187
vi 1187
view 1187
egrep command 501-505
end a process 552
env command 393-394
environment, changing login 689
eqn command 395, 396
constructs removed by the deroff command 313
used in pipeline with nroff 710
used with tbl 1053
erase command 461
errdead command 397
errdemon command 398-399
error-logging daemon 398
error-logging daemon termination 404
error records extraction from dump 397
error report 400
errpd command 399, 403
errpt command 400-403
errstop command 404
errupdate 406
errupdate command 405
eval command 932
evaluating expressions
expr 412
test 1064
ex command 407-411
examining
connect accounting records 457

contents of i-nodes 545
files 110
system images 215
system parameters 171
exec command 932
exercising link system call 575
exit command 932
exp command 992
expanding packed files 730
export command 932
expr command 412-415
expression evaluation 412
extended character support
See international character support
external references, flow graph 154
extract error records from dump 397

F

factor command 416
factoring a number 416
false command 1099
fcc command 141
ff command 417-419
fgrep command 501-505
file
display checksum 1029
file command 420-421
file formats
acct 18, 30
ar 55
backup 88
tacct 28
utmp 458
wtmp 458
file pattern search 501
file system
See also device (special) files
See also maintenance commands
See also system files
backing up 88
make available for use 669
make unavailable for use 1112
making 658
moving a directory 682
renaming a directory 682
unmount 1112
file system debugger 450
files
See also accounting files
See also device (special) files
See also managing programs, files
See also system files
a.out 141
auditing files
backing up 88
calendar 134
changing
owner-ID 169
checking consistency
dfsck command 447
fsck command 445
comparing 883
cmp 177
diff 320
diffmk 326
diff3 323
dircmp 328
comparing large files 102
compressing 730
concatenating 137
copying
AIX files 202
DOS files 348, 350
creating SCCS files 41
deleting
del 308
DOS files 345
determining type 420
displaying 137
displaying formatted files 744, 761
expanding 730
finding 422
identifying the processes using a file 455
initializing SCCS files 41
linking 581
merge lines 736
merging 958
modifying the user mask 1110
naming SCCS files 43

packing 730
parallel merging 736
removing
rm 833
skulker 951
repairing
dfsck command 447
fsck command 445
repairing damage 215
return base name 95
scanning 110
searching 110
searching for a pattern 501
serial merging 736
setting file-creation permission code mask 1110
sorting 958
squeezing 730
text
changing the format 686
transforming 301
translating 301
unpacking 730
unsqueezing 730
writing the last part 1044
3-way comparison 323
filter commands
acctcom 20
acctcon1 24
acctmerg 28
awk 81
bdiff 102
cb 139
cmp 177
col 179
comb 181
cw 275
definition of 914
fwtmp 457
hp 514
nl 701
nroff 710
paste 736
ptx 794
tbl 1053
troff 710
wtmpfix 458
find command 422, 426
acctdusg 26
find hyphenated words 516
find necessary order of files in an object library 591
fish game 427
fixed minidisk information 650
Floating-Point Accelerator 143
floating point configuration 444
floor command 992
flow graph of external references 154
fmt command 428
folder command 429-432
folders command 433-435
fonts
setting virtual terminal 332
for command 930
format command 436
formats
See file formats
formatting C Language source code 139
formatting text
constant-width text 275
for a phototypesetter 709
for a printing device 709
inverse line feeds and half-line feeds 179
mathematical text 395
tables for nroff, troff 1053
fortune game 437
forw command 438-443
forwarding mail 105
fptype command 444
free disk space, reporting 318
fromnls conversion routine 302, 303
fromsjis conversion routine (Japanese Language Support) 302, 303
fsck command 445-449
fsdb command 175, 450-454
fuser command 455-456
fwtmp command 457

G

games
  arithmetic 59
  back 87
  backgammon 87
  bj 117
  blackjack 117
  craps 214
  fish 427
  fortune 437
  hangman 512
  man 635
  moo 668
  number 721
  quiz 803
  ttt 1104
  wump 1231
gamma command 993
gd command 510
gdev commands 460-462
ged command 463-469
gencat command 470-474
gend command 475-476
generating C program cross-reference listing 279
generating names from i-numbers 683
get command 310, 311, 477-484
getopt command 485-487
gettext command 488-489
getty command 490-493
going to maintenance mode 946
graph command 494-496
graph, system activity 865
graphical editor 463
graphics command 497-498
graphics commands
  abs 988
  af 989
  bar 1009
  bel 509
  bucket 1001
  ceil 991
  cor 1002
  cusum 991
  cvrtopt 509
  dtoc 1074
  erase 461
  exp 992
  floor 992
  gamma 993
  gd 510
  ged 463
  gend 475
  graph 494
  graphics 497
  gtop 510
  hardcopy 461
  hilo 1003
  hist 1010
  hpd 460
  label 1011
  list 993
  log 994
  lreg 1004
  mean 1005
  mod 995
  pair 996
  pd 510
  pie 1012
  plot 1014
  point 1006
  power 997
  prime 986
  prod 1006
  ptog 510
  qsort 1007
  quit 510
  rand 987
  rank 1007
  remcom 510
  root 997
  round 998
  siline 999
  sin 1000
  spline 972
  subset 1000
  td 461
  tekset 461
  title 1016
  total 1008

tplot 1079
ttoc 1075
utility commands 508
var 1008
vtoc 1075
whatis 511

graphing
external references 154

greek command 499-500

grep command 501-505

group
adding 1129
changing 1129
deleting 1129

group identification, changing 689

group IDs and names, displaying 517

group membership 506

group membership, display 506

groups command 506-507

gtop command 510

guess a word 512

H

hangman game 512

hardcopy command 461

hash command 932

hashcheck command 970

hashmake command 970

header flags, SCCS 43, 45

help command 513

hilo command 1003

hist command 1010

hog factor computation 22

HOME 922

hp command 514-515

hpd command 460-461

hyphen command 516

hyphenated words 516

I

i-node content, displaying 545

i-node examination 545

i-numbers
generating names 683

I/O counts 22

id command 517

if command 930

IFS 924

inc command 518-520

init command 521-523

initialization, normal startup 806

initializing
SCCS files 41

install command 524-526

install-mh command 527-528

installp command 529-536

interactive processor 97

international character support
at command 64
batch command 64
collating sequence
csh command 231
ctab command 257
dd command 302
dfsck command 447
equivalence classes
fsck command 447
fsdb command 452
li command 568
ls command 596
od command 724
print command 770
ps command 787
sh command 923
sort command 958
stty command 1023

interpolating a smooth curve 972

interpreters
bc 97
bs 118
sno 956

Interprocess Communication key mapping
installation 354

interprocess communication status 539
inudocm command 1125-1126
inuupdt command 1127
ipc queues table
  access 544
ipcrm command 537-538
ipcs command 539-543
ipctable command 544
istat command 545-546

J

join command 547-550
joinconf command 189
joining database files 547
joint editing of an SCCS file 482

K

kanji characters, used in variable names 83
kcore minutes, definition 19
keyboard command 551
keyboard, redefine 306
kill all nonancestral processes 555
kill command 552-554
killall command 555

L

label command 1011
language support
  See international character support
lastlogin command 15
ld command 143, 557-561
lex command 562-566
li command 567-573, 838
LIBPATH 922
library maintainer 55
library search order 144
line command 574
line editor 371
line numbering filter 701
line printer back end 593
link command 575-576
link library maintaining 55
linkage editor 557
linking
  files 581
  object files
    cc 141
    ld 557
lint command 577-580
list command 993
listing
  directory contents
    di 567
    li 567
    ls 595
  DOS directory contents 346
  file names for a file system 417
  statistics for a file system 417
ln command 581-582
locator command 583
log command 994
logged error report 400
login command 584-586
login environment, changing 689
login names 491
login session records 24
loginx command 587-588
LOGNAME 922
logname command 589
logout command 590
looking at
  connect accounting records 457
  contents of i-nodes 545
  files 110
  system images 215
  system parameters 171
lorder command 591-592
lp command 593-594
lreg command 1004
ls command 595-599

M

m4 command 603, 607
macroprocessor 603
MAIL 922
mail command 107
mail, Mail command 608-622
MAILCHECK 922
MAILMSG 923
MAILPATH 922
mailstats command 623-624
maintaining groups of programs 625
maintaining linkage libraries 55
maintenance commands
  backup 88
  clri 175
  cpio 205
  mount 669
  umount 1112
  unmount 1112
maintenance mode, going to 946
make a directory 657
make a tags file 261
make command 625-631
makekey command 634
making a file system 658
making two files the same 102
man pubs 635-636
manager, virtual terminals 32
managing programs
  See also programming
  See also SCCS
  commands
    admin 41
    cdc 152
    comb 181
    delta 310
    get 477
    help 513
    prs 781
    sact 862
    sccsdiff 874
    unget 1116
    val 1175
    what 1213
  files
    auxiliary files 478
    creating a delta 310
    g-file 43, 310, 311, 477, 478, 483
    l-file 478, 482
    lock file 43
    p-file 478, 479, 482
    s-file 478
    x-file 43
    z-file 43, 478
  make 625
  security management
    auditbin command 71
    auditstream command 78
  z-file 479
mant command 666
mark command 637-639
marking differences between files 326
mdrc command 640-641
mean command 1005
membership, display group 506
merging files 958
merging lines in files 736
merging total accounting files 28
mesg command 642
Message Handling commands
  ali 48
  anno 50
  ap 53
  burst 129
  comp 185
  conflict 196
  dist 336
  dp 352
  folder 429
  folders 433
  forw 438
  inc 518
  install-mh 527
  mark 637
  mhl 643
  mhmail 646
  mhpath 648
  msgchk 675
  msh 677
  next 694

packf 733
pick 748
post 758
prev 765
prompter 778
rcvdist 808
rcvpack 810
rcvstore 812
rcvtty 815
refile 817
repl 821
rmf 839
rmm 841
scan 871
send 893
show 942
slocal 954
sortm 965
spost 978
vmh 1203
whatnow 1215
whom 1222
message queue removal 537
messages, permitting 642
messages, refusing 642
messages, send 104
messages, sending 1225
MH commands
See Message Handling commands
mhl command 643-645
mhmail command 646-647
mhpath command 648-649
minidisks 650
minidisks command 650
mkcatdefs command 651-656
mkdir command 657
mkfs command 658-660
mknod command 661-662
mm command 663-665
mmt command 666-667
mod command 995
modification request number 311
modification times of a file, changing 1077
modifying
changing permission codes 160
current directory 150
devices 315
files, SCCS 310
group identification 689
group ownership 156
groups 1129
login environment 689
owner-ID of files or directories 169
password 735
primary group 689
root directory 172
SCCS delta comments 152
system parameters 171
users 1129
modifying access times of a file 1077
modifying modification times of a file 1077
monacct command 15
moo game 668
mount a file system 669
mount command 669-673
mount table, creating 911
mountd command 674
move command 679-681
moving a directory 682
moving files 679
msgchk command 675-676
msh command 677-678
mt command 666
Multi-User Services commands
acctcms 18
acctcom 20
acctcon1 24
acctcon2 25
acctdisk 26
acctdusg 26
acctmerg 28
accton 31
acctprc1 30
acctprc2 31
acctwtmp 458
chargefee 14
ckpacct 14
dodisk 14
fwtmp 457
id 517
lastlogin 15
mesg 642

monacct 15
nulladm 15
prctmp 15
prdaily 15
prtacct 16
runacct 848
sar 867
shutacct 16
startup 16
turnacct 16
who 1219
wtmpfix 458
300 1262
4014 1264
450 1265
mv command 679-681
mvdir command 682
mvmd command 534-535
mvt command 666

N

ncheck command 683-684
ndtable command 685
neqn command 395-396
used with tbl 1053
network Users/Groups table
access 1109
newform command 686-688
newgrp command 689-690, 933
news command 691-693
next command 694-695
nice command 699-700
nl command 701-704
NLCTAB 923
nm command 705-706
node table
access 685, 801
nohup command 707-708
normal startup initialization 806
nroff command 709-720
tbl, preprocessor 1053
nulladm command 15
number factoring 416
number game 721
numbering lines 701

O

object library
ordering relation 591
octal dump 723
od command 723-725
open command 728, 729
used after actman command 32

P

pack command 730-732
packf command 733-734
pair command 996
parallel merging of lines in files 736
parameters
work station
erase 490
kill 490
logmodes 490
owner 492
parity 490
program 492
protection 492
runmodes 490, 491, 492
special purpose options 492
speed 490
parameters, setting work station 1018
terminal mapping 1023
parsing command line flags 485
passwd command 735
password table 801
password, change 735
passwords, characteristics
paste command 736-738
PATH 923
path name, return directory 95
pattern matching

  wtmpfix 458
readonly command 933
recovering from a system crash 215
red command 371
redefine keyboard 306
refile command 817-819
refusing messages 642
regcmp command 820
rejecting lines common to two sorted files 183
relational database operator 547
remcom command 510
reminder service 134
remote system mail 836
remote system, connection 198
remotely-settable hardware tabs 1041
remove a message queue 537
remove a semaphore set 537
remove a shared memory id 537
remove command 16
removing
  delta from SCCS file 837
  devices 315
  directories
    rm 833
    rmdir 838
  DOS files 345
  files
    del 308
    rm 833
    skulker 951
  groups 1129
  repeated words 1118
  users 1129
  users, SCCS 45
renaming a directory 682
renaming files 679
repairing damaged files 215
repairing wtmp records 458
repeated words, deleting 1118
repl command 821-825
report process data and system activity 1069
report, process accounting 21
reporting free disk space 318
reports
  SCCS 182
reports, accounting
restarting tlogger daemon 1071
restore command 826-831
return base file name or directory path name 95
return command 933
returning a true or false value 1099
rexd command 832
rm command 833-835, 838
rmail command 836
rmdel command 837
rmdir command 838
rmf command 839-840
rmm command 841-842
root command 997
root directory, changing 172
round command 998
rpcgen command 843
rpcinfo command 845
rsh command 935
rsprayd command 983
rstatd command 847
runacct command 848-851
runcat command 852-853
running a command at low priority 699
running commands at a later time 63
rup command 854
rusers command 856
rusersd command 858
rwall command 859
rwalld command 861

S

sact command 862
sadc command 863-864
sag command 865-866
sar command 867-870
saving files 88
sa1 command 863-864
sa2 command 863-864
scan command 871-873
scanning files 110
SCCS
  branching from leaf deltas 481

changing delta comments 152
checking the structure of SCCS files 45
comments, including kanji characters 312
creating a file 41
creating a specified version of a file 477
Data Keywords 781
delta summary 482
getting help information 513
header flags 43
identification keywords 480
initializing a file 41
interpreting errors 513
joint editing of files 482
modification request (MR) number 46, 311
Modification Requests 152
naming a file 43
recalculating the SCCS file checksum 46
removing a delta 837
reports 182
SID (SCCS Identification) 311
specifying version date cutoff 482

sccsdiff command 874
scheduling commands 63
scheduling queue requests 802
sdb command 875-882
sdiff command 883-884
search order, library 144
searching files 110
secure command 885-886
sed command 887-892
segment files 252
selecting fields from a file 269
selecting lines common to two sorted files 183
semaphore set removal 537
send command 893-896
sending messages 104, 1225
sendmail command 897-909
serial merging of lines in files 736
session records 24
set command 485, 933
set environment 393
setdma command 910
setmnt command 911-912
setting DMA channel 910
setting file-creation permission code mask 1110
setting port characteristics 490
setting tabs on a work station 1041
setting the date 281
setting the parameters for a work station 1018
sh command 913-937
See also shell
command line separators and terminators 914
SHACCT 924
shared memory id removal 537
shell 924
actman 32
blank interpretation 930
built-in commands 931
command environment 919
command execution 914
command-line substitution 917
command substitution 925
conditional substitution 920
control commands 930
diagnostic output 927
file descriptors 926
file-name substitution 916
inline input documents 928
positional parameters 918
predefined special variables
profile file 915
quoting mechanisms 925
redirection of input and output 926, 930
running the shell 935
shell variables
$- 925
$! 924
$$ 924
$? 924
$# 924
CDPATH 922
HOME 922
IFS 924
LIBPATH 922
LOGNAME 922
MAIL 922
MAILCHECK 922
MAILMSG 923
MAILPATH 922
NLCTAB 923

PATH 923
PS1 923
PS2 923
SHACCT 924
SHELL 924
TIMEOUT 924
signals 915
standard input and output 926
subshell 915
summary of redirection options 929
user-defined variables 918
variables 917
shell command 938
shell environment 491
shell procedures for accounting 13
shell variable arithmetic 412
shift command 934
shlib command 939-941
show command 942-944
show login records 15
showing
a calendar 132
accounting report 15
audit record
auditpr 73
auditselect 76
audit trail 73
compressed files 730
corresponding group names and IDs 517
corresponding user names and IDs 517
current directory 800
date 281
documents formatted with the Memorandum Macros 663
file checksum 1029
formatted files 744, 761
login name 589
news items 691
packed files 730
process accounting records 20
process status 786
profile data 773
SCCS file editing activity 862
session record 15
squeezed files 730
total accounting report 16
showmount command 945
shutacct command 16
shutdown command 946-948
shutting down the system 946
siline command 999
sin command 1000
sinstallck command 1036
size command 949-950
skulker command 951
sleep command 952-953
slocal command 954-955
sno command 956-957
sort command 958-964
sorting characters (Japanese Language Support) 959
sorting files 958
sortm command 965-966
sound command 967-968
source code
external references flow graph 154
Source Code Control System
See SCCS
special (device) files
/dev/audit 78
/dev/null
acctcom 20
standard input assigned to 20
adding 315
changing 315
creating 661
deleting 315
specifying version date cutoff 482
spell command 969-971
spellin command 970
spellprog program 971
spline command 972-973
split command 974
split files by context 252
splitting a file into pieces 974
splp command 975-977
spost command 978-980
spray command 981
squeezing files 730
standard input
acctcon1 24
acctmerg 28

as 61
at 63
awk 81
batch 63
bdiff 102
cb 139
cmp 177
comb 181
fwtmp 457
wtmpfix 458
standard output
acctcon1 24
acctmerg 28
acctwtmp 458
awk 81
bdiff 102
cal 132
cb 139
cflow 154
cmp 177
comb 181
fwtmp 457
wtmpfix 458
start accounting functions 16
starting up the system 521
startup command 16
startup initialization 806
startup shell
actman 32
stat commands 984-1016
state values
changing 361
status of interprocess communication 539
stop a process 552
stopping error-logging daemon 404
stopping tlogger daemon 1071
stream editor 887
strings 94
strip command 1017
stty command 1018-1025
su command 1026-1028
subset command 1000
sum command 1029
summarize disk usage 364
summary of command usage 18
superblock update 1030
superuser authority 770, 771, 1050, 1159
biodd_cfg 115
commands
adduser 1129
at 63
chmod 160
chroot 172
ckprereq 533
cpio 206
crontab 222
cvid 272
date 281
devices 315
dfsck 447
errdemon 398
errstop 404
fsck 446
installc 166
installp 529
killall 555
mesg 642
minidisks 650
mount 669
mvmd 534
nice 699
passwd 735
print 770
removing scheduled jobs 64
rm 833
shown in report 21
shutdown 946
tar 1049
tlog 1071
updatec 167
updatep 1122
users 1129
uustat 1159
dsipc 354
dsldxprof 355
dsstate 361
dsxlate 363
ipctable 544
ndtable 685
pwtable 801
removing files 833
ugtable 1109

suspend a process 552
sync command 1030
syntax checking, C programs 577
sysck command 1031
syslogd command 1037-1039
system activity graph 865
system activity reporter 863, 867
system files
  $HOME/.profile 1027
  /dev/null
    expr 415
  /etc/.ilog 585, 1220
  /etc/budate
    mount 90
  /etc/environment 523, 585
  /etc/filesystems 272, 446, 456, 658
    cvid 272
    mount 89, 669, 670
  /etc/group 1129
    groups 506
  /etc/magic 420
  /etc/mnttab
    mount 669
    umount 1113
  /etc/ogroup 1129
  /etc/opasswd 1129
  /etc/passwd 585, 1026, 1027, 1129
    acctdisk 27
    acctprc1 30
    groups 506
  /etc/portstatus 522, 742, 792
  /etc/profile 1027
  /etc/qconfig 754
  /etc/rasconf 398
  /usr/adm/sulog 1026
  /usr/adm/user.cfile 1129, 1131
  /usr/adm/wtmp 458, 1220
  /usr/games/lib/backrules 87
  /usr/lib/cron/at.allow 63
  /usr/lib/cron/at.deny 63
  /usr/lib/eign 795
  /usr/lib/terminfo 746
  /usr/lib/tmac 313
  a.out 61
  connect.con 198
  group 423, 689
  passwd 423, 689
  ports 741, 791, 1220
  portstatus 741, 791
  utmp 1220
system group commands
  adduser 1129
  backup 88
  biodd_cfg 115
  ckprereq 533
  cvid 272
  date 281
  devices 315
  dsipc 354
  dsldxprof 355
  dsstate 361
  dsxlate 363
  installc 166
  installp 529
  ipctable 544
  minidisks 650
  mount 669
  mvmd 534
  ndtable 685
  print 770
  pwtable 801
  turnoff 1107
  turnon 1107
  ugtable 1109
  updatec 167
  users 1129
system image examination 215
system parameters, changing or examining 171
system procedures
  restarting the terminal-logging daemon 1071
  starting the error-logging daemon 398
  starting the terminal-logging daemon 1072
  stopping the terminal-logging daemon 1071, 1072
system startup 521

T

tab command 1040
tabs command 1041-1043
tail command 1044-1046
tape archiver 1048
tapechk command 1047
tar command 1048-1052
tbl command 1053, 1055
  descriptions removed by the deroff command 313
  used in pipeline with nroff 710
tc command 1056-1057
tctl command 1058-1059
td command 461-462
tee command 1060-1061
tekset command 461
termdef command 1062
terminal characteristics 1062
terminal-logging daemon 1072
terminal mapping 1023
terminals
  DASI 300 1262
  DASI 300s 1262
  DASI 450 1265
  Diablo 1620 1265
  HP2621 514
  HP2640 514
  phototypesetter simulator 1056
  Tektronix 4014 1056, 1264
  Xerox 1700 1265
terminals, multiple virtual 32
terminating error-logging daemon 404
terminating tlogger daemon 1071
test command 934, 1064-1066
text file, changing the format of 686
text processing commands
  checkcw 275
  checkeq 395
  checkmm 663
  col 179
  cw 275
  eqn 395
  eqncheck 395
  greek 499
  mant 666
  mm 663
  mmt 666
  mt 666
  mvt 666
  neqn 395
  nroff 709
  spell 969
  tbl 1053
  troff 710
tic command 1067
time a command 1068, 1069
time command 1068
TIMEOUT 924
times command 934
timex command 1069-1070
title command 1016
tlog command 1071, 1073
tlogger command 1072-1073
tlogger daemon 1071
toc commands 1074-1076
tonls conversion routine 302, 303
tosjis conversion routine (Japanese Language Support) 302, 303
total command 1008
touch command 1077-1078
tplot command 1079-1080
tput command 1081-1082
tr command 1083-1085
trace command 1086-1090
translate characters 1083
translate profile
trap command 934
trcrpt command 1091-1092
trcstop command 1093
trcupdate command 1094-1096
trdiag command 1097-1098
troff command 710, 720
  requests removed by the deroff command 313
  tbl, preprocessor 1053
true command 1099
tsh command 1100-1101
tsort command 1102-1103
ttoc command 1075
ttt game 1104

tty command 1105-1106
turn off process accounting 16
turn on accounting functions 16
turnacct command 16
turning the computer off 946
turnoff command 1107
turnon command 1107
tvi command 1108
type command 934

U

uadm command 1133-1136
ugtable command 1109
ulimit command 934
umask command 935, 1110-1111
umount command 1112-1113
uname command 1114-1115
unget command 1116-1117
uniq command 1118
units command 1119-1121
unlink 575
unlink command 576
unmount command 1112-1113
unmounting a file system 1112
unpack command 730-732
unset command 935
untab command 1040
until command 931
update a program 1127
update groups of programs 625
updatep command 1122-1128
updating access times of a file 1077
updating modification times of a file 1077
updating the superblock 1030
usage, command summary 18
user
  adding 1129
  adm 15
  changing 1129
  deleting 1129
user IDs and names, displaying 517
user mask, modifying 1110
userid
  adm 848
  root 552, 848, 1026, 1027
users command 1129-1132
using
  display station features 332
uucheck command 1137-1138
uucico command 1139-1140
uucleanup command 1141-1143
uucp command 1144-1148
uulog command 1149-1150
uuname command 1151-1152
uupick command 1153-1155
uusched command 1156-1157
uustat command 1158-1161
uuto command 1162-1163
uutry command 1164-1165
uux command 1166-1171
uuxqt command 1172-1173

V

val command 1175-1176
var command 1008
varyoff command 1177-1179
varyon command 1180-1181
vc command 1182-1185
vcc command 141
vedit command 1187-1202
verify command 1186
version date cutoff, specifying 482
vi command 1187-1202
view command 1187-1202
virtual terminal
  assigning default display 332
  assigning physical display 332
  changing DMA pinned page 333
  setting active color palette 332
  setting background colors 332
  setting fonts 332
  setting foreground colors 332
virtual terminal manager 32
vmh command 1203-1204
vrmconfig command 1206-1207
vrmfmt command 141

vrm2rtfont command 1205
vtoc command 1075

W

wait command 935
wall command 1208
watch command 1209
wc command 1211-1212
what command 1213-1214
whatis command 511
  utility commands 511
whatnow command 1215-1218
while command 931
who command 1219-1221
whom command 1222-1224
work station characteristics 1062
work station parameters, setting 1018
work stations
  DASI 300 1262
  DASI 300s 1262
  DASI 450 1265
  Diablo 1620 1265
  HP2621 514
  HP2640 514
  phototypesetter simulator 1056
  Tektronix 4014 1056, 1264
  Xerox 1700 1265
write command 1225-1229
write operations 22
writesrv command 1230
writing audit records 80
writing buffered files to fixed disk 1030
writing the last part of a file 1044
writing to standard output
  acctconl 24
  acctmerg 28
  acctwtmp 458
  awk 81
  bdiff 102
  cal 132
  cb 139
  cflow 154
  cmp 177
  comb 181
  fwtmp 457
  wtmpfix 458
wtmpfix command 458
wump game 1231

X

X windows 1236
xargs command 1232-1235
xdbx command 1236

Y

yacc command 1237-1238
ypbind daemon 1239
ypinit command 1243
yppasswd command 1247
yppasswdd command 1249
yppoll command 1251
yppush command 1252
ypserv daemon 1256
ypwhich command 1258
ypxfr command 1260

Numerics

300 command 1262-1263
4014 command 1264
450 command 1265-1266

acctcom 22
awk 81
pattern matching (Japanese Language Support) 232
pattern, search for 501
pcat command 730-732
pd command 510
pdelay command 791-793
pdisable command 741-743
penable command 791-793
perform disk accounting functions 14
perform monthly accounting 15
performing process accounting 30
permission codes, changing 160
changing
permissions 160
permitting messages 642
pg command 744-747
phold command 741-743
pick command 748-752
pie command 1012
piobe command 753-756
pipe fitting 1060
pipeline
asynchronous execution 914
conditional execution 914
definition of 914
sequential execution 914
plot command 1014
point command 1006
port characteristics 490
port characteristics, setting 490
portmap 757
post command 758-760
power command 997
pr command 761-764
prctmp command 15
prdaily command 15, 848
precision arithmetic 97
preprocessing
source code
cc 141
cpp 210
preprocessor
macro 603
prev command 765-766
primary group 506
primary group, changing 689
prime command 986
print command 767-772
printer back end 593
printing
a calendar 132
accounting report 15
audit record
auditpr 73
auditselect 76
audit trail 73
compressed files 730
corresponding group names and IDs 517
corresponding user names and IDs 517
current directory 800
date 281
documents formatted with the Memorandum Macros 663
file checksum 1029
formatted files 744, 761
login name 589
news items 691
packed files 730
process accounting records 20
process status 786
profile data 773
SCCS file editing activity 862
session record 15
squeezed files 730
total accounting report 16
priority, running a command 699
process a report of logged errors 400
process accounting 30
process accounting records, display 20
process accounting report 21
process accounting, turn off 16
/usr/adm/wtmp
shutacct 16
process suspension 552
prod command 1006
producing C program cross-reference listing 279
prof command 144, 773-774
profiler commands 775
profiling the operating system 775

program checking, C programs 577
program maintenance 625
program update 1127
program updating 625
programming
    See also managing programs
    assembler
        as 61
    assembling source code 141
    C Language
        cb 139
        cc 140
        cflow 154
        cpp 210
        fcc 141
        formatting source code 139
        lint 577
        vcc 141
        vrmfmt 141
    debugging programs
    files
        a.out 141
    linking object files 141
    preprocessing source code 141
    the shell
programs, checking
    awk command 81
    bs 118
    cvid 272
    managing programs
        awk 81
        make 625
    messages
    miscellaneous languages
        ar 55
        awk 81
prompter command 778-779
proto command 780
prs command 781-785
prtacct command 16
ps command 786-790
pshare command 791-793
pstart command 791-793
PS1 923
PS2 923
ptecms.awk program 17
ptelus.awk program 17
ptog command 510
ptx command 794-795
puttext command 796-797
pwck command 798-799
pwd command 800, 933
pwtable command 801

Q

qdaemon command 802
qsort command 1007
query terminal characteristics 1062
quit command 510
quiz game 803-805

R

rand command 987
rank command 1007
rc command 806, 807
    startup 16
rcvdist command 808-809
rcvpack command 810-811
rcvstore command 812-814
rcvtty command 815-816
read command 933
read operations 22
reading one line 574
reading standard input
    acctcon1 24
    acctmerg 28
    as 61
    at 63
    awk 81
    batch 63
    bdiff 102
    cb 139
    cmp 177
    comb 181
    fwtmp 457