Stardent

COMMANDS
REFERENCE
MANUAL
Change History

340-0019-02 Original
340-0019-03 Software Release 2.0
340-0103-01 January, 1990

Copyright © 1990
an unpublished work of Stardent Computer Inc.
All Rights Reserved.

This document has been provided pursuant to an agreement with Stardent Computer Inc. containing restrictions on its disclosure, duplication, and use. This document contains confidential and proprietary information constituting valuable trade secrets and is protected by federal copyright law as an unpublished work. This document (or any portion thereof) may not be: (a) disclosed to third parties; (b) copied in any form except as permitted by the agreement; or (c) used for any purpose not authorized by the agreement.

Restricted Rights Legend for Agencies of the U.S. Department of Defense
Use, duplication or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at 252.227-7013 of the DoD Supplement to the Federal Acquisition Regulations. Stardent Computer Inc., 880 West Maude Avenue, Sunnyvale, California 94086.

Restricted Rights Legend for civilian agencies of the U.S. Government
Use, reproduction or disclosure is subject to restrictions set forth in subparagraph (a) through (d) of the Commercial Computer Software—Restricted Rights clause at 52.227-19 of the Federal Acquisitions Regulations and the limitations set forth in Stardent’s standard commercial agreement for this software. Unpublished—rights reserved under the copyright laws of the United States.

Starden™, Doré™, and Titan™ are trademarks of Stardent Computer Inc. UNIX® is a registered trademark of AT&T. VAX® is a registered trademark of Digital Equipment Corporation.
1. User Commands

intro(1) ................................... introduction to commands and utilities
accept(1M) .................................. allow or prevent LP requests
acctcoms(1M) ... command summary from per-process accounting records
acctcom(1) .................................. search and print process accounting file(s)
acctcon1(1M) .................................. connect-time accounting
acctcon2(1M)(acctcon1) ........................ connect-time accounting
acctdisk(1M) .................................. miscellaneous accounting commands
acctduisg(1M)(acctdisk) .................... miscellaneous accounting commands
acctmerg(1M) .................................. merge or add total accounting files
acctn(1M)(acctdisk) .......................... miscellaneous accounting commands
acctprc1(1M) .................................. process accounting
acctprc2(1M)(acctprc1) ..................... process accounting
acctwtmp(1M)(acctdisk) ..................... miscellaneous accounting commands
adapt(1) ..................................... postload an a.out file to run on a Stardent 3000
admin(1) ..................................... create and administer SCCS files
ansitape(1) ................................... ANSI-standard magtape label program
apply(1) ...................................... apply a command to a set of arguments
apropos(1) .................................... locate commands by keyword lookup
ar(1) ........................................ archive and library maintainer for portable archives
as(1) .......................................... common assembler
asa(1)(fpr) .................................... print Fortran file
at(1) .......................................... execute commands at a later time
awk(1) ........................................ pattern scanning and processing language
banner(1) ...................................... make posters
basename(1) ................................... deliver portions of path names
batch(1)(at) ................................... execute commands at a later time
bc(1) .......................................... arbitrary-precision arithmetic language
bcheckrc(1) ................................... system initialization procedures
bdiff(1) ....................................... big diff
bfs(1) ........................................ big file scanner
biff(1) ........................................ be notified if mail arrives and who it is from
cal(1) ......................................... print calendar
calendar(1) ................................... reminder service
cancel(1)(lp) .................................. send/cancel requests to an LP line printer

NOTE
Entries of the form
accton2(1M)(accton1) indicate that the command listed first is
described in the entry for the
command given in parentheses.
caption(C)info(1M) . convert a termcap description into a terminfo description
cat(1) ........................................... concatenate and print files
cat(1B) ............................................ catenate and print
cb(1) ............................................. C program beautifier
ccl(1) .............................................. C compiler
cd(1) ............................................... change working directory
cdc(1) ........................................... change the delta commentary of an SCCS delta
chargefee(1M) ................................... shell procedures for accounting
chfn(1) ............................................ change finger entry
chgrp(1)(chown) .................................. change owner or group
chmod(1) ........................................... change mode
chown(1) .......................................... change owner or group
chroot(1M) ...................................... change root directory for a command
chsh(1) ............................................ change default login shell
ckpacct(1M)(chargefee) ......................... shell procedures for accounting
clear(1) .......................................... clear terminal screen
crl(1M) ............................................ clear i-node
cmp(1) ............................................. compare two files
col(1) ............................................... filter reverse line-feeds
colcrt(1) ........................................ filter nroff output for CRT previewing
comb(1) ........................................... combine SCCS deltas
comm(1) ......................................... select or reject lines common to two sorted files
compress(1) .................................... compress and expand data
cp(1) .............................................. copy, link or move files
cp(1B) ............................................. copy
cpio(1) ........................................... copy file archives in and out
cpp(1) ............................................ the C language preprocessor
cpset(1M) ....................................... install object files in binary directories
crash(1M) ....................................... examine system images
cron(1M) ......................................... clock daemon
crontab(1) ..................................... user crontab file
crypt(1) .......................................... encode/decode
csh(1) ........................................... a shell(command interpreter) with C-like syntax
csplit(1) ........................................ context split
c(1C) ............................................ spawn getty to a remote terminal
ctags(1) .......................................... create a tags file
cu(1C) ............................................ call another UNIX system
cut(1) ............................................ cut out selected fields of each line of a file
date(1) ........................................... print and set the date
dbg(1) ............................................ debugger
dc(1) ............................................. desk calculator
dd(1M) ........................................... convert and copy a file
delta(1) .......................................... make a delta(change) to an SCCS file
deroff(1) ........................................ remove nroff/troff, tbl, and eqn constructs
devnm(1M) ...................................... device name
df(1M) ........................................... report number of free disk blocks and i-nodes
dfconf(1M) .............................................. disk farms and administration
dflist(1M)(dfconf) .............................................. disk farms and administration
dfreset(1M)(dfconf) .............................................. disk farms and administration
diff(1) ...................................................... differential file comparator
diff(1B) ....................................................... differential file and directory comparator
diff3(1) ...................................................... 3-way differential file comparison
dircmp(1) ..................................................... directory comparison
dirname(1)(basename) ........................................ deliver portions of path names
disable(1)(enable) ............................................. enable/disable LP printers
diskusg(1M) ..................................................... generate disk accounting data by user ID
dodisk(1M)(chargefee) ......................................... shell procedures for accounting
du(1M) ........................................................ summarize disk usage
dvhtool(1M) ..................................................... modify disk volume header information
echo(1) ........................................................... echo arguments
ed(1) ............................................................. text editor
ed(1) ............................................................. text editor(variant of ex for casual users)
egrep(1) .......................................................... search a file for a pattern using full regular expressions
enable(1) ....................................................... enable/disable LP printers
e(1) .............................................................. set environment for command execution
error(1) ........................................................ analyze and disperse compiler error messages
ex(1) .............................................................. text editor
expand(1) ...................................................... expand tabs to spaces, and vice versa
expr(1) ........................................................ evaluate arguments as an expression
factor(1) ....................................................... obtain the prime factors of a number
false(1)(true) ..................................................... provide truth values
fc(1) .............................................................. Fortran compiler
fgrep(1) ........................................................ search a file for a character string
file(1) ........................................................... determine file type
find(1) ........................................................ find files
finger(1) ....................................................... user information lookup program
fold(1) ......................................................... fold long lines for finite width output device
fprint(1) ....................................................... print Fortran file
from(1) ........................................................ who is my mail from?
fsc(1M) ........................................................ check and repair file systems
fsdb(1M) ....................................................... file system debugger
fsplit(1) ....................................................... split a multi-routine Fortran file into individual files
fstat(1M) ...................................................... report file system status
fstyp(1M) ..................................................... determine file system identifier
ftp(1C) ......................................................... ARPANET file transfer program
fuser(1M) ..................................................... identify processes using a file or file structure
futmp(1M) ..................................................... manipulate connect accounting records
get(1) ........................................................ get a version of an SCCS file
getopt(1) ...................................................... parse command options
getoptctv(1)(getopts) ........................................ parse command options
getopts(1) ..................................................... parse command options
getty(1M) ..................................................... set terminal type, modes, speed, and line discipline
grep(1) ........................................ search a file for a pattern
grpck(1M)(pwck) ................................ password/group file checkers
hashcheck(1)(spell) ................................ find spelling errors
hashmake(1)(spell) ................................ find spelling errors
hd(1) ......................................... hexadecimal and ASCII dump
head(1) ........................................ give first few lines
help(1) ........................................ ask for help regarding SCCS
hostid(1) ...................................... set or print identifier of current host system
hostname(1) ................................... set or print name of current host system
id(1M) ........................................ print user and group IDs and names
indent(1) ...................................... indent and format C program source
infocmp(1M) .................................. compare or print out terminfo descriptions
init(1M) ....................................... process control initialization
install(1B) .................................... install binaries
install(1M) .................................... install commands
installpkg(1) ................................ install all software distribution tapes
ipcrm(1) .. remove a message queue, semaphore set or shared memory id
ipcs(1) ........................................ report inter-process communication facilities status
join(1) ......................................... relational database operator
kill(1) ......................................... terminate a process
killall(1M) ................................... kill all active processes
labelit(1M) ................................... provide labels for file systems
last(1) ........................................ indicate last logins of users and teletypes
lastcomm(1) ................................... show last commands executed in reverse order
lastlogin(1M)(chargen) ........................ shell procedures for accounting
ld(1) .......................................... link editor for common object files
leave(1) ....................................... remind you when you have to leave
lex(1) ......................................... generate programs for simple lexical tasks
line(1) .......................................... read one line
link(1M) ...................................... link and unlink files and directories
ln(1)(cp) ...................................... copy, link or move files
localize(1) ................................... make names local to an object file
logger(1) ..................................... make entries in the system log
login(1) ....................................... sign on
logname(1) .................................... get login name
look(1) ....................................... find lines in a sorted list
lp(1) .......................................... send/cancel requests to an LP line printer
lpadmin(1M) .................................. configure the LP spooling system
lpmove(1M)(ipsched) ...... start/stop the LP scheduler and move requests
lpshut(1M)(ipsched) ...... start/stop the LP scheduler and move requests
lpstat(1) ..................................... print LP status information
ls(1) .......................................... list contents of directory
m(1) .......................................... macro processor
m68000(1)(machid) ......................... get processor type truth value
machid(1) .................................... get processor type truth value
mail(1) ........................................ send mail to users or read mail
mailx(1) .................................... interactive message processing system
make(1) ....................................... maintain, update, and regenerate groups of programs
makekey(1) ..................................... generate encryption key
man(1) ........................................ find manual information by keywords; print out the manual
mesg(1) ....................................... permit or deny messages
mips(1)(machtid) ......................... get processor type truth value
mkdir(1) ...................................... make directories
mkfarm(1M) .................................. configure a striped file system
mkfs(1M) ...................................... construct a file system
mklist(1) ..................................... PostScript label generator
mknod(1M) .................................... build special file
mkprof(1)(prof) ............................. display profile data
mkstr(1) ...................................... create an error message file by massaging C source
monacct(1M)(chargefee) ............... shell procedures for accounting
more(1) ....................................... file perusal filter for crr viewing
mount(1M) .................................... mount and unmount file systems and remote resources
mountall(1M) ................................. mount, unmount multiple file systems
mt(1) .......................................... magnetic tape manipulating program
mv(1)(cp) ..................................... copy, link or move files
mvdir(1M) .................................... move a directory
ncheck(1M) .................................... generate path names from i-numbers
nda(1M) ....................................... network daemon
netstat(1) .................................... show network status
newform(1) ................................... change the format of a text file
newgrp(1M) ................................... log in to a new group
news(1) ....................................... print news items
nice(1) ........................................ run a command at low priority
nl(1) .......................................... line numbering filter
nm(1) .......................................... print name list of common object file
nohup(1) ..................................... run a command immune to hangups and quits
nslookup(1) ................................ query name servers interactively
nulladm(1M)(chargefee) ............... shell procedures for accounting
nvram(1M) .................................... display or set values of NVRAM variables
oawk(1) ....................................... pattern scanning and processing language
od(1) .......................................... octal dump
pack(1) ....................................... compress and expand files
page(1)(more) .............................. file perusal filter for crr viewing
pagesize(1) .................................. print system page size
pal(1) ......................................... turn on/off PAL video mode
passwd(1) .................................... change login password
paste(1) ... merge same lines of several files or subsequent lines of one file
pca(1)(pack) .................................. compress and expand files
pdp11(1)(machtid) ...................... get processor type truth value
pg(1) .......................................... file perusal filter for CRTs
pled(1M) .................................... control the front panel LED
pr(1) print files
prctmp(1M)(chargefee) shell procedures for accounting
prdailv(1M)(chargefee) shell procedures for accounting
printenv(1) print out the environment
prmail(1) print out mail in the post office
prof(1) display profile data
pr(1) print an SCCS file
prtaclt(1M)(chargefee) shell procedures for accounting
ps(1) report process status
pwck(1M) password/group file checkers
pwd(1) working directory name
ratfor(1) rational Fortran dialect
rc0(1M) run commands performed to stop the operating system
rc2(1M) run commands performed for multi-user environment
rcp(1C) remote file copy
red(1)(ed) text editor
regcmp(1) regular expression compile
reject(1M)(accept) allow or prevent LP requests
reset(1)(tset) terminal dependent initialization
rev(1) reverse lines of a file
rlogin(1C) remote login
rm(1) remove files or directories
rmail(1)(mail) send mail to users or read mail
rmdev(1) remove a delta from an SCCS file
rmdir(1)(rm) remove files or directories
rs170(1) turn on/off rs170 video mode
rsh(1C) remote shell
rsh(1)(sh) shell; standard/restricted command programming language
runacct(1M) run daily accounting
runacct(1M)(chargefee) shell procedures for accounting
rusage(1C) show host status of local machines
rwho(1C) who's logged in on local machines
sa1(1M) system activity report package
sa2(1M)(sa1) system activity report package
sct(1) print current SCCS file editing activity
sact(1M)(sa1) system activity report package
sar(1) system activity reporter
scs(1) front end for the SCCS subsystem
scsdiff(1) compare two versions of an SCCS file
scrdump(1) dump the screen display
scrspt(1) make typescript of terminal session
sideby(1) side-by-side difference program
sed(1) stream editor
setmnt(1M) establish mount table
sh(1) shell, the standard/restricted command programming language
shutacct(1M)(chargefee) shell procedures for accounting
shutdown(1M) .................................. shut down system, change system state
size(1) .................................. print section sizes in bytes of common object files
sleep(1) .................................. suspend execution for an interval
sort(1) .................................. sort and/or merge files
spell(1) .................................. find spelling errors
spellin(1)(spell) .......................... find spelling errors
split(1) .................................. split a file into pieces
startup(1M)(chargefee) ............... shell procedures for accounting
strings(1) .................................. find the printable strings in an object, or other binary, file
strip(1) strip symbol/line number information from a common object file
stty(1) .................................. set the options for a terminal
stty(1B) .................................. set terminal options
su(1M) .................................. become super-user or another user
sum(1) .................................. print checksum and block count of a file
swap(1M) .................................. swap administrative interface
sync(1M) .................................. update the super block
tabs(1) .................................. set tabs on a terminal
tail(1) .................................. deliver the last part of a file
tar(1) .................................. tape file archiver
tcopy(1) .................................. copy a mag tape
tee(1) .................................. pipe fitting
telinit(1M)(init) ......................... process control initialization
telnet(1C) .................................. user interface to the TELNET protocol
test(1) .................................. condition evaluation command
tftp(1C) ............................... transfer files using DARPA Trivial File Transfer Protocol
tic(1M) .................................. terminfo compiler
time(1) .................................. time a command
timex(1) .................................. time a command; report process data and system activity
titan(1)(machid) ......................... get processor type truth value
touch(1) .................................. update access and modification times of a file
tput(1) .................................. initialize a terminal or query terminfo database
tr(1) .................................. translate characters
ttrue(1) .................................. provide truth values	set(1) .................................. terminal dependent initialization	tsort(1) .................................. topological sort	tty(1) .................................. get the name of the terminal
turnacct(1M)(chargefee) ............... shell procedures for accounting
u3b(1)(machid) .................................. get processor type truth value
u3b2(1)(machid) .................................. get processor type truth value
u3b5(1)(machid) .................................. get processor type truth value
uadmin(1M) .................................. administrative control
ul(1) .................................. do underlining
umask(1) .................................. set file-creation mode mask
umount(1M)(mount) mount/unmount file systems and remote resources
umountall(1M)(mountall) ................ mount, unmount multiple file systems
uname(1) .................................. print name of current UNIX system
uncompress(1)(compress) .................................. compress and expand data
text
unexpand(1)(expand) .................................. expand tabs to spaces, and vice versa
unget(1) ...................................................... undo a previous get of an SCCS file
unifdef(1) .................................................. remove ifdef'd lines
uniq(1) ....................................................... report repeated lines in a file
units(1) ......................................................... conversion program
unlink(1M)(link) .......................................... link and unlink files and directories
unpack(1)(pack) .......................................... compress and expand files
uptime(1) ..................................................... show how long system has been up
users(1) ...................................................... compact list of users who are on the system
uucheck(1M) ............................................. check the uucp directories and permissions file
uucico(1M) ................................................ file transport program for the uucp system
uucleanup(1M) .......................................... uucp spool directory clean-up
uucp(1C) .................................................. UNIX-to-UNIX system copy
uudecode(1C)(uencode) ........ encode/decode binary file for transmission
uencode(1C) ...... encode/decode a binary file for transmission via mail
ugets(1M) ................................................ set terminal type, modes, speed, and line discipline
uulog(1C)(uucp) ......................................... UNIX-to-UNIX system copy
uname(1C)(uucp) ........................................ UNIX-to-UNIX system copy
uuipick(1C)(uuto) ...................................... public UNIX-to-UNIX system file copy
uusched(1M) ............................................ the scheduler for the uucp file transport program
uustat(1C) ................................................ uucp status inquiry and job control
uuto(1C) .................................................. public UNIX-to-UNIX system file copy
uutry(1M) ............................................. try to contact remote system with debugging on
uuux(1C) .................................................. UNIX-to-UNIX system command execution
uuxqt(1M) ............................................... execute remote command requests
vacation(1) ............................................ return "I am on vacation" indication
val(1) ....................................................... validate SCCS file
vax(1)(machid) ........................................... get processor type truth value
vc(1) ....................................................... version control
vers(1) ..................................................... obtain version information
vgind(1) ................................................... grind nice listings of programs
vi(1) ................................................. screen-oriented(visual) display editor based on ex
w(1) ................................................. who is on and what they are doing
wait(1) ................................................... await completion of process
wall(1) ................................................... write to all users
wc(1) ...................................................... word count
what(1) ................................................... identify SCCS files
whatis(1) ................................................ describe what a command is
whereis(1) ... locate binary, source, and manual page files for a command
which(1) ................................................ locate a program file including aliases and paths
who(1) .................................................... who is on the system
whoami(1) ............................................. print effective current user id
whodo(1M) .............................................. who is doing what
write(1) ................................................ write to another user
writefix(1M)(fwtmp) ........ manipulate connect accounting records
8. System Maintenance

intro(8) ........................................... introduction to system maintenance procedures
accton(8) .................................................................. system accounting
arp(8c) ............................................................... address resolution display and control
boot(8) .................................................................. system boot procedure
catman(8) ......................................................... create the cat files for the manual
comsat(8c) .......................................................... biff server
dump(8) .............................................................. capture the state of the memory for analysis using crash(1M)
ftpd(8c) .............................................................. DARPA Internet File Transfer Protocol server
gated(8) .............................................................. gateway routing daemon
halt(8) ................................................................. stop the processor
ifconfig(8c) ........................................................ configure network interface parameters
inetd(8) ................................................................ internet “super-server”
makeindex(8) ...................................................... index manpages allowing man very_long_name to work
named(8) ............................................................. Internet domain name server
ping(8) ............................................................... send ICMP ECHO_REQUEST packets to network hosts
reboot(8) ............................................................ reboot /unix
rlogind(8c) ........................................................ remote login server
route(8c) ............................................................ manually manipulate the routing tables
routed(8c) ............................................................ network routing daemon
rshd(8c) ............................................................. remote shell server
rwhod(8c) .......................................................... system status server
sendmail(8) ........................................................ send mail over the internet
syslogd(8) ........................................................... log systems messages
telnetd(8c) ......................................................... DARPA TELNET protocol server
tftpd(8c) ........................................................... DARPA Trivial File Transfer Protocol server
zdump(8) ............................................................. time zone dumper
zic(8) .................................................................. time zone compiler
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>accept(1M)</td>
<td>reject(1M) allow or prevent LP requests</td>
</tr>
<tr>
<td>work makeindex(8)</td>
<td>index manpages allowing man very_long_name to makeindex(8)</td>
</tr>
<tr>
<td>vacation(1)</td>
<td>return I am on vacation indication</td>
</tr>
<tr>
<td>the state of the memory</td>
<td>for later analysis using crash(1M) /capture dump(8)</td>
</tr>
<tr>
<td>error messages</td>
<td>error(1) analyze and disperse compiler error(1)</td>
</tr>
<tr>
<td>sort(1) sort</td>
<td>and/or merge files sort(1)</td>
</tr>
<tr>
<td>program ansi tape(1)</td>
<td>ANSI-standard magtape label ansi tape(1)</td>
</tr>
<tr>
<td>label program</td>
<td>ansi tape(1) ANSI-standard magtape label ansi tape(1)</td>
</tr>
<tr>
<td>3000 adapt(1)</td>
<td>postload an a.out file to run on a Stardent adapt(1)</td>
</tr>
<tr>
<td>/introduction to commands,</td>
<td>application programs, and / intro(1)</td>
</tr>
<tr>
<td>arguments apply(1)</td>
<td>apply a command to a set of apply(1) apply a command to a set apply(1)</td>
</tr>
<tr>
<td>of arguments</td>
<td>apply(1) apply a command to a set apply(1)</td>
</tr>
<tr>
<td>keyword lookup</td>
<td>apropos(1) locate commands by apropos(1)</td>
</tr>
<tr>
<td>maintainer for portable</td>
<td>archives ar(1) archive and library ar(1)</td>
</tr>
<tr>
<td>archives for portable</td>
<td>ar(1) archive and library maintainer ar(1)</td>
</tr>
<tr>
<td>tar(1) tape file</td>
<td>archiver tar(1)</td>
</tr>
<tr>
<td>library maintainer for</td>
<td>portable archives ar(1) archives ar(1) archive and ar(1)</td>
</tr>
<tr>
<td>portable</td>
<td>cpio(1) copy file archives in and out cpio(1)</td>
</tr>
<tr>
<td>command xargs(1)</td>
<td>construct argument list(s) and execute xargs(1)</td>
</tr>
<tr>
<td>apply a command to a set</td>
<td>apply(1) arguments apply(1) apply(1)</td>
</tr>
<tr>
<td>of expr(1) evaluate</td>
<td>arguments as an expression expr(1)</td>
</tr>
<tr>
<td>echo(1) echo</td>
<td>arguments echo(1)</td>
</tr>
<tr>
<td>bc(1) arbitrary-precision</td>
<td>arithmetic language bc(1)</td>
</tr>
<tr>
<td>display and control</td>
<td>arp(8C) address resolution arp(8C)</td>
</tr>
<tr>
<td>biff(1) be notified if</td>
<td>mail arrives and who it is from biff(1)</td>
</tr>
<tr>
<td>mail</td>
<td>as(1) common assembler as(1)</td>
</tr>
<tr>
<td>fpr(1)</td>
<td>ask for help regarding SCCS help(1)</td>
</tr>
<tr>
<td>hd(1) hexadecimal and</td>
<td>ASCII dump hd(1)</td>
</tr>
<tr>
<td>help(1)</td>
<td>ask as(1) assembler as(1)</td>
</tr>
<tr>
<td>at(1) batch(1)</td>
<td>execute commands at(1)</td>
</tr>
<tr>
<td>at(1) batch(1)</td>
<td>execute commands at(1)</td>
</tr>
<tr>
<td>aw(1) await completion</td>
<td>of process aw(1)</td>
</tr>
<tr>
<td>processing language</td>
<td>banner(1) make posters banner(1)</td>
</tr>
<tr>
<td>(visual) display editor</td>
<td>based on ex /screen-oriented vi(1)</td>
</tr>
<tr>
<td>portions of path names</td>
<td>basename(1) dirname(1) deliver basename(1)</td>
</tr>
<tr>
<td>later time at(1)</td>
<td>batch(1) execute commands at(1)</td>
</tr>
<tr>
<td>arithmetic language</td>
<td>bc(1) arbitrary-precision bc(1)</td>
</tr>
<tr>
<td>procedures</td>
<td>bcheck(1) system initialization bcheck(1)</td>
</tr>
<tr>
<td>bdiff(1) big diff</td>
<td>bdiff(1)</td>
</tr>
<tr>
<td>cb(1) C program</td>
<td>beautifier cb(1)</td>
</tr>
<tr>
<td>su(1M)</td>
<td>become super-user or another user su(1M)</td>
</tr>
<tr>
<td>bdiff(1) big diff</td>
<td>bdiff(1)</td>
</tr>
<tr>
<td>bfs(1) big file scanner</td>
<td>bfs(1)</td>
</tr>
<tr>
<td>bfs(1)</td>
<td>bfs(1)</td>
</tr>
<tr>
<td>comsat(8C)</td>
<td>comsat(8C)</td>
</tr>
<tr>
<td>arrives and who it is</td>
<td>biff(1) be notified if mail biff(1)</td>
</tr>
<tr>
<td>from</td>
<td>bdiff(1)</td>
</tr>
<tr>
<td>install(1B) install</td>
<td>binaries install(1B)</td>
</tr>
<tr>
<td>cpset(1M) install object</td>
<td>binary directories cpset(1M)</td>
</tr>
<tr>
<td>files in an object, or</td>
<td>binary, file /find the printable strings(1)</td>
</tr>
<tr>
<td>other</td>
<td>binary, source, and manual page whereis(1)</td>
</tr>
<tr>
<td>sync(1M) update the super</td>
<td>block sync(1M)</td>
</tr>
<tr>
<td>sum(1) print checksum and</td>
<td>block count of a file sum(1)</td>
</tr>
<tr>
<td>df(1M) report number of</td>
<td>blocks and i-nodes df(1M)</td>
</tr>
<tr>
<td>boot(8) system</td>
<td>boot procedure boot(8)</td>
</tr>
</tbody>
</table>
boot(8) system boot procedure
mknod(1M)
build special file
size(1) print section sizes in
bytes of common object files
cc(1) C compiler
cpp(1) the C language preprocessor
cb(1) C program beautifier
indent(1) indent and format
C program source
xstr(1) extract strings from
C programs to implement shared/
mkstr(1) create an error message file by massaging
cal(1) print calendar
cal(1) reminder service
dc(1) desk calculator
edit(1) casual users
edi(1) cat files for the manual
cat(1B) catenate and print
for the manual
man(8) create the
cat(1B) catenate and print
man(8) create the cat files
man(8) C program beautifier
cc(1) C compiler
cd(1) change working directory
commentary of an SCCS delta
cdc(1) change the delta
chsh(1) change default login shell
chfn(1) change finger entry
passwd(1) change login password
chmmod(1) change mode
chown(1) change owner or group
chgrp(1) change group
command chroot(1M)
shutdown(1M) shut down system
SCCS delta cdc(1)
change the delta commentary of an
newform(1) change the format of a text file
delta(1) make a delta
to an SCCS file
change working directory
fgrep(1) search a file for a
character string
tr(1) translate characters
dodisk(1M) lastlogin(1M)/
fsck(1M) chargefee(1M) ckpacct(1M)
check and repair file systems
permissions file uucheck(1M)
check the uucp directories and
grpck(1M) password/group file
checkers pwck(1M)
file sum(1) print checksum and block count of a
check(1) change default login
chfn(1) change finger entry
chgrp(1) change group
change owner or group
chown(1) change owner or group
chmod(1) change mode
for a command
chroot(1M) change root directory
shell chsh(1) change default login
cr(1M) clear i-node
cli(1M) clear terminal screen
for a command
shell chroot(1M) change root directory
lastlogin(1M)/chargefee(1M)
ckpacct(1M) dodisk(1M)
clean-up uucleanup(1M)
clear(1) clear terminal screen
shell (command interpreter) with
C-like syntax csh(1) a
 cron(1M) clock daemon .............................. cron(1M)
crl(1M) clear i-node ............................ crl(1M)
cmp(1) compare two files ........................ cmp(1)
coll(1) filter reverse line-feeds ............ coll(1)

CRT previewing
colrct(1) filter roff output for ............ colrct(1)
comb(1) combine SCCS deltas .................. comb(1)
combine SCCS deltas .......................... comb(1)
comm(1) select or reject lines ............... comm(1)
command at low priority ..................... nice(1)
command chroot(1M) ......................... chroot(1M)
command execution ........................... env(1)
command immune to hangups and ............. nohup(1)
(command interpreter) with C-like ........... csh(1)
command is .................................. whatis(1)
command options ............................. getopt(1)
command options ............................. getopts(1)
command programming language .............. sh(1)
command; report process data and .......... times(1)
command ................................... test(1)
command ................................... time(1)
command requests ............................ uuxqt(1M)
command summary from per-process ........... acctmns(1M)
command to a set of arguments ............... apply(1)
command / the binary, source, ............... whereis(1)
command xargs(1) construct ................... xargs(1)
commands, application programs, .......... intro(1)
commands at a later time .................... at(1)
commands by keyword lookup .................. apropos(1)
commands executed in reverse ............... lastcomm(1)
commands / overview of accounting ......... acctdisk(1M)
commands ................................... install(1M)
commands performed for multi-user ........... rc2(1M)
commands performed to stop the ............. rc0(1M)
commentary of an SCCS delta ................. cdc1(1)
common assembler ............................. as(1)
common object file ........................... nm(1)
common object file / symbol and ............. strip(1)
common object files .......................... ld(1)
common object files size .................... size(1)
common to two sorted files ................. comm(1)
communication facilities status ............. ipcs(1)
compact list of users who are on ........... users(1)
comparator ................................... diff(1)
comparator diff(1B) ........................... diff(1B)
differential file and directory .............. descript(1M)
compare or print out terminfo ............... info(1M)
cmp(1) compare two files ..................... cmp(1)
file sccsdiff(1) compare two versions of an SCCS ........ sccsdiff(1)
diff(1) 3-way differential file ............... diff(1)
dircmp(1) directory .......................... dircmp(1)
regcmp(1) regular expression ................ regcmp(1)
cc(1) C compile ................................ compile(1)
cc(1) C compiler .............................. cc(1)
tic(1M) terminfo ............................. tic(1M)
tic(8) time zone ............................. tic(8)
error(1) analyze and disperse compiler error messages ........ error(1)
fc(1) Fortran ................................. fc(1)
yacc(1) yet another compiler-compiler ........ yacc(1)
wait() await completion of process wait()
compress() uncompress() zcat() compress and expand data compress() uncompress() zcat() compress()
pack() pcat() unpack() compress and expand files pack() compress()
cat() concatenate and print files cat()
test() condition evaluation command test()
mkfarm() parameters ifconfig() configure a striped file system mkfarm() ifconfig() configure network interface
fwtmp() manipulate wtmp()x() manipulate fwtmp() configure the LP spooling system fwpadm()
acctcon() acctcon2() handle accounting records acctcon() handle accounting files
mkfs() construct a file system mkfs()
exe() execute command xargs() construct argument list(s) and xargs()
remove() rroff() troff() eqn remove() rroff() troff() eqn
debugging() on() Utry() try to debugging() on() Utry() try to
ls() list contents of directory ls()
csplit() context split csplit()
v() version control v()
addr() display and control arp() control arp
init() telinit() process control initialization init()
led() control the front panel LED led()
name() administrative control uadmin()
in-to-a-terminfo() captinfo() convert a termcap description captinfo()
dd() convert and copy a file dd()
cp() copy cp()
dd() convert and copy a file dd()
tcory() copy copy a mag tape tcory()
cpio() copy file archives in and out cpio()
cp() ln() mv() copy, link or move files cp()
sum() count checksum and block count sum()
w() word count w()
m() move files move files
pre() copy() copy() pre() copy() copy()
pre() binary directories pre() binary directories
memory for later analysis using memory for later analysis using
ctags() create a tags file ctags()
ms() mkstr() create an error message file by mkstr()
admin() create and administer SCCS files admin()
man() catman() create the cat files for the catman()
chron() chron() chron() clock daemon chron()
cron() cron() cron() user cron file
colcrt() filter() output for CRT previewing colcrt() CRT previewing
page() file() perusal filter for crt viewing more() pag() CRTs
pg() file() perusal filter for crypt() encode/decode crypt() CRTs
interpreter() with C-like syntax csh() a shell (command csh() csh()
cy() context split csplit()
tags() create a tags file tags()
set() or print() set() or print() set() or print() host() current host system hostid() current host system
name() set() or print() name() set() or print() name()
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dir(1B) differential file and</td>
<td>diff(1B) differential file and</td>
</tr>
<tr>
<td>dir(1)</td>
<td>diff(1)</td>
</tr>
<tr>
<td>dir(1) 3-way differential file</td>
<td>diff(1) 3-way differential file</td>
</tr>
<tr>
<td>difference program</td>
<td>sdiff(1) side-by-side</td>
</tr>
<tr>
<td>dir(1) differential file and directory</td>
<td>diff(1B) differential file and directory</td>
</tr>
<tr>
<td>dir(1) differential file comparator</td>
<td>diff(1B) differential file comparator</td>
</tr>
<tr>
<td>dir(1) differential file comparison</td>
<td>diff(1) differential file comparison</td>
</tr>
<tr>
<td>dir(1)</td>
<td>dir(1)</td>
</tr>
<tr>
<td>dircmp(1) directory comparison</td>
<td>dircmp(1) directory comparison</td>
</tr>
<tr>
<td>directories</td>
<td>mknod(1)</td>
</tr>
<tr>
<td>directories</td>
<td>rm(1) remove files or</td>
</tr>
<tr>
<td>directories</td>
<td>ucheck(1M) check the uucp</td>
</tr>
<tr>
<td>directories</td>
<td>install object files in binary</td>
</tr>
<tr>
<td>directories</td>
<td>link and unlink files and</td>
</tr>
<tr>
<td>directories</td>
<td>cd(1) change working</td>
</tr>
<tr>
<td>directory clean-up</td>
<td>uucleanup(1M) uucp spool</td>
</tr>
<tr>
<td>directory comparator</td>
<td>diff(1B) differential file and</td>
</tr>
<tr>
<td>directory comparator</td>
<td>dircmp(1) directory comparator</td>
</tr>
<tr>
<td>directory comparator</td>
<td>ls(1) list contents of</td>
</tr>
<tr>
<td>directory comparator</td>
<td>chroot(1M) change root</td>
</tr>
<tr>
<td>directory comparator</td>
<td>mvdir(1M) move a</td>
</tr>
<tr>
<td>directory comparator</td>
<td>pwd(1) working</td>
</tr>
<tr>
<td>directory comparator</td>
<td>path names basename(1)</td>
</tr>
<tr>
<td>directory comparator</td>
<td>print(1) print(1)</td>
</tr>
<tr>
<td>directory comparator</td>
<td>type, modes, speed, and line</td>
</tr>
<tr>
<td>directory comparator</td>
<td>diskusg(1M) generate</td>
</tr>
<tr>
<td>directory comparator</td>
<td>df(1M) report number of free</td>
</tr>
<tr>
<td>directory comparator</td>
<td>dfconf(1M) dflist(1M) dresdt(1M)</td>
</tr>
<tr>
<td>directory comparator</td>
<td>du(1M) summarize</td>
</tr>
<tr>
<td>directory comparator</td>
<td>dvhtool(1M) modify</td>
</tr>
<tr>
<td>directory comparator</td>
<td>accounting data by user ID</td>
</tr>
<tr>
<td>directory comparator</td>
<td>error(1) analyze and</td>
</tr>
<tr>
<td>directory comparator</td>
<td>arp(8C) address resolution</td>
</tr>
<tr>
<td>directory comparator</td>
<td>vi(1) screen-oriented (visual)</td>
</tr>
<tr>
<td>directory comparator</td>
<td>variables nvr(1M)</td>
</tr>
<tr>
<td>directory comparator</td>
<td>prof(1) mkprov(1)</td>
</tr>
<tr>
<td>directory comparator</td>
<td>screendump(1) dump the screen</td>
</tr>
<tr>
<td>directory comparator</td>
<td>install all software</td>
</tr>
<tr>
<td>directory comparator</td>
<td>chargeeef(1M) ckpacct(1M)</td>
</tr>
<tr>
<td>directory comparator</td>
<td>w(1) who is on and what they are</td>
</tr>
<tr>
<td>directory comparator</td>
<td>named(8) Internet</td>
</tr>
<tr>
<td>directory comparator</td>
<td>hd(1) hexadecimal and ASCII dump</td>
</tr>
<tr>
<td>directory comparator</td>
<td>od(1) octal</td>
</tr>
<tr>
<td>directory comparator</td>
<td>screen dump</td>
</tr>
<tr>
<td>directory comparator</td>
<td>memory for later analysis using/ zdump(8) time zone</td>
</tr>
<tr>
<td>directory comparator</td>
<td>header information</td>
</tr>
<tr>
<td>directory comparator</td>
<td>hosts ping(8) send ICMP</td>
</tr>
<tr>
<td>directory comparator</td>
<td>ex for casual users</td>
</tr>
<tr>
<td>directory comparator</td>
<td>sact(1) print current SCCS file</td>
</tr>
<tr>
<td>directory comparator</td>
<td>screen-oriented (visual) display</td>
</tr>
<tr>
<td>directory comparator</td>
<td>ed(1) red(1) text</td>
</tr>
</tbody>
</table>
ex(1) text  editor ........................................ ex(1)
ld(1) link  editor for common object files .......... ld(1)
sed(1) stream  editor ................................... sed(1)
users) edit(1) text  editor (variant of ex for casual edit(1)
whoami(1) print  effective current user id whoami(1)
pattern using full regular/ enable/disable LP printers enable(1) disable(1) enable(1)
crypt(1)  enable/disable LP printers enable(1)
makekey(1) generate  encryption key makekey(1)
sccs(1) front  end for the SCCS subsystem ...... sccs(1)
logger(1) make  entries in the system log logger(1)
chfn(1) change finger  entry chfn(1)
command execution  env(1) set environment for env(1)
printenv(1) print out the environment commands performed for multi-user printenv(1)
remove nroff/troff, tbl, and eqn constructs deroff(1) deroff(1)
source mkstr(1) create an error message file by massaging C mkstr(1)
analyze and disperse compiler error messages error(1) error(1)
compiler error messages error(1) analyze and disperse error(1)
hashcheck(1) find spelling errors /hashmake(1) spellin(1) spell(1)
setmnt(1M) establish mount table setmnt(1M)
expression expr(1) evaluate arguments as an expr(1)
test(1) condition evaluation command test(1)
edit(1) text editor (variant of ex for casual users) edit(1)
(visual) display editor based on ex vi(1) screen-oriented vi(1)
ex(1) text editor ex(1)
\texttt{crash(1M)}  examine system images \texttt{crash(1M)}
construct argument list(s) and execute command xargs(1) xargs(1)
at(1) batch(1) execute commands at a later time at(1)
uxqt(1M) execute remote command requests uuxqt(1M)
lastcomm(1) show last commands executed in reverse order lastcomm(1)
set environment for command \texttt{env(1)} \texttt{env(1)}
sleep(1) suspend execution for an interval sleep(1)
zcpr(1) compress and expand data /uncompress(1) compress(1)
pcat(1) unpack(1) compress and expand files pack(1) pack(1)
versa expand(1) unexpand(1) expand tabs to spaces, and vice versa expand(1)
to spaces, and vice versa expand(1) expand(1)
expression expr(1) evaluate arguments as an expr(1)
regcmp(1) regular expression compile regcmp(1)
expr(1) evaluate arguments as an expression expression expr(1)
for a pattern using full regular expressions /search a file egrep(1)
to implement shared /xstr(1) extract strings from C programs xstr(1)
inter-process communication facilities status ipcs(1) report/ipcs(1)
factor(1) obtain the prime factor(1)
factors of a number factors of a number factor(1)
true(1) false(1) provide truth values true(1)
/dfiset(1M) dfreset(1M) disk farms and administration dfconf(1M)
f(1) Fortran compiler fc(1)
head(1) give first few lines head(1)
character string grep(1) search a file for a fgrep(1)
cut(1) cut out selected fields of each line of a file cut(1)
diff(1B) differential file and directory comparator diff(1B)
tar(1) tape file archiver tar(1)
cpio(1) copy file archives in and out cpio(1)
mkstr(1) create an error message
pwck(1M) grpck(1M) password/group
   file by massaging C source ............................ mkstr(1)
   file checkers ........................................ pwck(1M)
   file checkers ........................................ pwck(1M)
dd(1M) convert and copy a
   file .................................................. dd(1M)
mknod(1M) build special
   file .................................................. mknod(1M)
diff(1) differential
   file comparator ....................................... diff(1)
diff3(1) 3-way differential
   file comparison ...................................... diff3(1)
crontab(1) user crontab
   file .................................................. crontab(1)
ctags(1) create a tags
   file cut(1) cut out .................................. cut(1)
make a delta (change) to an SCSS
   file delta(1) ........................................ delta(1)
sact(1) print current SCSS
   file editing activity ................................ sact(1)
   file for a character string .......................... fgrep(1)
grep(1) search a
   file for a pattern ................................... grep(1)
grep(1) search a
   file for a pattern using full ........................ egrep(1)
grep(1) search a
   file for a pattern using full ........................ egrep(1)
get(1) get a version of an SCSS
   file including aliases and paths ...................... which(1)
/split a multi-runtime Fortran
   file into individual files ........................... fsplit(1)
split(1) split a
   file into pieces ..................................... split(1)
make names local to an object
   file localize(1) ..................................... localize(1)
change the format of a text
   file newform(1) ...................................... newform(1)
   file newform(1) ...................................... newform(1)
print name list of common object
   file /merge same lines of several .................... paste(1)
files or subsequent lines of one
   file or file structure fuser(1M) ...................... fuser(1M)
identify processes using a
   file or file structure fuser(1M) ...................... fuser(1M)
viewing more(1) page(1)
   file /printable strings .............................. strings(1)
   file perusal filter for CBT .......................... more(1)
   file perusal filter for CRTs ........................ pg(1)
   file perusal filter for CRTs ........................ pg(1)
   file /symbol and line number ......................... strip(1)
   file sum(1) print .................................... sum(1)
bfs(1) big
   file scanner ......................................... bfs(1)
compare two versions of an SCSS
   file scssdiff(1) ...................................... scssdiff(1)
in an object, or other binary,
   file find the printable strings ...................... strings(1)
processes using a file or
   file structure /identify ............................ fuser(1M)
information from a common object
   file /symbol and line number ......................... strip(1)
   file /symbol and line number ......................... strip(1)
   file sum(1) print .................................... sum(1)
fsdb(1M) file system debugger
   file system identifier ............................... fsdb(1M)
fstype(1M) determine
   file system ......................................... fsctype(1M)
mkfarm(1M) configure a striped
   file system .......................................... mkfarm(1M)
mkfs(1M) construct a
   file system ......................................... mkfs(1M)
fsstat(1M) report
   file system status .................................. fsstat(1M)
/umount(1M) mount and umount
   file systems and remote resources ................... mount(1M)
   file systems and remote resources ................... mount(1M)
fsck(1M) check and repair
   file systems ......................................... fsck(1M)
labelit(1M) provide labels for
   file systems ......................................... labelit(1M)
mount, umount multiple
   file systems /umountall(1M) ......................... mountall(1M)
deliver the last part of a
   file tail(1) ......................................... tail(1)
adapt(1) postload an a.out
   file to run on a Stardent 3000 ....................... adapt(1)
and modification times of a
   file touch(1) update access ........................ touch(1)
ftpd(8C) DARPA Internet File Transfer Protocol server
   file Transfer Protocol server ......................... ftpd(8C)
tftpd(8C) DARPA Trivial File Transfer Protocol server
   file Transfer Protocol server ......................... tftpd(8C)
ucp system uucico(1M) file transport program for the
   file transport program .............................. uucico(1M)
/the scheduler for the uucp
   file transport program .............................. uucpCHED(1M)
file(1) determine
   file type ............................................. file(1)
undo a previous get of an SCSS
   file unget(1) .................................... unget(1)
report repeated lines in a
   file uniq(1) ........................................ uniq(1)
val(1) validate SCSS
   file .................................................. val(1)
uucp directories and permissions
   file uucheck(1M) check the ........................ uucheck(1M)
Commands Reference Manual

Permutated Index xxiii
file systems
labelit(1M) provide labels for labelit(1M)
labels for file systems labelit(1M)
pattern scanning and processing
language awk(1) awk(1)
language bc(1) bc(1)
language oawk(1) oawk(1)
language preprocessor cpp(1)
command programming language /the standard/restricted sh(1)
executed in reverse order last(1) indicate last logins of last(1)
users and teletypes lastcomm(1) show last commands lastcomm(1)
cp(1) the C later analysis using crash(1M) dump(8)
executed in reverse order /ckpacct(1M) dodisk(1M)
executed in reverse order /lastlogin(1M) monacct(1M)/ chargefee(1M)
executed in reverse order /the state of the memory for
batch(1) execute commands at a later time at(1) at(1)
object files id(1) link editor for common id(1)
remind you when you have to leave leave(1) leave(1)
ployed(1M) control the front panel leave(1) remind you when you have leave(1)
plexical tasks LED lex(1) generate programs for plexical tasks lex(1)
generate programs for simple lex(1) lex(1)
archives ar(1) archive and line(1) read one line
line(1) read one line line(1)
terminal type, modes, speed, and line discipline getty(1M) set getty(1M)
terminal type, modes, speed, and line discipline ugetty(1M) set ugetty(1M)
common/strip(1) strip symbol and line number information from a strip(1)
nl(1) line numbering filter nl(1)
cut out selected fields of each line of a file cut(1) cut(1)
send/cancel requests to an LP line printer lp(1) cancel(1) lp(1)
line(1) read one line line(1)
col(1) filter reverse line-feeds col(1)
comm(1) select or reject lines common to two sorted files comm(1)
device fold(1) fold long lines for finite width output fold(1)
head(1) give first few lines lines head(1)
uniq(1) report repeated lines in a file uniq(1)
look(1) find lines in a sorted list look(1)
rev(1) reverse lines of a file rev(1)
of several files or subsequent lines of one file /same lines paste(1)
subsequent/paste(1) merge same lines of several files or paste(1)
unset(1) remove lidef
lines unifdef(1)
directories link(1M) unlink(1M) link and unlink files and link(1M)
files ld(1) link editor for common object ld(1)
cp(1) ln(1) mv(1) copy, link or move cp(1)
unlink files and directories link(1M) unlink(1M) link and link(1M)
ls(1) list contents of directory ls(1)
look(1) find lines in a sorted list look(1)
nm(1) print name list of common object file nm(1)
system users(1) compact list of users who are on the users(1)
vgrind(1) grind nice listings of programs vgrind(1)
xargs(1) construct argument list(s) and execute command xargs(1)
files cp(1)
localize(1) make names local to an object file localize(1)
localize(1) make names local to localize(1)
locate a program file including which(1)
locate commands by keyword lookup apropos(1)
manual page files for/whereis(1) locate the binary, source, and whereis(1)
newgrp(1M) log in to a new group newgrp(1M)
make entries in the system log logger(1) logger(1)
syslogd(8) log systems messages syslogd(8)
system log
logname(1) get
login name
password(1) change
llogin(8C) remote
login server
chsh(1) change default
login shell
login(1) sign on
last(1) indicate last
logins of users and teletypes
llogin(1) get login name
locate commands by keyword
look(1) find lines in a sorted
lookup apropos(1)
finger(1) user information
lookup program
nice(1) run a command at
low priority
send/cancel requests to an
LP line printer lp(1) cancel(1)
LP printers enable(1)
reject(1M) allow or prevent
LP requests accept(1M)
/lpmove(1M) start/stop the
LP scheduler and move requests
lpadmin(1M) configure the
LP spooling system
lpsched(1M) print the
lpstat(1) print
LP status information
lp(1) cancel(1) send/cancel
lpadmin(1M) configure the LP
lpsched(1M) start/stop the LP
Lpsched(1M) lpsched(1M)
lpshut(1M) lpmove(1M) start/stop
lpsched(1M) lpshut(1M) lpmove(1M) start/stop
lpshut(1M) lpmove(1M)
lpstat(1) print LP status
ls(1) list contents of directory
m(4) macro processor
machid(1) Titan(1) mips(1)
m68000(1) pdp11(1) u3b(1) u3b2(1)/
machid(1) titan(1) mips(1)
m4(1)
tcopy(1) copy a
mag tape
program mt(1)
magnetic tape manipulating
ansitape(1) ANSI-standard
mailtape label program
biff(1) be notified if
mail arrives and who it is from
from(1) who is my
mail from?
mail(1) print out
mail in the post office
send mail to users or read
mail(1) rm(1)
sendmail(8) send
mail over the internet
mail(1) rm(1) send mail to
mail(1) rm(1) send mail to
mail(1) interactive message
mailx(1) interactive message
maintain, update, and regenerate
make(1)
make(1) maintain, update, and regenerate
maimake(8) index
make(1) generate encryption
man(1) find manual information by
manipulate connect accounting
manipulate the routing tables
man(1) find manual information by
manipulate the routing tables
route(8C)
manpages allowing man/
manual catman(8)
manual Information by keywords,
manual page files for a command
manually manipulate the routing
route(8C)
server
deliver portions of path
ncheck(1M) generate path
print user and group IDs and
localize(1) make from i-numbers
nda(1M)
send ICMP ECHO_REQUEST packets to
ifconfig(8C) configure
routed(8C) text file
netstat(1) show
news(1) print
vgrind(1) grind
priority
object file
hangups and quits
it is from biff(1) be
colcr(1) filter
constructs deroff(1) remove
interactively
/lastlogin(1M) monacct(1M)
obtain the prime factors of a
strip(1) strip symbol and line
i-nodes df(1M) report
nl(1) line display or set values of
of NVRAM variables
processing language
make names local to an
nm(1) print name list of common
number information from a common
id(1) link editor for common
directories cpset(1M) Install
section sizes in bytes of common
/find the printable strings in an
number factor(1)
vers(1)
octal dump
print out mail in the post
pal(1) turn
rs170(1) turn
commands performed to stop the
join(1) relational database
stty(1B) set terminal
stty(1) set the
getopt(1) parse command
goptxvt(1) parse command
last commands executed in reverse
fold long lines for finite width
named(8) Internet domain name
names basename(1) dirname(1)
names from i-numbers
names id(1M)
names local to an object file
ncheck(1M) generate path names
nda(1M) network daemon
netstat(1) show network status
network daemon
network hosts ping(8)
network interface parameters
network routing daemon
news items
nice listings of programs
nice(1) run a command at low
nl(1) line numbering filter
nm(1) print name list of common
nohup(1) run a command immune to
notified if mail arrives and who
nroff output for CRT previewing
nslookup(1) query name servers
number factor(1)
number information from a common/
number of free disk blocks and
numbering filter
NVRAM variables
number of VAX registers
oaw(1) pattern scanning and
object file localize(1)
object file
object file /symbol and line
object files
object files in binary
object files size(1) print
object, or other binary, file
obtain the prime factors of a
obtain version information
od(1)

xxviii Permutated Index

Commands Reference Manual
cat(1B) catenate and
prsl print an SCSS file
date(1) print and set the date
cal(1) print calendar
sum(1) print checksum and block count of
sact(1) print current SCSS file editing
whoami(1) print effective current user id
whoami(1) print files
pr(1) print files
fpr(1) print Fortran file
asat(1) print identifier of current host
lpstat(1) print LP status information
nm(1) print name list of common object
hostname(1) print name of current host system
uname(1) print name of current UNIX system
news(1) print news items
prmail(1) print out mail in the post office
prmail(1) print out mail in the post office
inform(1M) compare or
printenv(1) print out the environment
manual information by keywords;
acctenv(1) search and
common object files
size(1) print section sizes in bytes of
pagesize(1) print system page size
Freame(1M) print user and group IDs and
id(1M) printable strings in an object;
strings(1)
printenv(1) print out the environment
printer /cancel(1) send/cancel
disable(1) enable/disable LP
printers enable(1) enable(1)
nice(1) run a command at low
priority nice(1)
post office prmail(1) print out mail in the
printenv(1) procedures
boot(8) system boot
bclock(1) system initialization
/startup(1M) turnon(1) shell
to system maintenance
acctproc(1M) acctproc(2M)
acctenv(1) search and print
init(1M) telinit(1M)
timex(1) time a command; report
kill(1) terminate a
ps(1) report
wait(1) await completion of
killall(1M) kill all active
structure fuser(1M) identify
awk(1) pattern scanning and
oawk(1) pattern scanning and
mail(1) Interactive message
halt(8) stop the
m4(1) macro
/m4b(1) u3b2(1) u3b5(1) vax(1) get
data
prof(1) mkprof(1) display
the standard/restricted command
application programs, and
/to commands, application
lex(1) generate
update, and regenerate groups of

xxx Permutated Index Commands Reference Manual
xstr(1) extract strings from C
vgrind(1) grind nice listings of
DARPA Internet File Transfer
     Protocol server ftpd(8C)  ................. ftpd(8C)
telnetd(8C) DARPA TELNET
     Protocol server telnetd(8C)
DARPA Trivial File Transfer
     labelit(1M)  provide labels for file systems labelit(1M)
true(1) false(1)  provide truth values  true(1)
/prctmp(1M) prdaily(1M)
     prs(1) print an SCCS file  prs(1)
file checkers
     practc(1M) runacct(1M)/  chargefee(1M)
     ps(1) report process status  ps(1)
pwck(1M) grpck(1M) password/group pwck(1M)
pwd(1) working directory name pwd(1)
nslookup(1)
     query name servers interactively nslookup(1)
tput(1) initialize a terminal or
memory  ipcrm(1) remove a message
     a command immune to hangups and
     queue, semaphore set or shared ipcrm(1)
dialect
     quits nohup(1) run nohup(1)
ratfor(1) rational Fortran ratfor(1)
     rational Fortran dialect ratfor(1)
rc(0(1M) run commands performed to rc(0(1M)
rc(2(1M) run commands performed rc(2(1M)
mail(1) send mail to users or
     read mail mail(1) mail(1)
line(1)  read one line line(1)
reboot(8)
     reboot /unix reboot(8)
     reboot(8) reboot /unix reboot(8)
from per-process accounting
manipulate connect accounting
     ed(1) text editor ed(1)
help(1) ask for help
     regarding SCCS help(1)
compile
     regcmp(1) regular expression regcmp(1)
make(1) maintain, update, and
     regenerate groups of programs make(1)
regcmp(1)
     regular expression compile regcmp(1)
a file for a pattern using full
files comm(1) select or
     regular expressions /search egrep(1)
requests accept(1M)
     reject lines common to two sorted reject(1M)
     join(1) relational database operator join(1)
leave(1)
     remind you when you have to leave leave(1)
calendar(1)
     reminder service calendar(1)
    uuxqt(1M) execute
    riogind(8C)
    remote command requests uuxqt(1M)
    remote login server riogind(8C)
and unmount file systems and
    remote resources /mount mount(1M)
    remote shell server riogind(8C)
Uutry(1M) try to contact
    remote system with debugging on Uutry(1M)
    remove a delta from an SCCS file rms(1)
    remove message queue, semaphore ipcrm(1)
    remove files or directories rm(1)
unifdef(1)
    remove tidedf lines unifdef(1)
constructs deroff(1)
    remove nroff/troff, tbl, and eqn deroff(1)
fsck(1M) check and
    repair file systems fsck(1M)
uniq(1) report
    repeated lines in a file uniq(1)
yes(1) be
    repetitively affirmative yes(1)
fsstat(1M) report file system status fsstat(1M)
communication facilities/ ipcs(1)
and i-nodes df(1M)
    report inter-process ipcs(1)
    report number of free disk blocks df(1M)
sa2(1M) sa3(1M) system activity
    report package sa2(1M) sa1(1M)
timex(1) time a command;
    report process data and system/ timex(1)
ps(1)
    report process status ps(1)
uniq(1) system activity
reject(1M) allow or prevent LP
the LP scheduler and move
uuxqt(1M) execute remote command
lp(1) cancel(1) send/cancel
arp(8C) address
unmount file systems and remote
indication vacation(1)
col(1) filter
rev(1) reverse line-feeds
reverse lines of a file
rev(1) reverse lines of a file
show last commands executed in
directories
read mail mail(1)
SCCS file
directories rm(1)
chroot(1M) change
routing tables
gated(8) gateway
routed(8C) network
route(8C) manually manipulate the
rs170(1) turn on/off
mode
standard/restricted/ sh(1)
nice(1)
and quits nohup(1)
multi-user environment rc2(1M)
the operating system rc0(1M)
rnact(1M)
postload an a.out file to
/prdaily(1M) prtact(1M)
activity report package sa1(1M)
report package sa1(1M)
editing activity
package sa1(1M) sa2(1M)
bfs(1) big file
awk(1) pattern
oawk(1) pattern
change the delta commentary of an
comb(1) combine
make a delta (change) to an
sact(1) print current
get(1) get a version of an
prs(1) print an
rmde(1) remove a delta from an
compare two versions of an
undo a previous get of an
val(1) validate
admin(1) create and administer
report repeated lines in a file
reporter
requests accept(1M)
requests /pmove(1M) start/stop
requests
requests
requests to an LP line printer
resolution display and control
resources /umount(1M) mount and
mount(1M)
return I am on vacation
vacation(1)
rev(1) reverse lines of a file
rev(1) reverse lines of a file
reverse order lastcomm(1)
lastcomm(1)
rlogind(8C) remote login server
rlogind(8C)
rmdir(1) remove files or
rm(1)
rm(1)
rm(1)
rmdir(1) remove a delta from an
rmdel(1)
rmdir(1) remove files or
rm(1)
root directory for a command
chroot(1M)
routed(8C) manually manipulate the
routed(8C) network routing daemon
routed(8C)
routed(8C) routing daemon
gated(8)
routing tables
gated(8)
routing tables
gated(8)
routed(8C) network
event video mode
event video mode
rsh(1) shell, the
rsh(1)
rsh(1)
run a command at low priority
nice(1)
r run commands immune to hangups
run commands performed for
run commands performed to stop
run daily accounting
runacct(1M)
run on a Stardent 3000
runacct(1M) run daily accounting
runacct(1M)
runacct(1M) shutacct(1M)/
chargees(1M)
rwhod(8C) system status server
rwhod(8C)
sa1(1M) sa2(1M) sadc(1M) system
da(1M) sa(1M) sadc(1M) system activity
sact(1) print current SCCS file
sact(1)
sadc(1M) system activity report
sa1(1M)
sar(1) system activity report
sar(1)
scanner
bfs(1)
scanning and processing language
awk(1)
scanning and processing language
oawk(1)
SCCS delta cdc(1)
SCCS delta
SCCS delta
SCCS file
SCCS file
SCCS file
SCCS file
SCCS file
SCCS file
SCCS file
SCCS file
SCCS file
SCCS file
SCCS files
SCCS files
information from a file
number information from a common
mkfarm(1M) configure a
processes using a file or file
terminal

another user
/same lines of several files or
sccs(1) front end for the SCCS
count of a file
du(1M)

accounting/acctcms(1M) command
sync(1M) update the
inetd(8) internet
su(1M) become
sleep(1)
swap(1M)
interface

information from/strip(1) strip
(command interpreter) with C-like
accton(8)
sa1(1M) sa2(1M) sadc(1M)
sar(1)
command; report process data and
boot(8)
shutdown(1M) shut down
fsdb(1M) file
uptime(1) show how long
print identifier of current host
set or print name of current host
fstyp(1M) determine file
crash(1M) examine
bcheckcr(1)

logger(1) make entries in the
configure the LP spooling
interactive message processing
intro(8) introduction to
configure a striped file
mkfs(1M) construct a file
pagesize(1) print

performed to stop the operating
shut down system, change
fsstat(1M) report file
rwhod(8C)

print name of current UNIX
list of users who are on the
who(1) who is on the
transport program for the uucp
UuTry(1M) try to contact remote
/umount(1M) mount and umount file
fsck(1M) check and repair file
provide labels for file
sylogd(8) log

mount, umount multiple file
strip symbol and line number
strip(1) strip symbol and line
strip(1) stripped file system
mkfarm(1M)
structure fuser(1M) identify
fuser(1M)
stty(1) set the options for a
stty(1)
stty(1B) set terminal options
stty(1B)
su(1M) become super-user or
su(1M)
successive lines of one file
paste(1)
subsystem
sccs(1)
syn(1) print checksum and block
sum(1)
syn(1) summarize disk usage
du(1M)

accounting/acctcms(1M) command
sync(1M) update the
inetd(8) internet
super-server
su(1M) become
super-user or another user
su(1M)
sleep(1) suspend execution for an interval
sleep(1)
snap(1M) swap administrative interface
swap(1M)
snap(1M) swap administrative
swap(1M)
symbol and line number
strip(1)
synthesis(1) update the super block
sync(1M)
synthesis(1) a shell
csh(1)
syslogd(8) log systems messages
syslogd(8)
system accounting
accton(8)
system activity report package
sa1(1M)
system activity reporter
sar(1)
system activity time(1) time a
sys(1)
system boot procedure
boot(8)
system change system state
shutdown(1M)
system debugger
fsdb(1M)
system has been up
uptime(1)
system hostid(1) set or
hostname(1)
system hostname(1)
hostname(1)
system identifier
fstyp(1M)
system images
crash(1M)
system initialization procedures
bcheckcr(1)
system log
logger(1)
system ldap(1M)
ldap(1M)
system mail(1)
mail(1)
system manpages procedures
Intro(8)
system mknfs(1M)
mknfs(1M)
system
mkfs(1M)
system page size
pagesize(1)
system rc(1M) run commands
rc(1M)
system state shutdown(1M)
shutdown(1M)
system status
fsstat(1M)
system status server
rwhod(8C)
system uname(1)
uname(1)
system users(1)
compact
users(1)
system
system
transport program for the uucp
UuTry(1M) try to contact remote
/umount(1M) mount and umount file
fsck(1M) check and repair file
provide labels for file
syslogd(8) log
systems
systems /umountall(1M)
mountall(1M)
the scheduler for the uucp file
server tftp(8C) DARPA
values u3b5(1) vax(1) get processor type
true(1) false(1) provide true
try to contact remote system with........uutty(1M)
initialization
true(1) false(1) provide
debugging on
Uutty(1M)
tset,reset(1) terminal dependent
tset,reset(1)
tsort(1) topological sort
 tty(1) get the name of the...
tty(1)
pal(1) turn on/off PAL video mode
pal(1)
rs170(1) turn on/off rs170 video mode
rs170(1)
accou /shutacct(1M) startup(1M)
determine file
file(1) type
getty(1M) set terminal
type, modes, speed, and line/.
ugetty(1M) set terminal
type, modes, speed, and line/
ugetty(1M)
u3b5(1) vax(1) get processor
type truth value /u3b(1) u3b2(1)
machid(1)
script(1) make
typescript of terminal session
/mips(1) m68000(1) pdp11(1)
u3b(1) u3b2(1) u3b5(1) vax(1) get/
machid(1)
/mips(1) m68000(1) pdp11(1) u3b(1)
u3b2(1) u3b5(1) vax(1) get/
machid(1)
/m68000(1) pdp11(1) u3b(1) u3b2(1)
u3b5(1) vax(1) get processor type/
machid(1)
uadmnt(1M) administrative control
uadmnt(1M)
ul(1) do underlining
ul(1)
machid(1)
unumask(1) set file-creation mode
mount(1M) mount and unmount file
mountall(1M) mount, unmount
mount(1M)
UNIX system
and expand data compress(1)
uncompress(1) zcat(1) compress
underlining
ul(1)
file unget(1)
do a previous get of an SCCS
Spaces, and vice versa expand(1)
unexpand(1) expand tabs to
an SCCS file
unget(1) undo a previous get of
a file
unifdef(1) remove ifdef lines
unix(1) report repeated lines in
unix(1) conversion program
reboot(8) reboot
/unix
uname(1) print name of current
UNIX system
link(1M) unlink(1M) link and
unlink files and directories
link(1M)
and directories link(1M)
mount(1M) umount(1M) mount and
umount file systems and remote/
mount(1M)
mountall(1M) mount, unmount
multiple file systems
files pack(1) pacat(1)
times of a file touch(1)
timeprograms make(1) maintain,
sync(1M)
has been up
du(1M) summarize disk
id(1M) print
user and group IDs and names
id(1M)
crontab(1)
user crontab file
whoami(1) print effective current
user id
whoami(1)
generate disk accounting data by
diskusage(1M)
user ID diskusage(1M)
user information lookup program
finger(1)
write(1) write to another
user
write(1)
become super-user or another
user
su(1M)
last(1) indicate last logins of
users and teletypes
users(1)
edit(1) text
e dit(1)
users or read mail
mail(1)
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cd(1)</td>
<td>change working directory</td>
</tr>
<tr>
<td>pwd(1)</td>
<td>working directory name</td>
</tr>
<tr>
<td>wall(1)</td>
<td>write to all users</td>
</tr>
<tr>
<td>write(1)</td>
<td>write to another user</td>
</tr>
<tr>
<td></td>
<td>write(1) write to another user</td>
</tr>
<tr>
<td>accounting records fwtmp(1M)</td>
<td>manipulate connect fwtmp(1M)</td>
</tr>
<tr>
<td>list(s) and execute command</td>
<td>xargs(1) construct argument</td>
</tr>
<tr>
<td></td>
<td>xstr(1) extract strings from C</td>
</tr>
<tr>
<td>compiler-compiler affirmative</td>
<td>yacc(1) yet another yacc(1)</td>
</tr>
<tr>
<td></td>
<td>yes(1) be repetitively yes(1)</td>
</tr>
<tr>
<td></td>
<td>yet another compiler yacc(1)</td>
</tr>
<tr>
<td></td>
<td>zcat(1) compress and expand data zcat(1)</td>
</tr>
<tr>
<td></td>
<td>zdump(8) time zone dumper zdump(8)</td>
</tr>
<tr>
<td></td>
<td>zic(8) time zone compiler zic(8)</td>
</tr>
<tr>
<td></td>
<td>zic(8) time zone compiler zic(8)</td>
</tr>
<tr>
<td></td>
<td>zone compiler zic(8)</td>
</tr>
<tr>
<td></td>
<td>zone dumper zdump(8)</td>
</tr>
</tbody>
</table>
**NAME**
intro – introduction to commands, application programs, and programming utilities

**DESCRIPTION**
This section describes, in alphabetical order, commands available for the Stardent 1500/3000 single user supercomputer. Certain distinctions of purpose are made in the headings.

The following Utility packages are delivered with the Stardent 1500/3000:

- Advanced C Utilities
- Basic Networking Utilities
- C Programming Language Utilities
- Directory and File Management Utilities
- Editing Utilities
- Essential Utilities
- Extended Software Generation System Utilities
- Graphics Utilities
- Help Utilities
- Inter-process Communications
- Line Printer Spooling Utilities
- Networking Support Utilities
- Performance Measurement Utilities
- Remote File Sharing Utilities
- Security Administration Utilities
- Software Generation System Utilities
- Source Code Control System Utilities
- Spell Utilities
- Terminal Filters Utilities
- Terminal Information Utilities
- User Environment Utilities
- Windowing Utilities

The following Utility Packages are available options:

- Documenter’s Workbench
- Fortran Programming Utilities
- Network File System (NFS) Utilities

In addition to the System V Release 3 commands that form the basis of the Stardent 1500/3000 operating system, some additional 4.3 BSD commands are available to duplicate the functionality of 4.3 BSD. These commands are: cat(1B), cp(1B), diff(1B), install(1B), and stty(1B). You can access these commands by including /usr/ucb as the first element in your search path.

**Manual Page Command Syntax**
Unless otherwise noted, commands described in the SYNOPIX section of a manual page accept options and other arguments according to the following syntax and should be interpreted as explained below.

```
name [-option...] [cmdarg...]
```

where:

```
[]                     Surround an option or cmdarg that is not required.
...
Indicates multiple occurrences of the option or cmdarg.
```
name

The name of an executable file.

option

(Always preceded by a "--").
noargletter ... or, argletter
optarg[,...]

noargletter
A single letter representing an option without an
option-argument. Note that more than one noargletter
option can be grouped after one "--" (Rule 5, below).

argletter
A single letter representing an option requiring an
option-argument.

optarg
An option-argument (character string) satisfying a
preceding argletter. Note that groups of optargs follow-
ing an argletter must be separated by commas, or
separated by white space and quoted (Rule 8, below).

cmdarg
Path name (or other command argument) not beginning with "--", or
"--" by itself indicating the standard input. Throughout the manual
pages there are references to TMPDIR, BINDIR, INCDIR, LIBDIR, and
LLIBDIR. These represent directory names whose value is specified
on each manual page as necessary. For example, TMPDIR might refer
to /tmp or /usr/tmp. These are not environment variables and cannot
be set. [There is also an environment variable called TMPDIR which
can be set. See setenv(3).]

Command Syntax Standard: Rules

These command syntax rules are not followed by all current commands, but all new
commands will obey them. getoptis(1) should be used by all shell procedures to parse
positional parameters and to check for legal options. It supports Rules 3-10 below.
The enforcement of the other rules must be done by the command itself.

1. Command names (name above) must be between two and nine characters
long.

2. Command names must include only lower-case letters and digits.

3. Option names (option above) must be one character long.

4. All options must be preceded by "--".

5. Options with no arguments may be grouped after a single "--".

6. The first option-argument (optarg above) following an option must be pre-
ceded by white space.

7. Option-arguments cannot be optional.

8. Groups of option-arguments following an option must either be
separated by commas or separated by white space and quoted (e.g., "-o
xxx, z, yy or "-o "xxx z yy")

9. All options must precede operands (cmdarg above) on the command line.

10. "--" may be used to indicate the end of the options.

11. The order of the options relative to one another should not matter.

12. The relative order of the operands (cmdarg above) may affect their
significance in ways determined by the command with which they appear.
13. "-" preceded and followed by white space should only be used to mean standard input.

SEE ALSO

getopts(1), exit(2), wait(2), getopt(3C)

DIAGNOSTICS

Upon termination, each command returns two bytes of status, one supplied by the system and giving the cause for termination, and (in the case of "normal" termination) one supplied by the program [see wait(2) and exit(2)]. The former byte is 0 for normal termination; the latter is customarily 0 for successful execution and non-zero to indicate troubles such as erroneous parameters, or bad or inaccessible data. It is called variously "exit code", "exit status", or "return code", and is described only where special conventions are involved.

WARNINGS

Some commands produce unexpected results when processing files containing null characters. These commands often treat text input lines as strings and therefore become confused upon encountering a null character (the string terminator) within a line.
NAME
accept, reject – allow or prevent LP requests

SYNOPSIS
/usr/lib/accept destinations
/usr/lib/reject [-r reason] destinations

DESCRIPTION
accept allows *lp*(1) to accept requests for the named *destinations*. A *destination* can be either a line printer (LP) or a class of printers. Use *lpstat*(1) to find the status of *destinations*.

Reject prevents *lp*(1) from accepting requests for the named *destinations*. A *destination* can be either a printer or a class of printers. Use *lpstat*(1) to find the status of *destinations*. The following option is useful with reject.

-\[ reason \] Associates a *reason* with preventing *lp* from accepting requests. This *reason* applies to all printers mentioned up to the next -r option. *Reason* is reported by *lp* when users direct requests to the named *destinations* and by *lpstat*(1). If the -r option is not present or the -r option is given without a *reason*, then a default *reason* will be used.

FILES
/usr/spool/lp/*

SEE ALSO
lpadmin(1M), lp sched(1M).
enable(1), *lp*(1), *lpstat*(1)
NAME

acctdisk, acctdusg, accton, acctwtmp – overview of accounting and miscellaneous accounting commands

SYNOPSIS

/usr/lib/acct/acctdisk
/usr/lib/acct/acctdusg [-u file] [-p file]
/usr/lib/acct/accton [file]
/usr/lib/acct/acctwtmp "reason"

DESCRIPTION

Accounting software is structured as a set of tools (consisting of both C programs and shell procedures) that can be used to build accounting systems. acctsh(1M) describes the set of shell procedures built on top of the C programs.

Connect time accounting is handled by various programs that write records into /etc/utmp, as described in utmp(4). The programs described in accton(1M) convert this file into session and charging records, which are then summarized by acctmerg(1M).

Process accounting is performed by the UNIX system kernel. Upon termination of a process, one record per process is written to a file (normally /usr/adm/pacct). The programs in acctprc(1M) summarize this data for charging purposes; acctcns(1M) is used to summarize command usage. Current process data may be examined using acctcom(1).

Process accounting and connect time accounting (or any accounting records in the format described in acct(4)) can be merged and summarized into total accounting records by acctmerg (see tacct format in acct(4)). prlacct (see acctsh(1M)) is used to format any or all accounting records.

acctdisk reads lines that contain user ID, login name, and number of disk blocks and converts them to total accounting records that can be merged with other accounting records.

acctdusg reads its standard input (usually from find / -print) and computes disk resource consumption (including indirect blocks) by login. If -u is given, records consisting of those filenames for which acctdusg charges no one are placed in file (a potential source for finding users trying to avoid disk charges). If -p is given, file is the name of the password file. This option is not needed if the password file is /etc/passwd. (See diskusg(1M) for more details.)

accton alone turns process accounting off. If file is given, it must be the name of an existing file, to which the kernel appends process accounting records (see acct(2) and acct(4)).

acctwtmp writes a utmp(4) record to its standard output. The record contains the current time and a string of characters that describe the reason. A record type of ACCOUNTING is assigned (see utmp(4)). reason must be a string of 11 or fewer characters, numbers, $, or spaces. For example, the following are suggestions for use in reboot and shutdown procedures, respectively:

```plaintext
acctwtmp uname >> /etc/wtmp
acctwtmp "file save" >> /etc/wtmp
```

FILES

/etc/passwd used for login name to user ID conversions
/usr/lib/acct holds all accounting commands listed in sub-class 1M of this manual
Process Accounting Utilities

ACCT(1M)

/usr/adm/pacct current process accounting file
/etc/wtmp login/logoff history file

SEE ALSO

acct(2), acct(4), acctcms(1M), acctcom(1) acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), diskusg(1M), fwtmp(1M), runacct(1M), utmp(4)
NAME
acctcms – command summary from per-process accounting records

SYNOPSIS
/usr/lib/acct/acctcms [options] files

DESCRIPTION
acctcms reads one or more files, normally in the form described in acct(4). It adds all records for processes that executed identically-named commands, sorts them, and writes them to the standard output, normally using an internal summary format. The options are:

- **a**: Print output in ASCII rather than in the internal summary format. The output includes command name, number of times executed, total kcore-minutes, total CPU minutes, total real minutes, mean size (in K), mean CPU minutes per invocation, "hog factor", characters transferred, and blocks read and written, as in acctcom(1). Output is normally sorted by total kcore-minutes.
- **c**: Sort by total CPU time, rather than total kcore-minutes.
- **j**: Combine all commands invoked only once under "*other".
- **n**: Sort by number of command invocations.
- **s**: Any filenames encountered hereafter are already in internal summary format.
- **t**: Process all records as total accounting records. The default internal summary format splits each field into prime and non-prime time parts. This option combines the prime and non-prime time parts into a single field that is the total of both, and provides upward compatibility with old (i.e., UNIX System V) style acctcms internal summary format records.

The following options may be used only with the –a option.

- **p**: Output a prime-time-only command summary.
- **o**: Output a non-prime (offshift) time only command summary.

When –p and –o are used together, a combination prime and non-prime time report is produced. All the output summaries will be total usage except number of times executed, CPU minutes, and real minutes, which will be split into prime and non-prime.

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
```

SEE ALSO
acct(1M), acct(2), acct(4), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M), utmp(4)

BUGS
Unpredictable output results if –t is used on new style internal summary format files, or if it is not used with old style internal summary format files.
NAME
acctcom – search and print process accounting file(s)

SYNOPSIS
acctcom [[options][file]]

DESCRIPTION
acctcom reads file, the standard input, or /usr/adm/pacct, in the form described by acct(4) and writes selected records to the standard output. Each record represents the execution of one process. The output shows the COMMAND NAME, USER, TTYNAME, START TIME, END TIME, REAL (SEC), CPU (SEC), MEAN SIZE(K), and optionally, F (the fork/exec flag: 1 for fork without exec), STAT (the system exit status), HOG FACTOR, KCORE MIN, CPU FACTOR, CHARs TRNSFD, and BLOCKS READ (total blocks read and written).

A # is prepended to the command name if the command was executed with superuser privileges. If a process is not associated with a known terminal, a ? is printed in the TTYNAME field.

If no files are specified, and if the standard input is associated with a terminal or /dev/null (as is the case when using & in the shell), /usr/adm/pacct is read; otherwise, the standard input is read.

If any file arguments are given, they are read in their respective order. Each file is normally read forward, i.e., in chronological order by process completion time. The file /usr/adm/pacct is usually the current file to be examined; a busy system may need several such files of which all but the current file are found in /usr/adm/pacct? The options are:

-a Show some average statistics about the processes selected. The statistics will be printed after the output records.
-b Read backwards, showing latest commands first. This option has no effect when the standard input is read.
-f Print the fork/exec flag and system exit status columns in the output. The numeric output for this option will be in octal.
-h Instead of mean memory size, show the fraction of total available CPU time consumed by the process during its execution. This "hog factor" is computed as:

(total CPU time)/(elapsed time).
-i Print columns containing the I/O counts in the output.
-k Instead of memory size, show total kcore-minutes.
-m Show mean core size (the default).
-r Show CPU factor (user time/(system-time + user-time)).
-t Show separate system and user CPU times.
-v Exclude column headings from the output.
-l line Show only processes belonging to terminal /dev/line.
-u user Show only processes belonging to user that may be specified by: a user ID, a login name that is then converted to a user ID, a #, which designates only those processes executed with superuser privileges, or ?, which designates only those processes associated with unknown user IDs.
-g group Show only processes belonging to group. The group may be designated by either the group ID or group name.
-s time Select processes existing at or after time, given in the format hr[:min[:sec]].
-e time Select processes existing at or before time.
-S time       Select processes starting at or after time.
-E time       Select processes ending at or before time. Using the same time for both
              -S and -E shows the processes that existed at time.
-n pattern    Show only commands matching pattern that may be a regular expres-
              sion as in ed(1) except that + means one or more occurrences.
-q            Do not print any output records, just print the average statistics as with
              the -a option.
-o ofile      Copy selected process records in the input data format to ofile; supress
              standard output printing.
-H factor     Show only processes that exceed factor, where factor is the "hog factor"
              as explained in option -h above.
-O sec        Show only processes with CPU system time exceeding sec seconds.
-C sec        Show only processes with total CPU time, system plus user, exceeding
              sec seconds.
-I chars      Show only processes transferring more characters than the cutoff
              number given by chars.

FILES
   /etc/passwd
   /usr/adm/pacct
   /etc/group

SEE ALSO
   acct(1M), acct(2), acct(4), acctcms(1M), acctcon(1M), acctmerg(1M), acctprc(1M),
   acctsh(1M), fwtmp(1M), ps(1), runacct(1M), su(1), utmp(4)

BUGS
   acctcom reports only on processes that have terminated; use ps(1) for active processes.
   If time exceeds the present time, then time is interpreted as occurring on the previous
day.
NAME

acctcon1, acctcon2 – connect-time accounting

SYNOPSIS

/usr/lib/acct/acctcon1 [options]
/usr/lib/acct/acctcon2

DESCRIPTION

acetcon1 converts a sequence of login/logoff records read from its standard input to a sequence of records, one per login session. Its input should normally be redirected from /etc/wtmp. Its output is ASCII, giving device, user ID, login name, prime connect time (seconds), non-prime connect time (seconds), session starting time (numeric), and starting date and time. The options are:

-p Print input only, showing line name, login name, and time (in both numeric and date/time formats).
-t acctcon1 maintains a list of lines on which users are logged in. When it reaches the end of its input, it emits a session record for each line 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. The -t flag causes it to use, instead, the last time found in its input, thus assuring reasonable and repeatable numbers for non-current files.

-l file file is created to contain a summary of line usage showing login name, number of minutes used, percentage of total elapsed time used, number of sessions charged, number of logins, and number of logoffs. This file helps track line usage, identify bad lines, and find software and hardware oddities. Hangup, termination of login(1) and termination of the login shell each generate logoff records, so that the number of logoffs is often three times the number of sessions. See init(1M) and utmp(4).

-o file file is filled with an overall record for the accounting period, giving starting time, ending time, number of reboots, and number of date changes.

acctcon2 expects as input a sequence of login session records and converts them into total accounting records (see tac format in acct(4)).

EXAMPLES

These commands are typically used as shown below. The file ctmp is created only for the use of acctprc(1M) commands:

acctcon1 -t -l lineuse -o reboots <wtmp | sort +ln +2 > ctmp
acctcon2 <ctmp | acctmerg > ctacct

FILES

/etc/wtmp

SEE ALSO

acct(1M), acct(2), acct(4), acctcms(1M), acctcom(1), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), init(1M), login(1), runacct(1M), utmp(4)

BUGS

The line usage report is confused by date changes. Use wtmpfix (see fwtmp(1M)) to correct this situation.
NAME
acctmerg – merge or add total accounting files

SYNOPSIS
/usr/lib/acct/acctmerg [options] [file] . . .

DESCRIPTION
acctmerg reads its standard input and up to nine additional files, all in the tacct format (see acct(4)) or an ASCII version thereof. It merges these inputs by adding records whose keys (normally user ID and name) are identical, and expects the inputs to be sorted on those keys. Options are:

- `a` Produce output in ASCII version of tacct.
- `i` Input files are in ASCII version of tacct.
- `p` Print input with no processing.
- `t` Produce a single record that totals all input.
- `u` Summarize by user ID, rather than user ID and name.
- `v` Produce output in verbose ASCII format, with more precise notation for floating-point numbers.

EXAMPLES
The following sequence is useful for making "repairs" to any file kept in this format:

```bash
acctmerg -v <file1 > file2
```

Edit file2 as desired . . .

```bash
acctmerg -i <file2 > file1
```

SEE ALSO
acct(1M), acct(2), acct(4), acctcms(1M), acctcom(1), acctcon(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M), utmp(4)
NAME
acctprc1, acctprc2 – process accounting

SYNOPSIS
/usr/lib/acct/acctprc1 [ctmp]
/usr/lib/acct/acctprc2

DESCRIPTION
acctprc1 reads input in the form described by acct(4), adds login names corresponding to user IDs, then writes for each process an ASCII line giving user ID, login name, prime CPU time (tics), non-prime CPU time (tics), and mean memory size (in memory segment units). If ctmp is given, it is expected to contain a list of login sessions, in the form described in acctcon(1M), sorted by user ID and login name. If this file is not supplied, it obtains login names from the password file. The information in ctmp helps it distinguish among different login names that share the same user ID.

acctprc2 reads records in the form written by acctprc1, summarizes them by user ID and name, then writes the sorted summaries to the standard output as total accounting records.

These commands are typically used as shown below:
acctprc1 ctmp </usr/adm/pacct | acctprc2 >ptacct

FILES
/etc/passwd

SEE ALSO
acct(1M), acct(2), acct(4), acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctsh(1M), cron(1M), fwtmp(1M), runacct(1M), utmp(4)

BUGS
Although it is possible to distinguish among login names that share user IDs for commands run normally, it is difficult to do this for those commands run from cron(1M), for example. More precise conversion can be done by faking login sessions on the console via the acctutmp program in acct(1M).

CAVEAT
A memory segment of the mean memory size is a unit of measure for the number of bytes in a logical memory segment on a particular processor.
NAME
chargefee, ckpacct, dodisk, lastlogin, monacct, nulladm, prctmp, prdaily, prtaacct, runacct, shutacct, startup, turnacct – shell procedures for accounting

SYNOPSIS
/usr/lib/acct/chargefee login-name number
/usr/lib/acct/ckpacct [blocks]
/usr/lib/acct/dodisk [-o] [files ...]
/usr/lib/acct/lastlogin
/usr/lib/acct/monacct number
/usr/lib/acct/nulladm file
/usr/lib/acct/prctmp
/usr/lib/acct/prdaily [-l] [-c] [ mmd ]
/usr/lib/acct/prtaacct file [ "heading"]
/usr/lib/acct/runacct [mmd] [mmd state]
/usr/lib/acct/shutacct ["reason"]
/usr/lib/acct/startup
/usr/lib/acct/turnacct on | off | switch

DESCRIPTION
chargefee can be invoked to charge a number of units to login-name. A record is written to /usr/adm/fee, to be merged with other accounting records during the night.
ckpacct should be initiated via cron(1M). It periodically checks the size of /usr/adm/pacct. If the size exceeds blocks, 1000 by default, turnacct will be invoked with argument switch. If the number of free disk blocks in the /usr file system falls below 500, ckpacct will automatically turn off the collection of process accounting records via the off argument to turnacct. When at least this number of blocks is restored, the accounting will be activated again. This feature is sensitive to the frequency at which ckpacct is executed, usually by cron.
dodisk should be invoked by cron to perform the disk accounting functions. By default, it will do disk accounting on the special files in /etc/fstab. If the -o flag is used, it will do a slower version of disk accounting by login directory. files specifies the one or more filesystem names where disk accounting will be done. If files are used, disk accounting will be done on these filesystems only. If the -o flag is used, files should be mount points of mounted filesystems. If omitted, they should be the special file names of mountable filesystems.
lastlogin is invoked by runacct to update /usr/adm/acct/sum/loginlog, which shows the last date on which each person logged in.
monacct should be invoked once each month or each accounting period. number indicates which month or period it is. If number is not given, it defaults to the current month (01–12). This default is useful if monacct is to executed via cron(1M) on the first day of each month. monacct creates summary files in /usr/adm/acct/fiscal and restarts summary files in /usr/adm/acct/sum.
nulladm creates file with mode 664 and ensures that owner and group are adm. It is called by various accounting shell procedures.
prctmp can be used to print the session record file (normally /usr/adm/acct/nlte/ctmp created by acccton(1M).
prdaily is invoked by runacct to format a report of the previous day’s accounting data. The report resides in /usr/adm/acct/sum/rprt mmmdd where mmmdd is the month and day of the report. The current daily accounting reports may be printed by typing prdaily. Previous days’ accounting reports can be printed by using the mmmdd option and specifying the exact report date desired. The –1 flag prints a report of exceptional usage by login id for the specified date. Previous daily reports are cleaned up and therefore inaccessible after each invocation of monacct. The –c flag prints a report of exceptional resource usage by command, and may be used on current day’s accounting data only.

prtacct can be used to format and print any total accounting (tacct) file.

runacct performs the accumulation of connect, process, fee, and disk accounting on a daily basis. It also creates summaries of command usage. For more information, see runacct(1M).

shutacct is invoked during a system shutdown to turn process accounting off and append a "reason" record to /etc/utmp.

startup can be called to turn the accounting on when the system is brought to a multi-user state.

turnacct is an interface to accton (see acct(1M)) to turn process accounting on or off. The switch argument turns accounting off, moves the current /usr/adm/pacct to the next free name in /usr/adm/pacctincr (where incr is a number starting with 1 and incrementing by one for each additional pacct file), then turns accounting back on again. This procedure is called by ckpacct and thus can be taken care of by the cron and used to keep pacct to a reasonable size. acct starts and stops process accounting via init and shutdown accordingly.

FILES

/usr/adm/fee accumulator for fees
/usr/adm/pacct current file for per-process accounting
/usr/adm/pacct* used if pacct gets large and during
execution of daily accounting procedure
/etc/wtmp login/logoff summary
/usr/lib/acct/ptelus.awk contains the limits for exceptional
usage by login id
/usr/lib/acct/ptecms.awk contains the limits for exceptional
usage by command name
/usr/adm/acct/nite working directory
/usr/lib/acct holds all accounting commands listed in
sub-class 1M of this manual
/usr/adm/acct/sum summary directory, should be saved

SEE ALSO

acct(1M), acct(2), acct(4), acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), cron(1M), diskusg(1M), fwtmp(1M), runacct(1M), utmp(4)
NAME
adapt – postload an a.out file to run on a Stardent 3000

SYNOPSIS
adapt [ -v ] [ infile [ outfile ] ]

DESCRIPTION
adapt takes as input an executable file for a Stardent 1500 system and produces an executable file for the Stardent 3000. The resultant output file is functionally identical to the input file. infile is the name of the input object file compiled for Stardent 1500 execution. outfile is optional. If no name is specified for the output file, a file named x.out is produced.

OPTIONS
-v causes adapt to provide diagnostic information about the postloading process. With this option, whenever an instruction gets changed by the postloader, the address in the input file where the change occurred and the new instruction inserted are listed on stderr.
NAME
admin - create and administer SCCS files

SYNOPSIS
admin [-n] [-i[name]] [-r[rel]] [-t[name]] [-fflag[flag-val]] [-dflag[flag-val]] [-allogin]
[-e[login]] [-m[mrlist]] [-y[ycomment]] [-h] [-z] files

DESCRIPTION
admin is used to create new SCCS files and change parameters of existing ones. Arguments to admin, which may appear in any order, consist of keyletter arguments, which begin with -, and named files (note that SCCS file names must begin with the character s.). If a named file does not exist, it is created, and its parameters are initialized according to the specified keyletter arguments. Parameters not initialized by a keyletter argument are assigned a default value. If a named file does exist, parameters corresponding to specified keyletter arguments are changed, and other parameters are left as is.

If a directory is named, admin behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.) and unreadable files are silently ignored. If a name of - is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The keyletter arguments are as follows. Each is explained as though only one named file is to be processed since the effects of the arguments apply independently to each named file.

-n
This keyletter indicates that a new SCCS file is to be created.

-i[name]
The name of a file from which the text for a new SCCS file is to be taken. The text constitutes the first delta of the file (see -r keyletter for delta numbering scheme). If the i keyletter is used, but the file name is omitted, the text is obtained by reading the standard input until an end-of-file is encountered. If this keyletter is omitted, then the SCCS file is created empty. Only one SCCS file may be created by an admin command on which the i keyletter is supplied. Using a single admin to create two or more SCCS files requires that they be created empty (no -i keyletter). Note that the -i keyletter implies the -n keyletter.

-r[rel]
The release into which the initial delta is inserted. This keyletter may be used only if the -i keyletter is also used. If the -r keyletter is not used, the initial delta is inserted into release 1. The level of the initial delta is always 1 (by default initial deltas are named 1.1).

-t[name]
The name of a file from which descriptive text for the SCCS file is to be taken. If the -t keyletter is used and admin is creating a new SCCS file (the -n and/or -i keyletters also used), the descriptive text file name must also be supplied. In the case of existing SCCS files: (1) a -t keyletter without a file name causes removal of descriptive text (if any) currently in the SCCS file, and (2) a -t keyletter with a file name causes text (if any) in the named file to replace the descriptive text (if any) currently in the SCCS file.
-fflag
This keyletter specifies a flag, and, possibly, a value for the flag, to be placed in the SCCS file. Several f keyletters may be supplied on a single admin command line. The allowable flags and their values are:

b Allows use of the -b keyletter on a get(1) command to create branch deltas.

cceil The highest release (i.e., "ceiling"), a number greater than 0 but less than or equal to 9999, which may be retrieved by a get(1) command for editing. The default value for an unspecified c flag is 9999.

ffloor The lowest release (i.e., "floor"), a number greater than 0 but less than 9999, which may be retrieved by a get(1) command for editing. The default value for an unspecified f flag is 1.

dSID The default delta number (SID) to be used by a get(1) command.

i[itr] Causes the "No id keywords (get6)" message issued by get(1) or delta(1) to be treated as a fatal error. In the absence of this flag, the message is only a warning. The message is issued if no SCCS identification keywords [see get(1)] are found in the text retrieved or stored in the SCCS file. If a value is supplied, the keywords must exactly match the given string, however the string must contain a keyword, and no embedded newlines.

j Allows concurrent get(1) commands for editing on the same SID of an SCCS file. This allows multiple concurrent updates to the same version of the SCCS file.

llist A list of releases to which deltas can no longer be made (get -e against one of these "locked" releases fails). The list has the following syntax:

<list> ::= <range> | <list>, <range>
<range> ::= a  | 

The character a in the list is equivalent to specifying all releases for the named SCCS file.

n Causes delta(1) to create a "null" delta in each of those releases (if any) being skipped when a delta is made in a new release (e.g., in making delta 5.1 after delta 2.7, releases 3 and 4 are skipped). These null deltas serve as "anchor points" so that branch deltas may later be created from them. The absence of this flag causes skipped releases to be non-existent in the SCCS file, preventing branch deltas from being created from them in the future.

qtext User definable text substituted for all occurrences of the %Q% keyword in SCCS file text retrieved by get(1).

mmod Mod ule name of the SCCS file substituted for all occurrences of the %M% keyword in SCCS file text retrieved by get(1). If the m flag is not specified, the value assigned is the name of the SCCS file with the leading s. removed.

ttype Type of module in the SCCS file substituted for all occurrences of %Y% keyword in SCCS file text retrieved by get(1).
Causes *delta*(1) to prompt for Modification Request (MR) numbers as the reason for creating a delta. The optional value specifies the name of an MR number validity checking program [see *delta*(1)]. (If this flag is set when creating an SCCS file, the m keyletter must also be used even if its value is null).

### -d flag

Causes removal (deletion) of the specified flag from an SCCS file. The -d keyletter may be specified only when processing existing SCCS files. Several -d keyletters may be supplied on a single *admin* command. See the -f keyletter for allowable flag names.

### list

A list of releases to be "unlocked". See the -f keyletter for a description of the I flag and the syntax of a list.

### -a login

A login name, or numerical UNIX system group ID, to be added to the list of users which may make deltas (changes) to the SCCS file. A group ID is equivalent to specifying all login names common to that group ID. Several a keyletters may be used on a single *admin* command line. As many logins, or numerical group IDs, as desired may be on the list simultaneously. If the list of users is empty, then anyone may add deltas. If login or group ID is preceded by a ! they are to be denied permission to make deltas.

### -e login

A login name, or numerical group ID, to be erased from the list of users allowed to make deltas (changes) to the SCCS file. Specifying a group ID is equivalent to specifying all login names common to that group ID. Several e keyletters may be used on a single *admin* command line.

### -m [mrlist]

The list of Modification Requests (MR) numbers is inserted into the SCCS file as the reason for creating the initial delta in a manner identical to *delta*(1). The v flag must be set and the MR numbers are validated if the v flag has a value (the name of an MR number validation program). Diagnostics will occur if the v flag is not set or MR validation fails.

### -y [comment]

The comment text is inserted into the SCCS file as a comment for the initial delta in a manner identical to that of *delta*(1). Omission of the -y keyletter results in a default comment line being inserted in the form:

- The date and time created YY/MM/DD HH:MM:SS by login
- The -y keyletter is valid only if the -i and/or -n keyletters are specified (i.e., a new SCCS file is being created).

### -h

Causes *admin* to check the structure of the SCCS file [see *sccsfile*(5)], and to compare a newly computed check-sum (the sum of all the characters in the SCCS file except those in the first line) with the check-sum that is stored in the first line of the SCCS file. Appropriate error diagnostics are produced. This keyletter inhibits writing on the file, so that it nullifies the effect of any other keyletters supplied, and is, therefore, only meaningful when processing existing files.

### -z

The SCCS file check-sum is recomputed and stored in the first line of the SCCS file (see -h, above).
Note that use of this keyletter on a truly corrupted file may prevent future detection of the corruption.

The last component of all SCCS file names must be of the form s.file-name. New SCCS files are given mode 444 [see chmod(1)]. Write permission in the pertinent directory is, of course, required to create a file. All writing done by admin is to a temporary x-file, called x.file-name, [see get(1)], created with mode 444 if the admin command is creating a new SCCS file, or with the same mode as the SCCS file if it exists. After successful execution of admin, the SCCS file is removed (if it exists), and the x-file is renamed with the name of the SCCS file. This ensures that changes are made to the SCCS file only if no errors occurred.

It is recommended that directories containing SCCS files be mode 755 and that SCCS files themselves be mode 444. The mode of the directories allows only the owner to modify SCCS files contained in the directories. The mode of the SCCS files prevents any modification at all except by SCCS commands.

If it should be necessary to patch an SCCS file for any reason, the mode may be changed to 644 by the owner allowing use of ed(1). Care must be taken! The edited file should always be processed by an admin -h to check for corruption followed by an admin -z to generate a proper check-sum. Another admin -h is recommended to ensure the SCCS file is valid.

admin also makes use of a transient lock file (called z.file-name), which is used to prevent simultaneous updates to the SCCS file by different users. See get(1) for further information.

**FILES**

g-file Existed before the execution of delta; removed after completion of delta.
p-file Existed before the execution of delta; may exist after completion of delta.
q-file Created during the execution of delta; removed after completion of delta.
x-file Created during the execution of delta; renamed to SCCS file after completion of delta.
z-file Created during the execution of delta; removed during the execution of delta.
d-file Created during the execution of delta; removed after completion of delta.

/usr/bin/bdiff Program to compute differences between the “gotten” file and the g-file.

**SEE ALSO**
delta(1), ed(1), get(1), help(1), prs(1), what(1), sccsfile(4).

**DIAGNOSTICS**

Use help(1) for explanations.
NAME
ansitape – ANSI-standard magtape label program

SYNOPSIS
ansitape txr[vqfaei3] [mt=devicel]
[vo=volume-name] [rs=[rl recordsize]] [bs=blocksize]
[rf=[v1f]] [cc=[ilflcl]]
filename1 filename2...

DESCRIPTION
ansitape reads, writes, and creates magtapes conforming to the ANSI standard for magtape labelling. Primarily, this is useful to exchange tapes with VAX/VMS, which makes this kind of tape by default.

ansitape is controlled by a function key letter (t, x, c, or r). Various options modify the format of the output tape.

Writing ANSI Tapes
The list of files on the command line is written to the tape. A full Unix pathname may be specified, however, only the last pathname component (everything after the last /) is used as the filename on the tape.

Normally, regular text files are to be exchanged. ansitape reads the files one line at a time and transfers them to the tape. The newline character at the end of each line is removed, and the file is written in a variable-length record format. Variable-format files have the length of the longest record specified in a file header. Therefore, ansitape will read each input file from disk before it goes on to tape, to determine the maximum record size. The read is skipped if the file is more than 100,000 bytes long. The default carriage control (implied) instructs the other host to restore the newline character before printing the record.

If ansitape thinks that the input file is a Unix text file (Fortran or implied carriage control), it will automatically strip the the Unix newline from the end of each record. No strip is done with embedded carriage control files, or with any file using a fixed-length record format.

For binary files, fixed-length records should be used. VAX/VMS normally uses a record length of 512 bytes for things like directories and executable files, but data files may have any record length. Binary files should be flagged for embedded (rf=e) carriage control.

Reading ANSI Tapes
When reading, the input file list is presumed to be the names of files to be extracted from the tape. The shell wildcard characters asterisk (*) and question-mark (?) may be used. Of course, they must be quoted to prevent the shell from interpreting them before ansitape sees them.

None of the options for record format or carriage control need be specified when reading files. ansitape will automatically pick up this information from the header records on the tape, and do the right thing. If you can’t get just what you want from ansitape, the resulting files may be run through dd(1).

FUNCTION LETTERS
These function letters describe the overall operation desired. One of them must be specified in the first argument to ansitape. For lexically rigorous Unix fans, a minus sign (-) is allowed, but optional, to introduce the first keyword option set.
r Write the named files on the end of the tape. This requires that the tape have been previously initialized with an ANSI volume header.

c Create a new magtape. The tape is initialized with a new ANSI volume header. All files previously on the tape are destroyed. This option implies r.

x Extract all files from the tape. Files are placed in the current directory. Protection is r/w to everyone, modified by the current umask(2).

t List all of the names on the tape.

**MODIFIER KEY LETTERS**

These key letters are part of the first argument to anstape.

v Normally anstape does its work silently; the v (verbose) option displays the name of each file anstape treats, preceded by the function letter. It also displays the volume name of each tape as it is mounted. When used with the t option, anstape displays the number of tape blocks used by each file, the record format, and the carriage control option.

q Query before writing anything. On write (c or r options), this causes anstape to ask before writing to the tape. On extract operations, anstape displays the Unix pathname, and asks if it should extract the file. Any response starting with a 'y' or 'Y' means yes, any other response (including an empty line) means no.

f File I/O is done to standard i/o instead. For example, when writing a tape file that is to contain a lint listing, we could specify

```
lint xyz.c l anstape rf xyz.lint
```

instead of

```
lint xyz.c > /tmp/xyz.lint
anstape r /tmp/xyz.lint
rm /tmp/xyz.lint
```

When reading, this option causes the extracted files to be sent to stdout instead of a disk file.

a The tape should be read or written with the ASCII character set. This is the default.

e The tape should be written with the EBCDIC character set. The mapping is the same one used by the dd(1) program with conv=ebcdic. This option is automatically enabled if IBM-format labels are selected.

i Use IBM-format tape labels. The IBM format is very similar, but not identical, to the ANSI standard. The major difference is that the tape will contain no HDR3 or HDR4 records, thus restricting the name of the files on the tape to 17 characters. This option automatically selects the EBCDIC character set for output. To make an IBM-format label on a tape using the ASCII character set (why?), use the option sequence ia.

3 Do not write HDR3 or HDR4 labels. The HDR3 label is reserved for the use of the operating system that created the file. HDR4 is for overflow of filenames that are longer than the 17 characters allocated in the HDR1 label. Not all systems process these labels correctly, or even ignore them correctly. This switch suppresses the HDR3 and HDR4 labels when the tape is to be transferred to a system that would choke on them.
FUNCTION MODIFIERS

Each of these options should be given as a separate argument to anstape. Multiple options may be specified. They must appear as after the key-letter options above, and before any filename arguments.

mt=device
Select an alternate drive on which the tape is mounted. The default is /dev/rmt8.

vo=volume-name
Specify the name of the output volume. Normally, this defaults to the first six characters of your login name. The string 'UNIX' is used as the default if anstape cannot determine your login name.

rs=recordsize
Specify the output recordsize in bytes. This is the maximum size in the case of variable-format files. This option also turns on the fixed-record-format option. Thus, if you want to have variable record sizes with a smaller maximum, you must specify

rs=recordsize rf=v

When the recordsize is manually given, anstape does not read disk files to determine the maximum record length.

rs=r
This is a variant of the rs= option. This causes anstape to read all disk files for recordsize, regardless of their size. Normally, files larger than 100K bytes are not scanned for recordsize. Using this option also implies variable-length records.

bs=blocksize
Specify the output blocksize, in bytes. As many records as will fit are crammed into each physical tape block. ANSI standards limit this to 2048 bytes (the default), but you may specify more or less. Be advised that specifying more may prevent some systems from reading the tape.

rf=v
Record format is variable-length. In other words, they are text files. This is the default, and should be left alone unless you really know what you're doing.

rf=f
Record format is fixed-length. This is usually a bad choice, and should be reserved for binary files. This also turns off the newline strip usually done for Unix text files.

cc=i
Carriage control implied (default). Unlike Unix text files, where records are delimited by a newline character, ANSI files do not normally include the newline as part of the record. Instead, a newline is automatically added to the record whenever it is sent to a printing device.

cc=f
Carriage control Fortran. Each line is expected to start with a Fortran carriage-control character. anstape does not insert these characters automatically, it merely marks the file as having them. This is of limited usefulness. (Good opportunity for another ambitious hacker.)

cc=e
Carriage control is embedded. Carriage control characters (if any) are a part of the data records. This is usually used in the case of binary data files.

SEE ALSO

dd(1), umask(2), mtio(4)
AUTHOR

David S. Hayes, Site Manager, US Army Artificial Intelligence Center. Originally
developed June 1986. Revised August 1986. This software is in the public domain.

BUGS

The r (write) option cannot be used with quarter-inch archive tapes, since these tape
drives cannot backspace.

There is no way to ask for the n-th occurrence of a file.

Tape errors are handled ungracefully.

Files with names longer than 80 characters have the name truncated. This is a limita-
tion of the ANSI labelling standard. If the tape is made without HDR3 and HDR4
labels (3 or i switch), the name is limited to 17 characters.

Multi-volume tape sets cannot yet be generated. anstape will read them just fine, but
it won’t write them. Unix provides no device-independent way to detect a physical
down-of-tape. It was decided that a 2400-foot limitation was preferrable to device-
dependence.
NAME
apply – apply a command to a set of arguments

SYNOPSIS
apply [-ac] [-n ] command args ...

DESCRIPTION
Apply runs the named command on each argument arg in turn. Normally arguments
are chosen singly; the optional number n specifies the number of arguments to be
passed to command. If n is zero, command is run without arguments once for each arg.
Character sequences of the form %d in command, where d is a digit from 1 to 9, are
replaced by the d’th following unused arg. If any such sequences occur, n is ignored,
and the number of arguments passed to command is the maximum value of d in com-
mand. The character ‘%’ may be changed by the –a option.
Examples:
    apply echo *
is similar to ls(1);
    apply –2 cmp a1 b1 a2 b2 ...
comparing the ‘a’ files to the ‘b’ files;
    apply –0 who 1 2 3 4 5
runs who(1) 5 times; and
    apply ‘ln %1 /usr/joe’ *
links all files in the current directory to the directory /usr/joe.

SEE ALSO
sh(1)

AUTHOR
Rob Pike

BUGS
Shell metacharacters in command may have bizarre effects; it is best to enclose compli-
cated commands in single quotes ‘’.

There is no way to pass a literal ‘%2’ if ‘%’ is the argument expansion character.
NAME

apropos – locate commands by keyword lookup

SYNOPSIS

apropos keyword ...

DESCRIPTION

apropos shows which manual sections contain instances of any of the given keywords in their title. Each word is considered separately and case of letters is ignored. Words which are part of other words are considered; thus, when looking for compile, apropos will find all instances of ‘compiler’ also. Try

    apropos password

and

    apropos editor

If the line starts ‘name(section) ...’ you can do ‘man section name’ to get the documentation for it. Try ‘apropos format’ and then ‘man 3s printf’ to get the manual on the subroutine printf.

apropos is actually just the –k option to the man(1) command.

FILES

    /usr/man/whatis            data base

SEE ALSO

    man(1), whatis(1), catman(8)

AUTHOR

    William Joy
DIRECTORY AND FILE MANAGEMENT UTILITIES

NAME

ar – archive and library maintainer for portable archives

SYNOPSIS

ar key [posname] afile [name] ...

DESCRIPTION

The ar command maintains groups of files combined into a single archive file. Its main use is to create and update library files as used by the link editor. It can be used, though, for any similar purpose. The magic string and the file headers used by ar consist of printable ASCII characters. If an archive is composed of printable files, the entire archive is printable.

When ar creates an archive, it creates headers in a format that is portable across all machines. The portable archive format and structure is described in detail in ar(4). The archive symbol table [described in ar(4)] is used by the link editor [ld(1)] to effect multiple passes over libraries of object files in an efficient manner. An archive symbol table may only be created and maintained by ar when the archive contains only object files. The archive symbol table is in a specially named file which is always the first file in the archive. This file is never mentioned or accessible to the user. The s option described below is used to force the symbol table to be rebuilt.

Unlike command options, the command key is a required part of ar’s command line. The key (which may begin with a −) is formed with one of the following letters: d q t p m v r x. Arguments to the key, alternatively, are made with one of more of the following set: vuaibcls. Posname is an archive member name used as a reference point in positioning other files in the archive. Afile is the archive file. The names are constituent files in the archive file. The meanings of the key characters are as follows:

d Delete the named files from the archive file.

r Replace the named files in the archive file. If the optional character u is used with r, then only those files with dates of modification later than the archive files are replaced. If an optional positioning character from the set ab1 is used, then the posname argument must be present and specifies that new files are to be placed after (a) or before (b or l) posname. Otherwise new files are placed at the end.

q Quickly append the named files to the end of the archive file. Optional positioning characters are invalid. The command does not check whether the added members are already in the archive. This option is useful to avoid quadratic behavior when creating a large archive piece-by-piece.

t Print a table of contents of the archive file. If no names are given, all files in the archive are tabled. If names are given, only those files are tabled.

p Print the named files in the archive.

m Move the named files to the end of the archive. If a positioning character is present, then the posname argument must be present and, as in r, specifies where the files are to be moved.

x Extract the named files. If no names are given, all files in the archive are extracted. In neither case does x alter the archive file.

The meanings of the key arguments are as follows:

v Give a verbose file-by-file description of the making of a new archive file from the old archive and the constituent files. When used with t, give a long listing of all information about the files. When used with x, precede each file with a name.
c  Suppress the message that is produced by default when *afile* is created.

l  Place temporary files in the local (current working) directory rather than in the default temporary directory, *TMPDIR*.

s  Force the regeneration of the archive symbol table even if *ar(1)* is not invoked with a command which will modify the archive contents.

**FILES**

*TMPDIR/*

Temporary files

*TMPDIR* is usually /usr/tmp but can be redefined by setting the environment variable *TMPDIR* [see *tempnam()* in *tmpnam(3S)*].

**SEE ALSO**


**NOTES**

If the same file is mentioned twice in an argument list, it may be put in the archive twice.
NAME

as – common assembler

SYNOPSIS

as [options] [input] output

DESCRIPTION

Note: This program differs from most UNIX assemblers because it may be used as a filter.

The as command assembles the named file. The following flags may be specified in any order:

-i filename
   Specifies a name for the input filename. (However, the input is still stdin.)

-o objfile
   Put the output of the assembly in objfile. By default, the output file name is
   formed by removing the .s suffix, if there is one, from the input file name
   specified with the -i option and appending a .o suffix. If there is no -i
   option, the default output file is named a.out.

-S
   Produce on the standard output a disassembled version of the input.

-V
   Write the version number of the assembler being run on the standard error output.

SEE ALSO

cc(1), ld(1), nm(1), strip(1), tmpnam(3S), a.out(4)

NOTES

Note: Writing assembly code that correctly uses the floating point or vector units involves subtle issues of synchronization that are best left to the compiler. Use of this option is strongly discouraged.

Wherever possible, the assembler should be accessed through a compilation system interface program such as cc(1). In this case, the C preprocessor is run, giving a rudimentary macro and include capability.
NAME

at, batch – execute commands at a later time

SYNOPSIS

at time [ date ] [ + increment ]
at -r job...
at -l [ job ... ]
batch

DESCRIPTION

at and batch read commands from standard input to be executed at a later time. at allows you to specify when the commands should be executed, while jobs queued with batch will execute when system load level permits. at may be used with the following options:

-r  Removes jobs previously scheduled with at.
-l  Reports all jobs scheduled for the invoking user.

Standard output and standard error output are mailed to the user unless they are redirected elsewhere. The shell environment variables, current directory, umask, and ulimit are retained when the commands are executed. Open file descriptors, traps, and priority are lost.

Users are permitted to use at if their name appears in the file /usr/lib/cron/at.allow. If that file does not exist, the file /usr/lib/cron/at.deny is checked to determine if the user should be denied access to at. If neither file exists, only root is allowed to submit a job. If at.deny is empty, global usage is permitted. The allow/deny files consist of one user name per line. These files can only be modified by the superuser.

The time may be specified as 1, 2, or 4 digits. One and two digit numbers are taken to be hours, four digits to be hours and minutes. The time may alternately be specified as two numbers separated by a colon, meaning hour:minute. A suffix am or pm may be appended; otherwise a 24-hour clock time is understood. The suffix zulu may be used to indicate GMT. The special names noon, midnight, now, and next are also recognized.

An optional date may be specified as either a month name followed by a day number (and possibly year number preceded by an optional comma) or a day of the week (fully spelled or abbreviated to three characters). Two special “days”, today and tomorrow are recognized. If no date is given, today is assumed if the given hour is greater than the current hour and tomorrow is assumed if it is less. If the given month is less than the current month (and no year is given), next year is assumed.

The optional increment is simply a number suffixed by one of the following: minutes, hours, days, weeks, months, or years. (The singular form is also accepted.)

Thus legitimate commands include:

  at 0815am Jan 24
  at 8:15am Jan 24
  at now + 1 day
  at 5 pm Friday

at and batch write the job number and schedule time to standard error.

batch submits a batch job. It is almost equivalent to “at now”, but not quite. For one, it goes into a different queue. For another, “at now” will respond with the error message too late.
at -r removes jobs previously scheduled by at or batch. The job number is the number given to you previously by the at or batch command. You can also get job numbers by typing at -l. You can only remove your own jobs unless you are the super-user.

**EXAMPLES**

The at and batch commands read from standard input the commands to be executed at a later time. sh(1) provides different ways of specifying standard input. Within your commands, it may be useful to redirect standard output.

This sequence can be used at a terminal:

```
batch
sort filename > outfile
<control-D> (hold down 'control' and depress 'D')
```

This sequence, which demonstrates redirecting standard error to a pipe, is useful in a shell procedure (the sequence of output redirection specifications is significant):

```
batch <<!
sort filename 2>&1 > outfile | mail loginid !
```

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 1900 thursday next week
```

**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/lib/cron/queue scheduling information
/usr/spool/cron/atjobsspool area
```

**SEE ALSO**

cron(1M), kill(1), mail(1), nice(1), ps(1), sh(1), sort(1), queuedefs(4)

**DIAGNOSTICS**

Complains about various syntax errors and times out of range.
NAME
awk – pattern scanning and processing language

SYNOPSIS
awk [-F re] [parameter...] ['prog'] [-f progfile] [file...]

DESCRIPTION
awk is a new version of awk that provides capabilities unavailable in previous versions. This version will become the default version of awk in the next major UNIX system release.

The -F re option defines the input field separator to be the regular expression re.

Parameters, in the form x=... y=... may be passed to awk, where x and y are awk built-in variables (see list below).

awk scans each input file for lines that match any of a set of patterns specified in prog. The prog string must be enclosed in single quotes (') to protect it from the shell. For each pattern in prog there may be an associated action performed when a line of a file matches the pattern. The set of pattern-action statements may appear literally as prog or in a file specified with the -f progfile option.

Input files are read in order; if there are no files, the standard input is read. The file name – means the standard input. Each input line is matched against the pattern portion of every pattern-action statement; the associated action is performed for each matched pattern.

An input line is normally made up of fields separated by white space. (This default can be changed by using the FS built-in variable or the -F re option.) The fields are denoted $1, $2, ...; $0 refers to the entire line.

A pattern-action statement has the form:

    pattern { action }

Either pattern or action may be omitted. If there is no action with a pattern, the matching line is printed. If there is no pattern with an action, the action is performed on every input line.

Patterns are arbitrary Boolean combinations ( !, ||, &&, and parentheses) of relational expressions and regular expressions. A relational expression is one of the following:

expression relop expression
expression matchop regular expression

where a relop is any of the six relational operators in C, and a matchop is either ~ (contains) or !~ (does not contain). A conditional is an arithmetic expression, a relational expression, the special expression

var in array,

or a Boolean combination of these.

The special patterns BEGIN and END may be used to capture control before the first input line has been read and after the last input line has been read respectively.

Regular expressions are as in egrep [see grep(1)]. In patterns they must be surrounded by slashes. Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions. A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines between an occurrence of the first pattern and the next occurrence of the second pattern.
A regular expression may be used to separate fields by using the -F re option or by assigning the expression to the built-in variable FS. The default is to ignore leading blanks and to separate fields by blanks and/or tab characters. However, if FS is assigned a value, leading blanks are no longer ignored.

Other built-in variables include:

- **ARGC** command line argument count
- **ARGV** command line argument array
- **FILENAME** name of the current input file
- **FNR** ordinal number of the current record in the current file
- **FS** input field separator regular expression (default blank)
- **NF** number of fields in the current record
- **NR** ordinal number of the current record
- **OFMT** output format for numbers (default %g)
- **OFS** output field separator (default blank)
- **ORS** output record separator (default new-line)
- **RS** input record separator (default new-line)

An action is a sequence of statements. A statement may be one of the following:

- `if ( conditional ) statement [ else statement ]`
- `while ( conditional ) statement`
- `do statement while ( conditional )`
- `for ( expression ; conditional ; expression ) statement`
- `for ( var in array ) statement`
- `delete array[subscript]`
- `break`
- `continue`
- `[ [ statement ] ... ]`
- `expression` # commonly variable = expression
- `print [ expression-list ] [ >expression ]`
- `printf format [ , expression-list ] [ >expression ]`
- `next` # skip remaining patterns on this input line
- `exit [ expr ]` # skip the rest of the input; exit status is expr
- `return [ expr ]`

Statements are terminated by semicolons, new-lines, or both. An empty expression-list stands for the whole input line. Expressions take on string or numeric values as appropriate, and are built using the operators +, −, *, /, %, and concatenation (indicated by a blank). The C operators ++, −−, +=, −=, *=, /=, and %= are also available in expressions. Variables may be scalars, array elements (denoted x[i]), or fields. Variables are initialized to the null string or zero. Array subscripts may be any string, not necessarily numeric; this allows for a form of associative memory. String constants are quoted (").

The print statement prints its arguments on the standard output, or on a file if >expression is present, or on a pipe if | cmd is present. The arguments are separated by the current output field separator and terminated by the output record separator. The printf statement formats its expression list according to the format [see printf(3S) in the Programmer's Reference Manual].
awk has a variety of built-in functions: arithmetic, string, input/output, and general.
The arithmetic functions are: \texttt{atan2}, \texttt{cos}, \texttt{exp}, \texttt{int}, \texttt{log}, \texttt{rand}, \texttt{sin}, \texttt{sqrt}, and \texttt{svrand}. \texttt{int} truncates its argument to an integer. \texttt{rand} returns a random number between 0 and 1.\texttt{svrand (expr)} sets the seed value for \texttt{rand} to \texttt{expr} or uses the time of \texttt{day} if \texttt{expr} is omitted.
The string functions are:
\begin{itemize}
\item \texttt{gsub(for, repl, in)}
\hspace{1cm} behaves like \texttt{sub} (see below), except that it replaces successive occurrences of the regular expression (like the \texttt{ed} global substitute command).
\item \texttt{index(s, t)}
\hspace{1cm} returns the position in string \texttt{s} where string \texttt{t} first occurs, or 0 if it does not occur at all.
\item \texttt{length(s)}
\hspace{1cm} returns the length of its argument taken as a string, or of the whole line if there is no argument.
\item \texttt{match(s, re)}
\hspace{1cm} returns the position in string \texttt{s} where the regular expression \texttt{re} occurs, or 0 if it does not occur at all. \texttt{RSTART} is set to the starting position (which is the same as the returned value), and \texttt{RLENGTH} is set to the length of the matched string.
\item \texttt{split(s, a, fs)}
\hspace{1cm} splits the string \texttt{s} into array elements \texttt{a[1]}, \texttt{a[2]}, \texttt{a[n]}, and returns \texttt{n}. The separation is done with the regular expression \texttt{fs} or with the field separator \texttt{FS} if \texttt{fs} is not given.
\item \texttt{strftime(fmt, expr, expr, \ldots)}
\hspace{1cm} formats the expressions according to the \texttt{printf(3S)} format given by \texttt{fmt} and returns the resulting string.
\item \texttt{gsub(for, repl, in)}
\hspace{1cm} substitutes the string \texttt{repl} in place of the first instance of the regular expression \texttt{for} in string \texttt{in} and returns the number of substitutions. If \texttt{in} is omitted, \texttt{awk} substitutes in the current record ($\$0$).
\item \texttt{substr(s, m, n)}
\hspace{1cm} returns the \texttt{n}-character substring of \texttt{s} that begins at position \texttt{m}.
\end{itemize}
The input/output and general functions are:
\begin{itemize}
\item \texttt{close(filename)}
\hspace{1cm} closes the file or pipe named \texttt{filename}.
\item \texttt{cmd | getline}
\hspace{1cm} pipes the output of \texttt{cmd} into \texttt{getline}; each successive call to \texttt{getline} returns the next line of output from \texttt{cmd}.
\item \texttt{getline}
\hspace{1cm} sets $\$0$ to the next input record from the current input file.
\item \texttt{getline <file}
\hspace{1cm} sets $\$0$ to the next record from \texttt{file}.
\item \texttt{getline var}
\hspace{1cm} sets variable \texttt{var} instead.
\item \texttt{getline var <file}
\hspace{1cm} sets \texttt{var} from the next record of \texttt{file}.
\item \texttt{system(cmd)}
\hspace{1cm} executes \texttt{cmd} and returns its exit status.
\end{itemize}
All forms of \texttt{getline} return 1 for successful input, 0 for end of file, and -1 for an error.
\texttt{awk} also provides user-defined functions. Such functions may be defined (in the pattern position of a pattern-action statement) as
\begin{verbatim}
function name(args, ...) { stmts }
func name(args, ...) { stmts }
\end{verbatim}
Function arguments are passed by value if scalar and by reference if array name. Argument names are local to the function; all other variable names are global. Function calls may be nested and functions may be recursive. The \texttt{return} statement may
be used to return a value.

**EXAMPLES**

Print lines longer than 72 characters:
```
length > 72
```

Print first two fields in opposite order:
```
{ print $2, $1 }
```

Same, with input fields separated by comma and/or blanks and tabs:
```
BEGIN { FS = "[,]\[ ]\[ ]+" } { print $2, $1 }
```

Add up first column, print sum and average:
```
{ s += $1 }
END { print "sum is", s, " average is", s/NR }
```

Print fields in reverse order:
```
{ for (i = NF; i > 0; —i) print $i }
```

Print all lines between start/stop pairs:
```
/start/, /stop/
```

Print all lines whose first field is different from previous one:
```
$1 != prev { print; prev = $1 }
```

Simulate `echo(1)`:
```
BEGIN { for (i = 1; i < ARGV; i++) printf "%s", ARGV[i] printf \"n\"
exit
}
```

Print file, filling in page numbers starting at 5:
```
/Page/ { $2 = n++; }
{ print }

command line: awk -f program n=5 input
```

**SEE ALSO**

grep(1), lex(1), oawk(1), printf(3S), sed(1)

**BUGS**

Input white space is not preserved on output if fields are involved. There are no explicit conversions between numbers and strings. To force an expression to be treated as a number add 0 to it; to force it to be treated as a string concatenate the null string (""") to it.
This page intentionally left blank.
NAME

banner – make posters

SYNOPSIS

banner strings

DESCRIPTION

banner prints its arguments (each up to 10 characters long) in large letters on the standard output.

SEE ALSO

echo(1).
NAME  
basename, dirname – deliver portions of path names

SYNOPSIS  
basename string [ suffix ]  
dirname string

DESCRIPTION  
basename deletes any prefix ending in / and the suffix (if present in string) from string,  
and prints the result on the standard output. It is normally used inside substitution  
marks (\~) within shell procedures.  

Dirname delivers all but the last level of the path name in string.

EXAMPLES  
The following example, invoked with the argument /usr/src/cmd/cat.c, compiles the  
named file and moves the output to a file named cat in the current directory:  
    cc $1  
    mv a.out `basename $1 \~.c`  
The following example will set the shell variable NAME to /usr/src/cmd:  
    NAME=`dirname /usr/src/cmd/cat.c`

SEE ALSO  
sh(1).
NAME

bc – arbitrary-precision arithmetic language

SYNOPSIS

bc [-c] [-l] [ file ... ]

DESCRIPTION

bc is an interactive processor for a language that resembles C but provides unlimited
precision arithmetic. It takes input from any files given, then reads the standard
input. The bc(1) utility is actually a preprocessor for dc(1), which it invokes automatic-
ally unless the -c option is present. In this case the dc input is sent to the standard
output instead. The options are as follows:

-c Compile only. The output is send to the standard output.
-l Argument stands for the name of an arbitrary precision math library.
The syntax for bc programs is as follows; L means letter a–z, E means expression, S
means statement.

Comments
are enclosed in /* and */.

Names
simple variables: L
array elements: L [ E ]
The words “ibase”, “obase”, and “scale”

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 right of 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 -1 math library
s(x) sine
c(x) cosine
e(x) exponential
l(x) log
a(x) arctangent
j(n,x) Bessel function

All function arguments are passed by value.

The value of a statement that is an expression is printed unless the main operator is an assignment. Either semicolons or new-lines may separate statements. Assignment to scale influences the number of digits to be retained on arithmetic operations in the manner of dc(1). Assignments to ibase or obase set the input and output number radix respectively.

The same letter may be used as 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 using arrays as function arguments or defining them as automatic variables, empty square brackets must follow the array name.

**EXAMPLE**

scale = 20
define e(x){
auto a, b, c, i, s
a = 1
b = 1
s = 1
for(i=1; i<=10; i++){
a = a*x
b = b*i
c = a/b
if(c == 0) return(s)
s = s+c
}
}  
defines a function to compute an approximate value of the exponential function and
for(i=1; i<10; i++) e(i)

prints approximate values of the exponential function of the first ten integers.

**FILES**

/usr/lib/lib.b   mathematical library
/usr/bin/dc      desk calculator proper

**SEE ALSO**

dc(1).

**BUGS**

The bc command does not yet recognize the logical operators, && and ||. For statement must have all three expressions (E's).
Quit is interpreted when read, not when executed.
NAME
bcgheckrc – system initialization procedures

SYNOPSIS
/etc/bcgheckrc

DESCRIPTION
These shell procedures are executed via entries in /etc/inittab by init(1M) whenever the system is booted (or rebooted).

First, the bcgheckrc procedure checks the status of the root file system. If the root file system is found to be bad, bcgheckrc repairs it.

Then, it clears the mounted file system table, /etc/mnttab, and puts the entry for the root file system into the mount table.

After these two procedures have executed, init checks for the initdefault value in /etc/inittab. This tells init in which run level to place the system. Since initdefault is initially set to 2, the system will be placed in the multi-user state via the /etc/rc2 procedure.

Also, these shell procedures may be used for several run-level states.

SEE ALSO
fsck(1M), init(1M), rc2(8), shutdown(1M)
NAME
bdiff – big diff

SYNOPSIS
bdiff file1 file2 [n] [-s]

DESCRIPTION
bdiff is used in a manner analogous to diff(1) to find which lines in two files must be changed to bring the files into agreement. Its purpose is to allow processing of files which are too large for diff.

The parameters to bdiff are:

file1 (file2)
The name of a file to be used. If file1 (file2) is –, the standard input is read.

n The number of line segments. The value of n is 3500 by default. If the optional third argument is given and it is numeric, it is used as the value for n. This is useful in those cases in which 3500-line segments are too large for diff, causing it to fail.

-s Specifies that no diagnostics are to be printed by bdiff (silent option). Note, however, that this does not suppress possible diagnostic messages from diff(1), which bdiff calls.

bdiff ignores lines common to the beginning of both files, splits the remainder of each file into n-line segments, and invokes diff upon corresponding segments. If both optional arguments are specified, they must appear in the order indicated above.

The output of bdiff is exactly that of diff, with line numbers adjusted to account for the segmenting of the files (that is, to make it look as if the files had been processed whole). Note that because of the segmenting of the files, bdiff does not necessarily find a smallest sufficient set of file differences.

FILES
/tmp/bd?????

SEE ALSO
diff(1), help(1).

DIAGNOSTICS
Use help(1) for explanations.
NAME

bfs - big file scanner

SYNOPSIS

bfs [-] name

DESCRIPTION

The bfs command is (almost) like ed(1) except that it is read-only and processes much larger files. Files can be up to 1024K bytes and 32K lines, with up to 512 characters, including new-line, per line. bfs is usually more efficient than ed(1) for scanning a file, since the file is not copied to a buffer. It is most useful for identifying sections of a large file where csplit(1) can be used to divide it into more manageable pieces for editing.

Normally, the size of the file being scanned is printed, as is the size of any file written with the w command. The optional - suppresses printing of sizes. Input is prompted with * if P and a carriage return are typed, as in ed(1). Prompting can be turned off again by inputting another P and carriage return. Note that messages are given in response to errors if prompting is turned on.

All address expressions described under ed(1) are supported. In addition, regular expressions may be surrounded with two symbols besides / and ?. > indicates downward search without wrap-around, and < indicates upward search without wrap-around. There is a slight difference in mark names: only the letters a through z may be used, and all 26 marks are remembered.

The e, g, v, k, p, q, w, =, ! and null commands operate as described under ed(1). Commands such as --,-, +++, +++, =, -12, and +4p are accepted. Note that 1,10p and 1,10 will both print the first ten lines. The f command only prints the name of the file being scanned; there is no remembered file name. The w command is independent of output diversion, truncation, or crunching (see the xo, xt and xc commands, below). The following additional commands are available:

xf file

Further commands are taken from the named file. When an end-of-file is reached, an interrupt signal is received or an error occurs, reading resumes with the file containing the xf. The xf commands may be nested to a depth of 10.

xn

List the marks currently in use (marks are set by the k command).

xo [file]

Further output from the p and null commands is diverted to the named file, which, if necessary, is created mode 666 (readable and writable by everyone), unless your umask setting (see umask(1)) dictates otherwise. If file is missing, output is diverted to the standard output. Note that each diversion causes truncation or creation of the file.

: label

This positions a label in a command file. The label is terminated by newline, and blanks between the : and the start of the label are ignored. This command may also be used to insert comments into a command file, since labels need not be referenced.

(... )xb/regular expression/label

A jump (either upward or downward) is made to label if the command
succeeds. It fails under any of the following conditions:
1. Either address
   is not between 1 and $.
2. The second address is less than the first.
3. The regular expression does not match at least one line in the
   specified range, including the first and last lines.
On success, . is set to the line matched and a jump is made to label. This
command 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 commands are executed. Note that the command

    xb/^

is an unconditional jump.
The xb command is allowed only if it is read from someplace other than a
terminal. If it is read from a pipe only a downward jump is possible.

xt number
Output from the p and null commands is truncated to at most number
characters. The initial number is 255.

xv[digit] [spaces] [value]
The variable name is the specified digit following the xv. The commands
xv5100 or xv5 100 both assign the value 100 to the variable 5. The command
xv61,100p assigns the value 1,100p to the variable 6. To reference a
variable, put a % in front of the variable name. For example, using the
above assignments for variables 5 and 6:

    1,%5p
    1,%5
    %6

will all print the first 100 lines.

    g/%5/p

would globally search for the characters 100 and print each line containing
a match. To escape the special meaning of %, a \ must precede it.

    g/\\%[cds]/p

could be used to match and list lines containing printf of characters,
decimal integers, or strings.

Another feature of the xv command is that the first line of output from a
UNIX system command can be stored into a variable. The only require-
ment is that the first character of value be an !. For example:

    .w junk
    xv5!cat junk
    !rm junk
    !echo "%5"
    xv6!expr %6 + 1

would put the current line into variable 5, print it, and increment the vari-
able 6 by one. To escape the special meaning of ! as the first character of
value, precede it with a \\.
\texttt{xv7!date}

stores the value \texttt{!date} into variable 7.

\texttt{xbz label}

\texttt{xbn label}

These two commands will test the last saved \textit{return code} from the execution of a UNIX system command (\texttt{!command}) or nonzero value, respectively, to the specified label. The two examples below both search for the next five lines containing the string \textit{size}.

\begin{verbatim}
\texttt{xv55}
\texttt{:1}
\texttt{/size/}
\texttt{xv5!expr \%5 - 1}
\texttt{!if 0\%5 != 0 exit 2}
\texttt{xbn l}
\texttt{xv45}
\texttt{:1}
\texttt{/size/}
\texttt{xv4!expr \%4 - 1}
\texttt{!if 0\%4 = 0 exit 2}
\texttt{xbz l}
\end{verbatim}

\texttt{xc [switch]}

If \texttt{switch} is 1, output from the \texttt{p} and \texttt{n} commands is crunched; if \texttt{switch} is 0 it is not. Without an argument, \texttt{xc} reverses \texttt{switch}. Initially \texttt{switch} is set for no-crunching. Crunched output has strings of tabs and blanks reduced to one blank and blank lines suppressed.

\textbf{SEE ALSO}

csplit(1), ed(1), umask(1).

\textbf{DIAGNOSTICS}

? for errors in commands, if prompting is turned off. Self-explanatory error messages when prompting is on.
NAME

biff – be notified if mail arrives and who it is from.

SYNOPSIS

biff [ yn ]

DESCRIPTION

Biff informs the system whether you want to be notified when mail arrives during the current terminal session. The command

  biff y

enables notification; the command

  biff n

disables it. When mail notification is enabled, the header and first few lines of the message will be printed on your screen whenever mail arrives. A “biff y” command is often included in the file .login or .profile to be executed at each login.

Biff operates asynchronously. For synchronous notification use the MAIL variable of sh(1) or the mail variable of csh(1).

SEE ALSO

csh(1), sh(1), mail(1), comsat(8C)
**NAME**
cal – print calendar

**SYNOPSIS**
cal [[ month ] year ]

**DESCRIPTION**
cal prints a calendar for the specified year. If a month is also specified, a calendar just for that month is printed. If neither is specified, a calendar for the present month is printed. Year can be between 1 and 9999. The month is a number between 1 and 12. The calendar produced is that for England and the United States.

**EXAMPLES**
An unusual calendar is printed for September 1752. That is the month 11 days were skipped to make up for lack of leap year adjustments. To see this calendar, type: cal 9 1752

**BUGS**
The year is always considered to start in January even though this is historically naive.
Beware that "cal 83" refers to the early Christian era, not the 20th century.
NAME
calendar – reminder service

SYNOPSIS
calendar [ – ]

DESCRIPTION
calendar consults the file calendar in the current directory and prints out lines that contain today’s or tomorrow’s date anywhere in the line. Most reasonable month-day dates such as “Aug. 24,” “august 24,” “8/24,” etc., are recognized, but not “24 August” or “24/8”. On weekends “tomorrow” extends through Monday.

When an argument is present, calendar does its job for every user who has a file calendar in his or her login directory and sends them any positive results by mail(1). Normally this is done daily by facilities in the UNIX operating system.

FILES
/usr/lib/calprog   to figure out today’s and tomorrow’s dates
/etc/passwd
/tmp/cal*

SEE ALSO
mail(1).

BUGS
Your calendar must be public information for you to get reminder service. calendar’s extended idea of “tomorrow” does not account for holidays.
NAME
captinfo – convert a termcap description into a terminfo description

SYNOPSIS
captinfo [-v ...] [-V] [-1] [-w width] file ...

DESCRIPTION
captinfo looks in file for termcap descriptions. For each one found, an equivalent
terminfo(4) description is written to standard output, along with any comments found.
A description which is expressed as relative to another description (as specified in
the termcap tc= field) will be reduced to the minimum superset before being output.

If no file is given, then the environment variable TERCAP is used for the filename or
entry. If TERCAP is a full pathname to a file, only the terminal whose name is
specified in the environment variable TERM is extracted from that file. If the
environment variable TERCAP is not set, then the file /etc/termcap is read.

-v print out tracing information on standard error as the program runs.
Specifying additional -v options will cause more detailed information to
be printed.

-V print out the version of the program in use on standard error and exit.

-1 cause the fields to print out one to a line. Otherwise, the fields will be
printed several to a line to a maximum width of 60 characters.

-w change the output to width characters.

FILES
/usr/lib/terminfo/?/* compiled terminal description database

CAVEATS
Certain termcap defaults are assumed to be true. For example, the bell character (terminfo bel)
is assumed to be ^G. The linefeed capability (termcap nl) is assumed to be
the same for both cursor_down and scroll_forward (terminfo cu1 and ind, respectively.)
Padding information is assumed to belong at the end of the string.

The algorithm used to expand parameterized information for termcap fields such as
cursor_position (termcap cn, terminfo cup) will sometimes produce a string which,
though technically correct, may not be optimal. In particular, the rarely used termcap
operation %n will produce strings that are especially long. Most occurrences of these
non-optimal strings will be flagged with a warning message and may need to be
recoded by hand.

The short two-letter name at the beginning of the list of names in a termcap entry, a
hold-over from an earlier version of the UNIX system, has been removed.

DIAGNOSTICS
tgetent failed with return code n (reason).
    The termcap entry is not valid. In particular, check for an invalid
    ‘tc=’ entry.
unknown type given for the termcap code cc.
    The termcap description had an entry for cc whose type was not
    boolean, numeric or string.
wrong type given for the boolean (numeric, string) termcap code cc.
    The boolean termcap entry cc was entered as a numeric or string
capability.
the boolean (numeric, string) termcap code cc is not a valid name.
    An unknown termcap code was specified.
tgetent failed on TERM=term.

The terminal type specified could not be found in the termcap file.

TERM=term: cap cc (info ii) is NULL: REMOVED

The termcap code was specified as a null string. The correct way to cancel an entry is with an '®', as in ':bs®'. Giving a null string could cause incorrect assumptions to be made by the software which uses termcap or terminfo.

a function key for cc was specified, but it already has the value vv.

When parsing the ko capability, the key cc was specified as having the same value as the capability cc, but the key cc already had a value assigned to it.

the unknown termcap name cc was specified in the ko termcap capability.

A key was specified in the ko capability which could not be handled.

the vi character v (info ii) has the value xx, but ma gives n.

The ma capability specified a function key with a value different from that specified in another setting of the same key.

the unknown vi key v was specified in the ma termcap capability.

A vi(1) key unknown to captoinfo was specified in the ma capability.

Warning: termcap sg (nn) and termcap ug (nn) had different values.

derminfo assumes that the sg (now xmc) and ug values were the same.

Warning: the string produced for ii may be inefficient.

The parameterized string being created should be rewritten by hand.

Null termname given.

The terminal type was null. This is given if the environment variable TERM is not set or is null.

cannot open file for reading.

The specified file could not be opened.

SEE ALSO

curses (3X), infocmp(1M), terminfo(4), tic(1M).

NOTES

captoinfo should be used to convert termcap entries to terminfo(4) entries because the termcap database (from earlier versions of UNIX System V) may not be supplied in future releases.
NAME
cat – concatenate and print files

SYNOPSIS
cat [ -u ] [ -s ] [ -v [-t] [-e] ] file ...

DESCRIPTION
cat reads each file in sequence and writes it on the standard output. Thus:
cat file
prints the file, and:
cat file1 file2 > file3
concatenates the first two files and places the result on the third.
If no input file is given, or if the argument – is encountered, cat reads from the standard input file.
The following options apply to cat.
- u  The output is not buffered. (The default is buffered output.)
- s  cat is silent about non-existent files.
- v  Causes non-printing characters (with the exception of tabs, new-lines and
     form-feeds) to be printed visibly. Control characters are printed \^X (control-x);
     the DEL character (octal 0177) is printed \^?. Non-ASCII characters (with the
     high bit set) are printed as M-x, where x is the character specified by the seven
     low order bits.

When used with the -v option, the following options may be used.
- t  Causes tabs to be printed as \^I's.
- e  Causes a $ character to be printed at the end of each line (prior to the new-line).
The -t and -e options are ignored if the -v option is not specified.

WARNING
Command formats such as cat file1 file2 > file1 will cause the original data in file1 to
be lost; therefore, take care when using shell special characters.

SEE ALSO
cp(1), pg(1), pr(1).
NAME

cat – catenate and print

SYNOPSIS

cat [ -u ] [ -n ] [ -s ] [ -v ] file ...

DESCRIPTION

cat reads each file in sequence and displays it on the standard output. Thus

cat file

displays the file on the standard output, and

cat file1 file2 >file3

concatenates the first two files and places the result on the third.

If no input file is given, or if the argument ‘-’ is encountered, cat reads from the standard input file. Output is buffered in the block size recommended by stat(2) unless the standard output is a terminal, when it is line buffered. The -u option makes the output completely unbuffered.

The -n option displays the output lines preceded by lines numbers, numbered sequentially from 1. Specifying the -b option with the -n option omits the line numbers from blank lines.

The -s option crushes out multiple adjacent empty lines so that the output is displayed single spaced.

The -v option displays non-printing characters so that they are visible. Control characters print like ^X for control-x; the delete character (octal 0177) prints as ^?. Non-ascii characters (with the high bit set) are printed as M- (for meta) followed by the character of the low 7 bits. A -e option may be given with the -v option, which displays a ‘$’ character at the end of each line. Specifying the -t option with the -v option displays tab characters as ^I.

SEE ALSO

cp(1), ex(1), more(1), pr(1), tail(1)

BUGS

Beware of ‘cat a b >a’ and ‘cat a b >b’, which destroy the input files before reading them.
NAME
cb – C program beautifier

SYNOPSIS
cb [ -s ] [ -j ] [ -l leng ] [ file ... ]

DESCRIPTION
The cb command reads C programs either from its arguments or from the standard input, and writes them on the standard output with spacing and indentation that display the structure of the code. Under default options, cb preserves all user newlines.

cb accepts the following options.

-\texttt{s} \hspace{1em} \text{Canonicalizes the code to the style of Kernighan and Ritchie in The C Programming Language.}

-\texttt{j} \hspace{1em} \text{Causes split lines to be put back together.}

-\texttt{l leng} \hspace{1em} \text{Causes cb to split lines that are longer than leng.}

SEE ALSO
cc(1).

BUGS
Punctuation that is hidden in preprocessor statements will cause indentation errors.
NAME

cc – C compiler

SYNOPSIS

cc [ options ] [ files ] [ options ] [ files ]

DESCRIPTION

The cc command is an interface to the Stardent 1500/3000 Compilation System. The compilation tools consist of a preprocessor, compiler, beautifier, assembler, and link editor. The cc command processes the supplied options and then executes the various tools with the proper arguments. The cc command accepts several types of files as arguments:

Files whose names end with .c are taken to be C source programs and may be preprocessed, compiled, optimized, assembled, and link edited. The compilation process may be stopped after the completion of any pass if the appropriate options are supplied. If the compilation process runs through the assembler then an object program is produced and is left in the file whose name is that of the source with .o substituted for .c. However, the .o file is normally deleted if a single C program is compiled and then immediately link edited. In the same way, files whose names end in .s are taken to be assembly source programs, and may be assembled and link edited; and files whose names end in .i are taken to be preprocessed C source programs and may be compiled, optimized, assembled and link edited. Files whose names do not end in .c, .s or .i are handed to the link editor.

Since the cc command usually creates files in the current directory during the compilation process, it is necessary to run the cc command in a directory in which a file can be created.

The following options are interpreted by cc:

--c Suppress the link editing phase of the compilation, and do not remove any produced object files.

--cpp Invoke the C preprocessor on the source file before compiling.

--cross_reference
   Generate a cross-reference, if a listing is generated.

--Dname
   Define name to have the value of 1, to the preprocessor.

--Dname=val
   Define name to have the value of val, to the preprocessor.

--debug
   Add debug data to the object file. Force optimization level to zero. This is synonymous with -g.

--E Run only cpp(1) on the named C programs, and send the result to the standard output.

--full_report
   Produce a detailed vectorizer report.

--fullsubcheck
   Generate code to check that every subscript in every array reference is within the bounds of the appropriate array dimensions.

--g Generate additional information needed for the use of dbg(1). Force optimization level to zero. This option is synonymous with -debug.
-I Suppress the default searching for preprocessor included files in /usr/include.

-Idir Search for include files in dir.

-i Suppress the automatic production of #ident information.

-inline
   Instruct the compiler to enable function inlining.

-Npaths=name.in
   Instruct the compiler to make use of the database of functions listed in the catalog name.in as the source for inlining.

-NW Suppress compiler warnings.

-n Suppress the standard C startup routine.

-O0 Turn off all optimizations.

-O1 Perform common subexpression elimination and instruction scheduling. If nothing is specified, this -O1 is the default setting of compiler optimization level.

-O2 Perform -O1 and vectorization.

-O3 Perform -O2 and parallelization.

-O This is synonymous with -O1.

-o filename
   Place the output into filename.

-P Run only cpp(1) on the named C programs and and leave the result in corresponding files suffixed .i. This option is passed to cpp(1).

-p Generate code to profile the loaded program during execution. (See prof(1) and mkprof(1).)

-ploop
   Generate code which allows loops within a single routine to be profiled separately.

-r Produce a relocatable output file.

-S Compile and do not assemble the named C programs, and leave the assembler output in corresponding files suffixed .s.

-safe=loops
   Guarantee that all for loops within the program have upper bounds that do not vary within the loop.

-safe=parms
   Declare that input arguments do not have hidden aliases.

-safe=ptrs
   Declare that pointers do not have hidden aliases.

-subcheck
   Produce code to check at runtime to ensure that each array element accessed is actually part of the appropriate array. However, at optimization level 02 and higher, this option ignores the vector mask. This means that some operations may generate subscriptranges that are not actually in the code.

-Uname
   Undefine name.
-V  Print version information.
-v  Generate more messages tracking the progress of the compilation.
-vector_c
   This is equivalent to specifying -safe=parms -safe=loops.
-vreport
   Invoke the vector reporting facility and tell the user what vectorization has
   been done. A detailed listing is provided for each loop nest and suggestions for
   achieving better performance are included.
-vsummary
   Invoke the vector reporting facility and tell the user what vectorization has
   been done. Print out what statements are and are not vectorized in each loop.
   This output is in Fortran-like notation.
-w  Suppress warning messages during compilation.
-43  Use this option to get 4.3 BSD header files and libraries.

The cc command recognizes -B hhhhhhh, -D hhhhhhh, -esym, -L, -Ldir, -ltag, -m, -N,
-filename, -opct, -p, -r, -s, -T hhhhhhh, -t, -uname, and -yname and passes these
options and their arguments directly to the loader. See the manual pages for cpp(1)
and ld(1) for descriptions.

Other arguments are taken to be C compatible object programs, typically produced
by an earlier cc run, or perhaps libraries of C compatible routines and are passed
directly to the link editor. These programs, together with the results of any compilations specified, are link edited (in the order given) to produce an executable program
with name a.out.

FILES

file.c          C source file
file.o          object file
file.s          assembly language file
a.out           link edited output
/lib/crt0.o     start-up routine
TMPDIR/*        temporary files
/lib/cpp        preprocessor, cpp(1)
/bin/as         assembler, as(1)
/bin/ld         link editor, ld(1)
/lib/libc.a     standard C library

TMPDIR is usually /usr/tmp but can be redefined by setting the environment vari-
able TMPDIR [see tempnam() in tmpnam(3)].

SEE ALSO

as(1), dbg(1), ld(1), cpp(1), mkprof(1), prof(1)
Kernighan, B. W., and Ritchie, D. M., The C Programming Language, Prentice-Hall,

NOTES

By default, the return value from a compiled C program is completely random. The
only two guaranteed ways to return a specific value is to explicitly call exit(2) or to
leave the function main() with a "return expression;" construct.
NAME

    cd – change working directory

SYNOPSIS

    cd [ directory ]

DESCRIPTION

    If directory is not specified, the value of shell parameter $HOME is used as the new working directory. If directory specifies a complete path starting with /, directory becomes the new working directory. If neither case applies, cd tries to find the designated directory relative to one of the paths specified by the $CDPATH shell variable. $CDPATH has the same syntax as, and similar semantics to, the $PATH shell variable. cd must have execute (search) permission in directory.

    Because a new process is created to execute each command, cd would be ineffective if it were written as a normal command; therefore, it is recognized and is internal to the shell.

SEE ALSO

    csh(1), pwd(1), sh(1)
NAME

cdc – change the delta commentary of an SCCS delta

SYNOPSIS

cdc -rSID [-m|mrlist] [-y|comment] files

DESCRIPTION

cdc changes the delta commentary, for the SID (SCCS IDentification string) specified by the -r keyletter, of each named SCCS file.

Delta commentary is defined to be the Modification Request (MR) and comment information normally specified via the delta(1) command (-m and -y keyletters).

If a directory is named, cdc behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with .) and unreadable files are silently ignored. If a name of -r is given, the standard input is read (see WARNINGS) and each line of the standard input is taken to be the name of an SCCS file to be processed.

Arguments to cdc, which may appear in any order, consist of keyletter arguments and file names.

All the described keyletter arguments apply independently to each named file:

- -rSID Used to specify the SCCS IDentification (SID) string of a delta for which the delta commentary is to be changed.

- -m|mrlist If the SCCS file has the v flag set [see admin(1)] then a list of MR numbers to be added and/or deleted in the delta commentary of the SID specified by the -r keyletter may be supplied. A null MR list has no effect.

MR entries are added to the list of MRs in the same manner as that of delta(1). In order to delete an MR, precede the MR number with the character ! (see EXAMPLES). If the MR to be deleted is currently in the list of MRs, it is removed and changed into a "comment" line. A list of all deleted MRs is placed in the comment section of the delta commentary and preceded by a comment line stating that they were deleted.

If -m is not used and the standard input is a terminal, the prompt MRs? is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. The MRs? prompt always precedes the comments? prompt (see -y keyletter).

MRs in a list are separated by blanks and/or tab characters. An unescaped new-line character terminates the MR list.

Note that if the v flag has a value [see admin(1)], it is taken to be the name of a program (or shell procedure) which validates the correctness of the MR numbers. If a non-zero exit status is returned from the MR number validation program, cdc terminates and the delta commentary remains unchanged.

- -y|comment Arbitrary text used to replace the comment(s) already existing for the delta specified by the -r keyletter. The previous comments are kept and preceded by a comment line stating that they were changed. A null comment has no effect.
If -y is not specified and the standard input is a terminal, the prompt comments? is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. An unescaped new-line character terminates the comment text.

Simply stated, the keyletter arguments are either (1) if you made the delta, you can change its delta commentary; or (2) if you own the file and directory you can modify the delta commentary.

**EXAMPLES**

```c
cdc -r1.6 -m"bl78-12345 !bl77-54321 bl79-00001" --trouble s.file
```

adds bl78-12345 and bl79-00001 to the MR list, removes bl77-54321 from the MR list, and adds the comment trouble to delta 1.6 of s.file.

```c
cdc -r1.6 s.file
MRs? !bl77-54321 bl78-12345 bl79-00001
comments? trouble
```

does the same thing.

**WARNINGS**

If SCCS file names are supplied to the cdc command via the standard input (- on the command line), then the -m and -y keyletters must also be used.

**FILES**

- x-file [see delta(1)]
- z-file [see delta(1)]

**SEE ALSO**

admin(1), delta(1), get(1), help(1), prs(1), sccsfile(4).

**DIAGNOSTICS**

Use help(1) for explanations.
NAME
chfn – change finger entry

SYNOPSIS
chfn [name]

DESCRIPTION
chfn is used to change information about users. The information is stored in the /etc/passwd file and used by finger(1). It is equivalent to passwd -f. chfn prompts the user for information for each field. Included in the prompt string is the current value for the field enclosed in brackets. Accept the default value by typing a carriage return. Enter a blank field by typing none. Here's a sample session:

Changing finger information for paul.
Default values are printed inside of '[]'.
To accept the default, type <return>.
To have a blank entry, type the word 'none'.

Name [Paul Anthony]:
Office []:
Office Phone [555]: 666
Home Phone [4085551211]: 408-555-1212

The optional argument name is used to change another person's finger information. This can only be done by the super-user.

FILES
/etc/passwd

SEE ALSO
finger(1), passwd(5)
NAME

chmod – change mode

SYNOPSIS

chmod mode file ...
chmod mode directory ...

DESCRIPTION

The permissions of the named files or directories are changed according to mode, which may be symbolic or absolute. Absolute changes to permissions are stated using octal numbers:

```
chmod nnn file(s)
```

where n is a number from 0 to 7. Symbolic changes are stated using mnemonic characters:

```
chmod a operator b file(s)
```

where a is one or more characters corresponding to user, group, or other; where operator is +, -, and =, signifying assignment of permissions; and where b is one or more characters corresponding to type of permission.

An absolute mode is given as an octal number constructed from the OR of the following modes:

- 4000  set user ID on execution
- 2000  set group ID on execution if # is 7, 5, 3, or 1
- 1000  sticky bit is turned on (see chmod(2))
- 0400  read by owner
- 0200  write by owner
- 0100  execute (search in directory) by owner
- 0070  read, write, execute (search) by group
- 0007  read, write, execute (search) by others

Symbolic changes are stated using letters that correspond both to access classes and to the individual permissions themselves. Permissions to a file may vary depending on your user identification number (UID) or group identification number (GID). Permissions are described in three sequences each having three characters:

```
User    Group    Other
rwx  rwx  rwx
```

This example (meaning that user, group, and others all have reading, writing, and execution permission to a given file) demonstrates two categories for granting permissions: the access class and the permissions themselves.

Thus, to change the mode of a file's (or directory's) permissions using chmod's symbolic method, use the following syntax for mode:

```
[ who ] operator [ permission(s) ], ...
```

A command line using the symbolic method would appear as follows:

```
chmod g+rw file
```

This command would make file readable and writable by the group.

The who part can be stated as one or more of the following letters:
u user’s permissions

g group’s permissions

o others permissions

The letter a (all) is equivalent to ugo and is the default if who is omitted.

Operator can be + to add permission to the file’s mode, – to take away permission, or = to assign permission absolutely. (Unlike other symbolic operations, = has an absolute effect in that it resets all other bits.) Omitting permission is only useful with = to take away all permissions.

Permission is any compatible combination of the following letters:

r reading permission

w writing permission

x execution permission

s user or group set-ID is turned on

t sticky bit is turned on

l mandatory locking will occur during access

Multiple symbolic modes separated by commas may be given, though no spaces may intervene between these modes. Operations are performed in the order given. Multiple symbolic letters following a single operator cause the corresponding operations to be performed simultaneously. The letter s is only meaningful with u or g, and t only works with u.

Mandatory file and record locking (l) refers to a file’s ability to have its reading or writing permissions locked while a program is accessing that file. It is not possible to permit group execution and enable a file to be locked on execution at the same time. In addition, it is not possible to turn on set-group-ID and enable a file to be locked on execution at the same time. The following examples,

```
chmod g+x,+l file

chmod g+s,+l file
```

are, therefore, illegal usages and will elicit error messages.

Only the owner of a file or directory (or the super-user) may change a file’s mode. Only the super-user may set the sticky bit. In order to turn on a file’s set-group-ID, your own group ID must correspond to the file’s, and group execution must be set.

**EXAMPLES**

```
chmod a-x file

chmod 444 file
```

The first examples deny execution permission to all. The absolute (octal) example permits only reading permissions.

```
chmod go+rw file

chmod 606 file
```

These examples make a file readable and writable by the group and others.

```
chmod +l file
```

This causes a file to be locked during access.

```
chmod =rwx,g+s file

chmod 2777 file
```
These last two examples enable all to read, write, and execute the file; and they turn on the set group-ID.

SEE ALSO

ls(1), chmod(2)
NAME

chown, chgrp – change owner or group

SYNOPSIS

chown owner file ...
chown owner directory ...
chgrp group file ...
chgrp group directory ...

DESCRIPTION

chown changes the owner of the files or directories to owner. The owner may be either a decimal user ID or a login name found in the password file.

Chgrp changes the group ID of the files or directories to group. The group may be either a decimal group ID or a group name found in the group file.

If either command is invoked by other than the super-user, the set-user-ID and set-group-ID bits of the file mode, 04000 and 02000 respectively, will be cleared.

Only the owner of a file (or the super-user) may change the owner or group of that file.

FILES

/etc/passwd
/etc/group

NOTES

In a Remote File Sharing environment, you may not have the permissions that the output of the ls -l command leads you to believe. For more information see the System Administrator's Guide.

SEE ALSO

chmod(1). chown(2), group(4), passwd(4).
NAME
croot - change root directory for a command

SYNOPSIS
/etc/chroot newroot command

DESCRIPTION
croot causes the given command to be executed relative to the new root. The meaning of any initial slashes (/) in the path names is changed for the command and any of its child processes to newroot. Furthermore, upon execution, the initial working directory is newroot.

Notice, however, that if you redirect the output of the command to a file:

    croot newroot command >x

will create the file x relative to the original root of the command, not the new one.

The new root path name is always relative to the current root: even if a croot is currently in effect, the newroot argument is relative to the current root of the running process.

This command can be run only by the super-user.

SEE ALSO
cd(1), chroot(2).

BUGS
One should exercise extreme caution when referencing device files in the new root file system.
NAME

csh – change default login shell

SYNOPSIS

chsh name [ shell ]

DESCRIPTION

csh is identical with passwd -s and is used to change your login shell. If no shell is specified, the shell reverts to the default login shell, /bin/sh. Only the super-user may specify anything other than /bin/csh, /bin/oldcsh, or /usr/new/csh. For example,

    chsh paul /bin/csh

SEE ALSO

csh(1), passwd(1), passwd(5)
NAME

clear – clear terminal screen

SYNOPSIS

clear

DESCRIPTION

Clear clears your screen if this is possible. It looks in the environment for the terminal type and then in /usr/lib/terminfo to figure out how to clear the screen.

FILES

/usr/lib/terminfo terminal information data base
NAME
crli – clear i-node

SYNOPSIS
/etc/crli special i-number ...

DESCRIPTION
crli writes nulls on the 64 bytes at offset i-number from the start of the i-node list. This effectively eliminates the i-node at that address. Special is the device name on which a file system has been defined. After crli is executed, any blocks in the affected file will show up as “not accounted for” when fsck(1M) is run against the file-system. The i-node may be allocated to a new file.

Read and write permission is required on the specified special device.

This command is used to remove a file which appears in no directory; that is, to get rid of a file which cannot be removed with the rm command.

SEE ALSO
fs(4), fsck(1M), fsdb(1M), ncheck(1M), rm(1).

WARNINGS
If the file is open for writing, crli will not work. The file system containing the file should NOT be mounted.

If crli is used on the i-node number of a file that does appear in a directory, it is imperative to remove the entry in the directory at once, since the i-node may be allocated to a new file. The old directory entry, if not removed, continues to point to the same file. This sounds like a link, but does not work like one. Removing the old entry destroys the new file.
NAME
cmp – compare two files

SYNOPSIS
cmp [-l] [-s] file1 file2

DESCRIPTION
The two files are compared. (If file1 is -, the standard input is used.) Under default options, cmp makes no comment if the files are the same; if they differ, it announces the byte and line number at which the difference occurred. If one file is an initial subsequence of the other, that fact is noted.

Options:
-1 Print the byte number (decimal) and the differing bytes (octal) for each difference.
-s Print nothing for differing files; return codes only.

SEE ALSO
comm(1), diff(1).

DIAGNOSTICS
Exit code 0 is returned for identical files, 1 for different files, and 2 for an inaccessible or missing argument.
**NAME**

col – filter reverse line-feeds

**SYNOPSIS**

```
col [-b] [-f] [-x] [-p]
```

**DESCRIPTION**

col reads from the standard input and writes onto the standard output. It performs the line overlays implied by reverse line feeds (ASCII code ESC-7), and by forward and reverse half-line-feeds (ESC-9 and ESC-8). col is particularly useful for filtering multicolumn output made with the .rt command of nroff and output resulting from use of the tbl(1) preprocessor.

If the `-b` option is given, col 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 place, only the last one read will be output.

Although col accepts half-line motions in its input, it normally does not emit them on output. Instead, text that would appear between lines is moved to the next lower full-line boundary. This treatment can be suppressed by the `-f` (fine) option; in this case, the output from col may contain forward half-line-feeds (ESC-9), but will still never contain either kind of reverse line motion.

Unless the `-x` option is given, col will convert white space to tabs on output wherever possible to shorten printing time.

The ASCII control characters SO (\017) and SI (\016) are assumed by col to start and end text in an alternate character set. The character set to which each input character belongs is remembered, and on output SI and SO characters are generated as appropriate to ensure that each character is printed in the correct character set.

On input, the only control characters accepted are space, backspace, tab, return, new-line, SI, SO, VT (\013), and ESC followed by 7, 8, or 9. The VT character is an alternate form of full reverse line-feed, included for compatibility with some earlier programs of this type. All other non-printing characters are ignored.

Normally, col will ignore any escape sequences unknown to it that are found in its input; the `-p` option may be used to cause col to output these sequences as regular characters, subject to overprinting from reverse line motions. The use of this option is highly discouraged unless the user is fully aware of the textual position of the escape sequences.

**SEE ALSO**

nroff(1), tbl(1).

**NOTES**

The input format accepted by col matches the output produced by nroff with either the `-T37` or `-T1p` options. Use `-T37` (and the `-f` option of col) if the ultimate disposition of the output of col will be a device that can interpret half-line motions, and `-T1p` otherwise.

**BUGS**

Cannot back up more than 128 lines.

Allows at most 800 characters, including backspaces, on a line.

Local vertical motions that would result in backing up over the first line of the document are ignored. As a result, the first line must not have any superscripts.
NAME

colrct - filter nroff output for CRT previewing

SYNOPSIS

colrct [-] [-2] [ file ... ]

DESCRIPTION

Colrct provides virtual half-line and reverse line feed sequences for terminals without such capability, and on which overstriking is destructive. Half-line characters and underlining (changed to dashing `\-'') are placed on new lines in between the normal output lines.

The optional `-` suppresses all underlining. It is especially useful for previewing "allboxed" tables from tbl(1).

The option `-2` causes all half-lines to be printed, effectively double spacing the output. Normally, a minimal space output format is used which will suppress empty lines. The program never suppresses two consecutive empty lines, however. The `-2` option is useful for sending output to the line printer when the output contains superscripts and subscripts which would otherwise be invisible.

A typical use of colrct would be

tbl exum.2.n | nroff -ms | colrct - | more

SEE ALSO

nroff/troff(1), col(1), more(1), ul(1)

BUGS

Should fold underlines onto blanks even with the `\-' option so that a true underline character would show; if we did this, however, colrct wouldn't get rid of `cu'd underlining completely.

Can't back up more than 102 lines.

General overstriking is lost; as a special case `\l' overstruck with `\-' or underline becomes `\+'.

Lines are trimmed to 132 characters.

Some provision should be made for processing superscripts and subscripts in documents which are already double-spaced.
COMB(1)

NAME
comb - combine SCCS deltas

SYNOPSIS
comb files

DESCRIPTION
comb generates a shell procedure [see sh(1)] which, when run, will reconstruct the
given SCCS files. The reconstructed files will, hopefully, be smaller than the original
files. The arguments may be specified in any order, but all keyletter arguments
apply to all named SCCS files. If a directory is named, comb behaves as though each
file in the directory were specified as a named file, except that non-SCCS files (last
component of the path name does not begin with s.) and unreadable files are silently
ignored. If a name of - is given, the standard input is read; each line of the input is
taken to be the name of an SCCS file to be processed; non-SCCS files and unreadable
files are silently ignored. The generated shell procedure is written on the standard
output.

The keyletter arguments are as follows. Each is explained as though only one named
file is to be processed, but the effects of any keyletter argument apply independently
to each named file.

-o  For each get -e generated, this argument causes the reconstructed file to
be accessed at the release of the delta to be created, otherwise the recon-
structed file would be accessed at the most recent ancestor. Use of the -o
keyletter may decrease the size of the reconstructed SCCS file. It may also
alter the shape of the delta tree of the original file.

-s  This argument causes comb to generate a shell procedure which, when
run, will produce a report giving, for each file: the file name, size (in
blocks) after combining, original size (also in blocks), and percentage
change computed by:

    100 * (original - combined) / original

It is recommended that before any SCCS files are actually combined, one
should use this option to determine exactly how much space is saved by
the combining process.

-pSID The SCCS IDentification string (SID) of the oldest delta to be preserved.
All older deltas are discarded in the reconstructed file.

-cList A list (see get(1) for the syntax of a list) of deltas to be preserved. All
other deltas are discarded.

If no keyletter arguments are specified, comb will preserve only leaf deltas and the
minimal number of ancestors needed to preserve the tree.

FILES
s.COMB The name of the reconstructed SCCS file.
comb????? Temporary.

SEE ALSO
admin(1), delta(1), get(1), help(1), prs(1), sh(1), sccsfile(4)

DIAGNOSTICS
Use help(1) for explanations.

BUGS
comb may rearrange the shape of the tree of deltas. It may not save any space; in fact,
it is possible for the reconstructed file to actually be larger than the original.
NAME
comm – select or reject lines common to two sorted files

SYNOPSIS
comm [ – [ 123 ] ] file1 file2

DESCRIPTION
*comm* reads *file1* and *file2*, which should be ordered in ASCII collating sequence (see *sort*(1)), and produces a three-column output: lines only in *file1*; lines only in *file2*; and lines in both files. The file name – means the standard input.

Flags 1, 2, or 3 suppress printing of the corresponding column. Thus *comm* –12 prints only the lines common to the two files; *comm* –23 prints only lines in the first file but not in the second; *comm* –123 prints nothing.

SEE ALSO
cmp(1), diff(1), sort(1), uniq(1).
NAME
compress, uncompress, zcat – compress and expand data

SYNOPSIS
compress [-f] [-v] [-c] [-b bits] [name ...]
uncompress [-f] [-v] [-c] [name ...]
zcat [name ...]

DESCRIPTION
Compress reduces the size of the named files using adaptive Lempel-Ziv coding. Whenever possible, each file is replaced by one with the extension .Z, while keeping the same ownership modes, access and modification times. If no files are specified, the standard input is compressed to the standard output. Compressed files can be restored to their original form using uncompress or zcat.

The -f option will force compression of name, even if it does not actually shrink or the corresponding name.Z file already exists. Except when run in the background under /bin/sh, if -f is not given the user is prompted as to whether an existing name.Z file should be overwritten.

The -c ("cat") option makes compress/uncompress write to the standard output; no files are changed. The nondestructive behavior of zcat is identical to that of uncompress -c.

Compress uses the modified Lempel-Ziv algorithm popularized in "A Technique for High Performance Data Compression", Terry A. Welch, IEEE Computer, vol. 17, no. 6 (June 1984), pp. 8-19. Common substrings in the file are first replaced by 9-bit codes 257 and up. When code 512 is reached, the algorithm switches to 10-bit codes and continues to use more bits until the limit specified by the -b flag is reached (default 16). Bits must be between 9 and 16. The default can be changed in the source to allow compress to be run on a smaller machine.

After the bits limit is attained, compress periodically checks the compression ratio. If it is increasing, compress continues to use the existing code dictionary. However, if the compression ratio decreases, compress discards the table of substrings and rebuilds it from scratch. This allows the algorithm to adapt to the next "block" of the file.

Note that the -b flag is omitted for uncompress, since the bits parameter specified during compression is encoded within the output, along with a magic number to ensure that neither decompression of random data nor recompression of compressed data is attempted.

The amount of compression obtained depends on the size of the input, the number of bits per code, and the distribution of common substrings. Typically, text such as source code or English is reduced by 50-60%. Compression is generally much better than that achieved by Huffman coding (as used in pack), or adaptive Huffman coding (compact), and takes less time to compute.

The -v option causes the printing of the percentage reduction of each file.

If an error occurs, exit status is 1, else if the last file was not compressed because it became larger, the status is 2; else the status is 0.

DIAGNOSTICS
Usage: compress [-fvc] [-b maxbits] [file ...]
  Invalid options were specified on the command line.
Missing maxbits
  Maxbits must follow -b.
file: not in compressed format
  The file specified to uncompress has not been compressed.
file: compressed with \( xx \) bits, can only handle \( yy \) bits
File was compressed by a program that could deal with more bits than the compress code on this machine. Recompress the file with smaller bits.

file: already has \( .Z \) suffix -- no change
The file is assumed to be already compressed. Rename the file and try again.

file: filename too long to tack on \( .Z \)
The file cannot be compressed because its name is longer than 12 characters. Rename and try again. This message does not occur on BSD systems.

file already exists; do you wish to overwrite (y or n)?
Respond "y" if you want the output file to be replaced; "n" if not.

uncompress: corrupt input
A SIGSEGV violation was detected which usually means that the input file is corrupted.

Compression: \( xx.xx\% \)
Percentage of the input saved by compression. (Relevant only for \( -v \).)

-- not a regular file: unchanged
When the input file is not a regular file, (e.g. a directory), it is left unaltered.

-- has \( xx \) other links: unchanged
The input file has links; it is left unchanged. See ln(1) for more information.

-- file unchanged
No savings is achieved by compression. The input remains virgin.

**BUGS**

Although compressed files are compatible between machines with large memory, \( -b12 \) should be used for file transfer to architectures with a small process data space (64KB or less, as exhibited by the DEC PDP series, the Intel 80286, etc.)

compress should be more flexible about the existence of the \( .Z \) suffix.
NAME
cp, ln, mv — copy, link or move files

SYNOPSIS

cp file1 [ file2 ...] target
ln [-f -s ] file1 [ file2 ...] target
mv [-f ] file1 [ file2 ...] target

DESCRIPTION

file1 is copied (linked, moved) to target. Under no circumstance can file1 and target be the same (take care when using slh(1) metacharacters). If target is a directory, then one or more files are copied (linked, moved) to that directory. If target is a file, its contents are destroyed.

If mv or ln determines that the mode of target forbids writing, it will print the mode (see chmod(2)), ask for a response, and read the standard input for one line; if the line begins with y, the mv or ln occurs, if permissible; if not, the command exits. When the -f option is used or if the standard input is not a terminal, no questions are asked and the mv or ln is done.

The -s option causes ln to create symbolic links. You may not create a symbolic link to a target file if the file already exists. There are no checks on the existence of the source of the symbolic link.

A symbolic link contains the name of the file to which it is linked. The referenced file is used when an open(2) operation is performed on the link. A stat(2) on a symbolic link will return the linked-to file; an lstat(2) must be done to obtain information about the link. The readlink(2) call may be used to read the contents of a symbolic link. Symbolic links may span file systems and may refer to directories.

Only mv will allow file1 to be a directory, in which case the directory rename will occur only if the two directories have the same parent; file1 is renamed target. If file1 is a file and target is a link to another file with links, the other links remain and target becomes a new file.

When using cp, if target is not a file, a new file is created which has the same mode as file1 except that the sticky bit is not set unless you are super-user; the owner and group of target are those of the user. If target is a file, copying a file into target does not change its mode, owner, nor group. The last modification time of target (and last access time, if target did not exist) and the last access time of file1 are set to the time the copy was made. If target is a link to a file, all links remain and the file is changed.

SEE ALSO

chmod(1), cpio(1), rm(1).

WARNINGS

ln will not create hard links across file systems. This restriction is necessary because file systems can be added and removed.

BUGS

If file1 and target lie on different file systems, mv must copy the file and delete the original. In this case any linking relationship with other files is lost.
NAME
cp - copy

SYNOPSIS
cp [ -ip ] file1 file2
cp [ -ipr ] file ... directory

DESCRIPTION
file1 is copied onto file2. By default, the mode and owner of file2 are preserved if it already existed; otherwise the mode of the source file modified by the current umask(2) is used. The -p option causes cp to attempt to preserve (duplicate) in its copies the modification times and modes of the source files, ignoring the present umask.

In the second form, one or more files are copied into the directory with their original file-names.

cp refuses to copy a file onto itself.

If the -i option is specified, cp will prompt the user with the name of the file whenever the copy will cause an old file to be overwritten. An answer of 'y' will cause cp to continue. Any other answer will prevent it from overwriting the file.

If the -r option is specified and any of the source files are directories, cp copies each subtree rooted at that name; in this case the destination must be a directory.

SEE ALSO
cat(1), mv(1), rcp(1C)
NAME
cpio – copy file archives in and out

SYNOPSIS
cpio -o[acBv]
cpio -i[BcdmrtuvfSb6] [ patterns ]
cpio -p[adlmuv] directory

DESCRIPTION
cpio -o (copy out) reads the standard input to obtain a list of path names and copies those files onto the standard output together with path name and status information. Output is padded to a 512-byte boundary.
cpio -i (copy in) extracts files from the standard input, which is assumed to be the product of a previous cpio -o. Only files with names that match patterns are selected. Patterns are regular expressions given in the name-generating notation of sh(1). In patterns, meta-characters ?, *, and [...] match the slash / character. Multiple patterns may be specified and if no patterns are specified, the default for patterns is * (i.e., select all files). Each pattern should be surrounded by double quotes. The extracted files are conditionally created and copied into the current directory tree based upon the options described below. The permissions of the files will be those of the previous cpio -o. The owner and group of the files will be that of the current user unless the user is super-user, which causes cpio to retain the owner and group of the files of the previous cpio -o. NOTE: If cpio -i tries to create a file that already exists and the existing file is the same age or newer, cpio will output a warning message and not replace the file. (The -u option can be used to unconditionally overwrite the existing file.)
cpio -p (pass) reads the standard input to obtain a list of path names of files that are conditionally created and copied into the destination directory tree based upon the options described below. The meanings of the available options are

a  Reset access times of input files after they have been copied. Access times are not reset for linked files when cpio -pla is specified.
B  Input/output is to be blocked 5,120 bytes to the record (does not apply to the pass option; meaningful only with data directed to or from a character special device, e.g., /dev/rmt/0m.
d  Directories are to be created as needed.
c  Write header information in ASCII character form for portability. Always use this option when origin and destination machines are different types.
r  Interactively rename files. If the user types a null line, the file is skipped. (Not available with cpio -p.)
t  Print a table of contents of the input. No files are created.
u  Copy unconditionally (normally, an older file will not replace a newer file with the same name).
v  Verbose: causes a list of file names to be printed. When used with the t option, the table of contents looks like the output of an ls -l command (see ls(1)).
l  Whenever possible, link files rather than copying them. Usable only with the -p option.
m  Retain previous file modification time. This option is ineffective on directories that are being copied.
f  Copy in all files except those in patterns.
s  Swap bytes within each half word. Use only with the -i option.
S  Swap halfwords within each word. Use only with the -i option.
B  Reverses the order of the bytes within each word. Use only with the -i option.
6  Process an old (i.e. UNIX System Sixth Edition format) file. Only useful with -i (copy in).

NOTE: cpio assumes four-byte words.
If cpio reaches end of medium (end of a diskette for example), when writing to (-o) or reading from (-i) a character special device, cpio will print the message:

If you want to go on, type device/file name when ready.
To continue, you must replace the medium and type the character special device name (/dev/tape/c0d6h for example) and carriage return. You may want to continue by directing cpio to use a different device. For example, if you have two floppy drives you may want to switch between them so cpio can proceed while you are changing the floppies. (A carriage return alone causes the cpio process to exit.)

EXAMPLES

The following examples show three uses of cpio.
When standard input is directed through a pipe to cpio -o, it groups the files so they can be directed (>) to a single file (./newfile). Instead of "ls," you could use find, echo, cat, etc. to pipe a list of names to cpio. You could direct the output to a device instead of a file.

ls | cpio -o >./newfile

cpio -i uses the output file of cpio -o (directed through a pipe with cat in the example), takes out those files that match the patterns (memo/a1, memo/b*), creates directories below the current directory as needed (-d option), and places the files in the appropriate directories. If no patterns were given, all files from "newfile" would be placed in the directory.

cat newfile | cpio -id "memo/a1" "memo/b*"

cpio -p takes the file names piped to it and copies or links (-l option) those files to another directory on your machine (newdir in the example). The -d options says to create directories as needed. The -m option says retain the modification time. (It is important to use the -depth option of find to generate path names for cpio. This eliminates problems cpio could have trying to create files under read-only directories.)

find . -depth -print | cpio -pdlmv newdir

SEE ALSO

ar(1), find(1), ls(1), tar(1), cpio(4).

NOTES

1) Path names are restricted to 256 characters.
2) Only the super-user can copy special files.
3) Blocks are reported in 512-byte quantities.
NAME
cpp – the C language preprocessor

SYNOPSIS
lib/cpp [ option ... ] [ ifile [ ofile ]]

DESCRIPTION
The C language preprocessor, cpp, is invoked as the first pass of any C compilation
by the cc(1) command. Thus cpp’s output is designed to be in a form acceptable as
input to the next pass of the C compiler. As the C language evolves, cpp and the rest
of the C compilation package will be modified to follow these changes. Therefore,
the use of cpp other than through the cc(1) command is not suggested, since the func-
tionality of cpp may someday be moved elsewhere. See m4(1) for a general macro
processor.

cpp optionally accepts two file names as arguments. Ifile and ofile are respectively the
input and output for the preprocessors. They default to standard input and standard
output if not supplied.

The following options to cpp are recognized:

-P Preprocess the input without producing the line control information used by
the next pass of the C compiler.

-C By default, cpp strips C-style comments. If the –C option is specified, all com-
ments (except those found on cpp directive lines) are passed along.

-Uname
Remove any initial definition of name, where name is a reserved symbol that is
predefined by the particular preprocessor. Following is the current list of these
reserved symbols:
operating system: unix
hardware: mips
manufacturer: ardent
language: LANGUAGE_C for C

-Dname
-Dname=def
Define name with value def as if by a #define. If no =def is given, name is
defined with value 1. The –D option has lower precedence than the –U option.
That is, if the same name is used in both a –U option and a –D option, the name
will be undefined regardless of the order of the options.

-I Do not search the standard places for #include files.

-Idir Change the algorithm for searching for #include files whose names do not
begin with / to look in dir before looking in the directories on the standard list.
Thus, #include files whose names are enclosed in "" will be searched for first in
the directory of the file with the #include line, then in directories named in –I
options, and last in directories on a standard list. For #include files whose
names are enclosed in <>, the directory of the file with the #include line is not
searched.

-H Print, one per line on standard error, the path names of included files.

-i Compute a #ident control line for the input file and every included file giving a
cyclic redundacy checksum for the file contents.

Two special names are understood by cpp. The name __LINE__ is defined as the
current line number (as a decimal integer) as known by cpp, and __FILE__ is defined
as the current file name (as a C string) as known by cpp. They can be used anywhere
(including in macros) just as any other defined name.

All cpp directive lines start with # in column 1. Any number of blanks and tabs is allowed between the # and the directive. The directives are:

```cpp
#define name token-string
    Replace subsequent instances of name with token-string.
#endif
define name( arg, ..., arg ) token-string
    Notice that there can be no space between name and the ). Replace subsequent instances of name followed by a (, a list of comma-separated sets of tokens, and a ) followed by token-string, where each occurrence of an arg in the token-string is replaced by the corresponding set of tokens in the comma-separated list. When a macro with arguments is expanded, the arguments are placed into the expanded token-string unchanged. After the entire token-string has been expanded, cpp re-starts its scan for names to expand at the beginning of the newly created token-string.
#endif
#undef name
    Cause the definition of name (if any) to be forgotten from now on. No additional tokens are permitted on the directive line after name.
#endif
define "string"
    Put string into the .comment section of an object file.
#endif
#include "filename"
#include <filename>
    Include at this point the contents of filename (which will then be run through cpp). When the <filename> notation is used, filename is only searched for in the standard places. See the -I and -Y options above for more detail. No additional tokens are permitted on the directive line after the final " or >.
#endif
#line integer-constant "filename"
    Causes cpp to generate line control information for the next pass of the C compiler. Integer-constant is the line number of the next line and filename is the file from which it comes. If "filename" is not given, the current file name is unchanged. No additional tokens are permitted on the directive line after the optional filename.
#endif
#undef name
    Ends a section of lines begun by a test directive (#if, ifndef, or #ifdef). Each test directive must have a matching #endif. No additional tokens are permitted on the directive line.
#endif
ifdef name
    The lines following will appear in the output if and only if name has been the subject of a previous #define without being the subject of an intervening #undef. No additional tokens are permitted on the directive line after name.
#endif
ifndef name
    The lines following will appear in the output if and only if name has not been the subject of a previous #define. No additional tokens are permitted on the directive line after name.
#endif
if constant-expression
    Lines following will appear in the output if and only if the constant-expression evaluates to non-zero. All binary non-assignment C operators, the ?: operator, the unary -, f, and ~ operators are all legal in constant-expression. The precedence of the operators is the same as defined by the C language. There is also a unary operator defined, which can be used in constant-expression in these two forms: defined ( name ) or 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 constant-expression. In particular, the sizeof operator is not available.

To test whether either of two symbols, foo and fum, are defined, use

```c
#if defined(foo) || defined(fum)

#elif constant-expression

An arbitrary number of #elif directives is allowed between a #if, #ifdef, or #ifndef directive and a #else or #endif directive. The lines following the #elif directive will appear in the output if and only if the preceding test directive evaluates to zero, all intervening #elif directives evaluate to zero, and the constant-expression evaluates to non-zero. If constant-expression evaluates to non-zero, all succeeding #elif and #else directives will be ignored. Any constant-expression allowed in a #if directive is allowed in a #elif directive.

#else

The lines following will appear in the output if and only if the preceding test directive evaluates to zero, and all intervening #elif directives evaluate to zero. No additional tokens are permitted on the directive line.

The test directives and the possible #else directives can be nested.

#pragma

name The name specifies a compiler optimization that is to be overridden. Additional details appear in the Programmer's Guide.

FILES

INCDIR standard directory list for #include files, usually /usr/include

LIBDIR usually /lib

SEE ALSO

c(1), lint(1), m4(1).

DIAGNOSTICS

The error messages produced by cpp are intended to be self-explanatory. The line number and file name where the error occurred are printed along with the diagnostic.

NOTES

Because the standard directory for included files may be different in different environments, this form of #include directive:

```c
#include <file.h>
```

should be used, rather than one with an absolute path, like:

```c
#include "./usr/include/file.h"
```

cpp warns about the use of the absolute pathname.
NAME

cpset – install object files in binary directories

SYNOPSIS

cpset [-o] object directory [mode owner group]

DESCRIPTION

cpset is used to install the specified object file in the given directory. The mode, owner, and group of the destination file may be specified on the command line. If this data is omitted, two results are possible:

If the user of cpset has administrative permissions (that is, the user’s numerical ID is less than 100), the following defaults are provided:

mode - 0755
owner - bin
group - bin

If the user is not an administrator, the default, owner, and group of the destination file will be that of the invoker.

An optional argument of -o will force cpset to move object to OLDobject in the destination directory before installing the new object. For example:

cpset echo /bin 0755 bin bin

cpset echo /bin

cpset echo /bin/echo

All the examples above have the same effect (assuming the user is an administrator). The file echo will be copied into /bin and will be given 0755, bin, bin as the mode, owner and group, respectively.

cpset utilises the file /usr/src/destinations to determine the final destination of a file. The locations file contains pairs of path names separated by spaces or tabs. The first name is the “official” destination (for example: /bin/echo). The second name is the new destination. For example, if echo is moved from /bin to /usr/bin, the entry in /usr/src/destinations would be:

/bin/echo    /usr/bin/echo

When the actual installation happens, cpset verifies that the “old” path name does not exist. If a file exists at that location, cpset issues a warning and continues. This file does not exist on a distribution tape; it is used by sites to track local command movement. The procedures used to build the source will be responsible for defining the “official” locations of the source.

Cross Generation

The environment variable ROOT will be used to locate the destination file (in the form $ROOT/usr/src/destination). This is necessary in the cases where cross generation is being done on a production system.

SEE ALSO

install(1M), make(1), mk(8)
NAME

crash – examine system images

SYNOPSIS

/etc/crash [ -d dumpfile ] [ -n namelist ] [ -w outputfile ] [ -m ]

DESCRIPTION

The crash command is used to examine the system memory image of a live or a crashed system by formatting and printing control structures, tables, and other information. Command line arguments to crash are dumpfile, namelist, and outputfile.

Dumpfile is the file containing the system memory image. The default dumpfile is /dev/mem. The system image can also be a regular file or disk partition of dump produced by the boot PROM sysdump command.

The text file namelist contains the symbol table information needed for symbolic access to the system memory image to be examined. The default namelist is /unix. If a system image from another machine is to be examined, the corresponding text file must be copied from that machine.

When the crash command is invoked, a session is initiated. The output from a crash session is directed to outputfile. The default outputfile is the standard output.

Input during a crash session is of the form:

    function [ argument ... ]

where function is one of the crash functions described in the "FUNCTIONS" section of this manual page, and arguments are qualifying data that indicate which items of the system image are to be printed.

The default for process-related items is the process 0 for both running systems and crashed systems. If the contents of a table are being dumped, the default is all active table entries.

The -m option is meant for running systems only. It opens the dump file with write permission so that use of the crash function modify will be allowed.

The following function options are available to crash functions wherever they are semantically valid.

- e Display every entry in a table.
- f Display the full structure.
- s process
    Specify a process slot other than the default.
- v Interpret all address arguments in the command line as virtual addresses.
- w file
    Redirect the output of a function to file.

The functions defproc and redirect correspond to the function options -s and -w. defproc sets the value of the process slot argument for subsequent functions; and redirect redirects all subsequent output.

Output from crash functions may be piped to another program in the following way:

    function [ argument ... ] ! shell_command

For example,

    mount ! grep rw
will write all mount table entries with an *rw* flag to the standard output. The redirection option (*-w*) cannot be used with this feature.

Depending on the context of the function, numeric arguments will be assumed to be in a specific radix. Counts are assumed to be decimal. Addresses are always hexadecimal. Table address arguments larger than the size of the function table will be interpreted as hexadecimal addresses; those smaller will be assumed to be decimal slots in the table. Default bases on all arguments may be overridden. The C conventions for designating the bases of numbers are recognized. A number that is usually interpreted as decimal will be interpreted as hexadecimal if it is preceded by *0x* and as octal if it is preceded by 0. Decimal override is designated by *0d*, and binary by *0b*.

Aliases for functions may be any uniquely identifiable initial substring of the function name. Traditional aliases of one letter, such as *p* for *proc*, remain valid.

Many functions accept different forms of entry for the same argument. Requests for table information will accept a table entry number, a physical address, a virtual address, a symbol, a range, or an expression. A range of slot numbers may be specified in the form *a-b* where *a* and *b* are decimal numbers. An expression consists of two operands and an operator. An operand may be an address, a symbol, or a number; the operator may be +, -, *, /, &, or |. An operand which is a number should be preceded by a radix prefix if it is not a decimal number (*0* for octal, *0x* for hexadecimal, *0b* for binary). The expression must be enclosed in parentheses (*()*). Other functions will accept any of these argument forms that are meaningful.

Two abbreviated arguments to *crash* functions are used throughout. Both accept data entered in several forms. They may be expanded into the following:

\[
\text{table_entry} = \text{table entry} | \text{address} | \text{symbol} | \text{range} | \text{expression}
\]

\[
\text{start_addr} = \text{address} | \text{symbol} | \text{expression}
\]

### FUNCTIONS

\*\*? \[-w\] file \*\*

List available functions.

\*\*!cmd \*\*

Escape to the shell to execute a command.

\*\*adv \[-e\] \*\*

\[\[-w\] file \] \[ table entry \] Print the advertise table.

\*\*base \[-w\] file \[ number \] \*\*

Print \textit{number} in binary, octal, decimal, and hexadecimal. A number in a radix other than decimal should be preceded by a prefix that indicates its radix as follows: *0x*, hexadecimal; *0*, octal; and *0b*, binary.

\*\*buffer \[-w\] file \[-format\] bufferslot \*\*

or

\*\*buffer \[-w\] file \[-format\] \[ start_addr \] \*\*

Alias: *b*.

Print the contents of a buffer in the specified format. The following format designations are recognized: *-b*, byte; *-c*, character; *-d*, decimal; *-x*, hexadecimal; *-o*, octal; *-r*, directory; and *-i*, inode. If no format is given, the previous format is used. The default format at the beginning of a *crash* session is hexadecimal.

\*\*bufhdr \[-f\] \[-w\] file \[ \] table_entry ... \*\*

Alias: *buf*.

Print system buffer headers.
help [-w file] function ...
  Print a description of the named function, including syntax and aliases.

inode [-e] [-f] [-w file] [table_entry ...]
  Alias: i.
  Print the inode table, including file system switch information.

lck [-e] [-w file] [table_entry ...]
  Alias: l.
  Print record locking information. If the -e option is used or table address arguments are given, the record lock list is printed. If no argument is entered, information on locks relative to inodes is printed.

linkblk [-e] [-w file] [table_entry ...]
  Print the linkblk table.

map [-w file] mapname ...
  Print the map structure of the given mapname.

mbfree [-w file]
  Print free streams message block headers.

mblock [-e] [-c] [-n] [-p] [-w filename] [table_entry ...]
  Print allocated streams message block headers. If the -e option is used, all message headers (including those on the free list) are printed. The -c option causes the printing to traverse the continuation pointer. Similarly, the -n option for the next block pointer and -p for the previous block pointer.

modify
  [-w file] [-mode] [-s process] [-v] start_addr Prints the current contents of the location given by start_addr in one of three modes: long (-l), short (-t), or byte (-b). It then accepts input, allowing the value to be changed. A carriage return means no change. A dot (.), or the letter q (q or Q) stops the command. This is an extremely dangerous command to use on running systems and you should exercise great care in its use.

mount [-e] [-w file] [table_entry ...]
  Alias: m.
  Print the mount table.

nm [-w file] symbol ...
  Print value and type for the given symbol.

nvram [-w file]
  Print all the non-volatile RAM variables.

  Aliases: rd, d
  Print count values starting at the start address in one of the following formats: character (-c), decimal (-d), hexadecimal (-x), octal (-o), ascii (-a), or hexadecimal/character (-h), and one of the following modes: long (-l), short (-t), or byte (-b). The default mode for character and ascii formats is byte; the default mode for decimal, hexadecimal, and octal formats is long. The format -h prints both hexadecimal and character representations of the addresses dumped; no mode needs to be specified. When format or mode is omitted, the previous value is used. At the start of a crash session, the format is hexadecimal and the mode is long. If no count is entered, 1 is assumed.

pcb [-w file] [-i start_addr]
  Print the process control block for the current process.
pdt [-e] [-w file] [-s process] section segment
or
pdt [-w file] start_addr [count]
The page descriptor table starting at the start address for count entries is printed. If no count is entered, 1 is assumed.
pfdat [-e] [-f] [-w file] [-t table_entry]
Print the pfdata table.
proc [-f] [-w file] [-p] [-t table_entry ... #procid ...]
or
proc [-f] [-w file] [-r]
Alias: p.
Print the process table. Process table information may be specified in two ways. First, any mixture of table entries and process ids may be entered. Each process id must be preceded by a #. Alternatively, process table information for runnable processes may be specified with the runnable option (-r).

putbuf
[-w file] Display the contents of the console logging buffer. All console messages produced by the kernel are logged to the console.

qrun [-w file]
Print the list of scheduled streams queues.

queue [-e] [-w file] [-t table_entry ...]
Print streams queues.

quit Alias: q.
Terminate the crash session.

r [-w file] [-f] [-e] [-i] [-m] [-g]
Print CPU registers as stored at the time of a panic. -i prints I/O board registers, -m prints memory board registers, and -g prints graphics board registers.

rcvd [-e] [-f] [-w file] [-t table_entry ...]
Print the receive descriptor table.

redirect [-w file] [-c]
or
redirect [-w file] [file]
Used with a file name, redirects output of a crash session to the named file. If no argument is given, the file name to which output is being redirected is printed. Alternatively, the close option (-c) closes the previously set file and redirects output to the standard output.

region [-e] [-f] [-w file] [-t table_entry ...]
Print the region table.

search [-w file] [-m mask] [-s process] pattern start_addr length
Print the words in memory that match pattern, beginning at the start address for length words. The mask is anded (&) with each memory word and the result compared against the pattern. The mask defaults to 0xffffffff.

semaphore
[-w file] start_addr Print the semaphore data structure for designated semaphores.
pdt [-e] [-w file] [-s process] section segment
or
pdt [-w file] start_addr [count]
The page descriptor table starting at the start address for count entries is
printed. If no count is entered, 1 is assumed.
pfdat [-e] [-f] [-w file] [table_entry ...]
Print the pfdata table.
proc [-f] [-w file] [[-p] table_entry ... #procid ...]
or
proc [-f] [-w file] [-r]
Alias: p.
Print the process table. Process table information may be specified in two ways.
First, any mixture of table entries and process ids may be entered. Each process
id must be preceded by a #. Alternatively, process table information for runnable
processes may be specified with the runnable option (-r).
putbuf
[-w file] Display the contents of the console logging buffer. All console mes-
sages produced by the kernel are logged to the console.
qrun [-w file]
Print the list of scheduled streams queues.
queue [-e] [-w file] [table_entry ...]
Print streams queues.
quit Alias: q.
Terminate the crash session.
rx [-w file] [-f] [-e] [-i] [-m] [-g]
Print CPU registers as stored at the time of a panic. -i prints I/O board regist-
ers, -m prints memory board registers, and -g prints graphics board registers.
rcvd [-e] [-f] [-w file] [table_entry ...]
Print the receive descriptor table.
redirect [-w file] [-c]
or
redirect [-w file] [file]
Used with a file name, redirects output of a crash session to the named file. If
no argument is given, the file name to which output is being redirected is
printed. Alternatively, the close option (-c) closes the previously set file and redirects output to the standard output.
region [-e] [-f] [-w file] [table_entry ...]
Print the region table.
search [-w file] [-m mask] [-s process] pattern start_addr length
Print the words in memory that match pattern, beginning at the start address
for length words. The mask is anded (&) with each memory word and the
result compared against the pattern. The mask defaults to 0xffffffff.
semaphore
[-w file] start_addr Print the semaphore data structure for designated sema-
phores.
size [-w file] [-x] [structure_name ...]
    Print the size of the designated structure. The (-x) option prints the size in hexadecimal. If no argument is given, a list of the structure names for which sizes are available is printed.

slog [-w file] Print the semaphore activity log.
sndd [-e] [-w file] [table_entry ...]
    Print the send descriptor table.
srmount [-e] [-w file] [table_entry ...]
    Print the server mount table.
stack [-w file] [-b] [-s process] [pc sp]
    Alias: s.
    Print a stack traceback of pc and sp values for the designated process. Default is the currently set defproc process. This command uses the pc and sp values from the pcb in the user area to start the traceback. These values are incorrect for processes that are active. Specify pc and sp values for stack traceback of active processes. The -b option attempts to do a reverse stack traceback starting from the first exception frame.

stat [-w file]
    Print system statistics.

stream [-e] [-f] [-w file] [table_entry ...]
    Print the streams table.

strstat [-w file]
    Print streams statistics.
ts [-w file] virtual_address ...
    Print closest text symbol to the designated address.
tty [-e] [-f] [-w file] [-t type] [table_entry ...] or

    Valid types: pp, iu.
    Print the tty table. If no arguments are given, the tty table for both tty types is printed. If the -t option is used, the table for the single tty type specified is printed. If no argument follows the type option, all entries in the table are printed. A single tty entry may be specified from the start address.

user [-f] [-w file] [process]
    Alias: u.
    Print the ublock for the designated process.

var [-w file]
    Alias: v.
    Print the tunable system parameters.

vtop [-w file] [-s process] start_addr ...
    Print the physical address translation of the virtual start address.

FILES
/dev/mem                             system image of currently running system
NAME
cron – clock daemon

SYNOPSIS
/etc/cron

DESCRIPTION
cron executes commands at specified dates and times. Regularly scheduled commands can be specified according to instructions found in crontab files in the directory /usr/spool/cron/crontabs. Users can submit their own crontab file via the crontab(1) command. Commands which are to be executed only once may be submitted via the at(1) command.

cron only examines crontab files and at command files during process initialization and when a file changes via crontab or at. This reduces the overhead of checking for new or changed files at regularly scheduled intervals.

Since cron never exits, it should be executed only once. This is done routinely through /etc/rc2.d/S75cron at system boot time. /usr/lib/cron/FIFO is used as a lock file to prevent the execution of more than one cron.

FILES
/usr/lib/cron main cron directory
/usr/lib/cron/FIFO used as a lock file
/usr/lib/cron/log accounting information
/usr/spool/cron spool area

SEE ALSO
at(1), crontab(1), sh(1), queuedefs(4)

DIAGNOSTICS
A history of all actions taken by cron are recorded in /usr/lib/cron/log.
CRONTAB (1)

NAME
crontab – user crontab file

SYNOPSIS
crontab [file]
crontab -r
crontab -l

DESCRIPTION
crontab copies the specified file, or standard input if no file is specified, into a directory that holds all users' crontabs. The -r option removes a user's crontab from the crontab directory. crontab -l will list the crontab file for the invoking user.

Users are permitted to use crontab if their names appear in the file /usr/lib/cron/cron.allow. If that file does not exist, the file /usr/lib/cron/cron.deny is checked to determine if the user should be denied access to crontab. If neither file exists, only root is allowed to submit a job. If cron.allow does not exist and cron.deny exists but is empty, global usage is permitted. The allow/deny files consist of one user name per line.

A crontab file consists of lines of six fields each. The fields are separated by spaces or tabs. The first five are integer patterns that specify the following:

- minute (0–59),
- hour (0–23),
- day of the month (1–31),
- month of the year (1–12),
- day of the week (0–6 with 0=Sunday).

Each of these patterns may be either an asterisk (meaning all legal values) or a list of elements separated by commas. An element is either a number or two numbers separated by a minus sign (meaning an inclusive range). Note that the specification of days may be made by two fields (day of the month and day of the week). If both are specified as a list of elements, both are adhered to. For example, 0 0 1,15 * 1 would run a command on the first and fifteenth of each month, as well as on every Monday. To specify days by only one field, the other field should be set to * (for example, 0 0 * * 1 would run a command only on Mondays).

The sixth field of a line in a crontab file is a string that is executed by the shell at the specified times. A percent character in this field (unless escaped by \) is translated to a new-line character. Only the first line (up to a % or end of line) of the command field is executed by the shell. The other lines are made available to the command as standard input.

The shell is invoked from your $HOME directory with an arg0 of sh. Users who desire to have their .profile executed must explicitly do so in the crontab file. Cron supplies a default environment for every shell, defining HOME, LOGNAME, SHELL(=/bin/sh), and PATH(=/bin/usr/bin/usr/lib).

If you do not redirect the standard output and standard error of your commands, any generated output or errors will be mailed to you.

FILES
/usr/lib/cron
/usr/spool/cron/crontabs
/usr/lib/cron/log
/usr/lib/cron/cron.allow
/usr/lib/cron/cron.deny
main crontab directory
spool area
accounting information
list of allowed users
list of denied users
SEE ALSO

sh(1), cron(1M), queuedefs(4)

WARNINGS

If you inadvertently enter the crontab command with no argument(s), do not attempt to get out with a CTRL-d. This will cause all entries in your crontab file to be removed. Instead, exit with a DEL.
NAME

crypt – encode/decode

SYNOPSIS

crypt [ password ]
crypt [-k]

DESCRIPTION

crypt reads from the standard input and writes on the standard output. The password is a key that selects a particular transformation. If no argument is given, crypt demands a key from the terminal and turns off printing while the key is being typed in. If the -k option is used, crypt will use the key assigned to the environment variable CRYPTKEY. crypt encrypts and decrypts with the same key:

    crypt key <clear >cypher
    crypt key <cypher | pr

Files encrypted by crypt are compatible with those treated by the editors ed(1), edit(1), ex(1), and vi(1) in encryption mode.

The security of encrypted files depends on three factors: the fundamental method must be hard to solve; direct search of the key space must be infeasible; “sneak paths” by which keys or clear text can become visible must be minimized.

crypt implements a one-rotor machine designed along the lines of the German Enigma, but with a 256-element rotor. Methods of attack on such machines are known, but not widely; moreover the amount of work required is likely to be large.

The transformation of a key into the internal settings of the machine is deliberately designed to be expensive, i.e., to take a substantial fraction of a second to compute. However, if keys are restricted to (say) three lower-case letters, then encrypted files can be read by expending only a substantial fraction of five minutes of machine time.

If the key is an argument to the crypt command, it is potentially visible to users executing ps(1) or a derivative. The choice of keys and key security are the most vulnerable aspect of crypt.

FILES

/dev/tty for typed key

SEE ALSO

ed(1), edit(1), ex(1), makekey(1), ps(1), stty(1), vi(1).

WARNING

This command is provided with the Security Administration Utilities, which is only available in the United States. If two or more files encrypted with the same key are concatenated and an attempt is made to decrypt the result, only the contents of the first of the original files will be decrypted correctly.

BUGS

If output is piped to nroff and the encryption key is not given on the command line, crypt can leave terminal modes in a strange state (see stty(1)).
NAME
csh – a shell (command interpreter) with C-like syntax

SYNOPSIS
csh [-cefinstvVxX] [ arg ... ]

DESCRIPTION

Csh is a first implementation of a command language interpreter incorporating a history mechanism (see History Substitutions), job control facilities (see Jobs), interactive file name and user name completion (see File Name Completion), and a C-like syntax. So as to be able to use its job control facilities, users of csh must (and automatically) use the new tty driver fully described in tty(4). This new tty driver allows generation of interrupt characters from the keyboard to tell jobs to stop. See stty(1B) for details on setting options in the new tty driver.

An instance of csh begins by executing commands from the file ‘.cshrc’ in the home directory of the invoker. If this is a login shell then it also executes commands from the file ‘.login’ there. It is typical for users on crt’s to put the command ‘"stty crt"’ in their .login file, and to also invoke tset(1) there.

In the normal case, the shell will then begin reading commands from the terminal, prompting with ‘%’. Processing of arguments and the use of the shell to process files containing command scripts will be described later.

The shell then repeatedly performs the following actions: a line of command input is read and broken into words. This sequence of words is placed on the command history list and then parsed. Finally each command in the current line is executed.

When a login shell terminates it executes commands from the file ‘.logout’ in the users home directory.

Lexical structure

The shell splits input lines into words at blanks and tabs with the following exceptions. The characters ‘&’ ‘!’ ‘?’ ‘<’ ‘>’ ‘(’ ‘)’ form separate words. If doubled in ‘&&’, ‘! !’, ‘<<’ or ‘>>’ these pairs form single words. These parser metacharacters may be made part of other words, or prevented their special meaning, by preceding them with ‘\’. A newline preceded by a ‘\’ is equivalent to a blank.

In addition strings enclosed in matched pairs of quotations, ‘"’ ‘’’ or ‘”’ form parts of a word; metacharacters in these strings, including blanks and tabs, do not form separate words. These quotations have semantics to be described subsequently. Within pairs of ‘’ or ‘”’ characters a newline preceded by a ‘\’ gives a true newline character.

When the shell’s input is not a terminal, the character ‘#’ introduces a comment which continues to the end of the input line. It is prevented this special meaning when preceded by ‘\’ and in quotations using ‘”’ ‘”’ and ‘”’.

Commands

A simple command is a sequence of words, the first of which specifies the command to be executed. A simple command or a sequence of simple commands separated by ‘!’ characters forms a pipeline. The output of each command in a pipeline is connected to the input of the next. Sequences of pipelines may be separated by ‘;’, and are then executed sequentially. A sequence of pipelines may be executed without immediately waiting for it to terminate by following it with an ‘&’.

Any of the above may be placed in ‘(’ ‘)’ to form a simple command (which may be a component of a pipeline, etc.) It is also possible to separate pipelines with ‘! ’ or ‘&’ indicating, as in the C language, that the second is to be executed only if the first fails or succeeds respectively. (See Expressions.)
Jobs

The shell associates a job with each pipeline. It keeps a table of current jobs, printed by the jobs command, and assigns them small integer numbers. When a job is started asynchronously with ‘&’, the shell prints a line which looks like:

[1] 1234

indicating that the job which was started asynchronously was job number 1 and had one (top-level) process, whose process id was 1234.

If you are running a job and wish to do something else you may hit the key ^Z (control-Z) which sends a STOP signal to the current job. The shell will then normally indicate that the job has been ‘Stopped’, and print another prompt. You can then manipulate the state of this job, putting it in the background with the bg command, or run some other commands and then eventually bring the job back into the foreground with the foreground command fg. A ^Z takes effect immediately and is like an interrupt in that pending output and unread input are discarded when it is typed. There is another special key ^Y which does not generate a STOP signal until a program attempts to read(2) it. This can usefully be typed ahead when you have prepared some commands for a job which you wish to stop after it has read them.

A job being run in the background will stop if it tries to read from the terminal. Background jobs are normally allowed to produce output, but this can be disabled by giving the command ‘stty tostop’. If you set this tty option, then background jobs will stop when they try to produce output like they do when they try to read input.

There are several ways to refer to jobs in the shell. The character ‘%’ introduces a job name. If you wish to refer to job number 1, you can name it as ‘%1’. Just naming a job brings it to the foreground; thus ‘%1’ is a synonym for ‘fg %1’, bringing job 1 back into the foreground. Similarly saying ‘%1 &’ resumes job 1 in the background. Jobs can also be named by prefixes of the string typed in to start them, if these prefixes are unambiguous, thus ‘%ex’ would normally restart a suspended ex(1) job, if there were only one suspended job whose name began with the string ‘ex’. It is also possible to say ‘%?string’ which specifies a job whose text contains string, if there is only one such job.

The shell maintains a notion of the current and previous jobs. In output pertaining to jobs, the current job is marked with a ‘+’ and the previous job with a ‘−’. The abbreviation ‘%+’ refers to the current job and ‘%-’ refers to the previous job. For close analogy with the syntax of the history mechanism (described below), ‘%%’ is also a synonym for the current job.

Status reporting

This shell learns immediately whenever a process changes state. It normally informs you whenever a job becomes blocked so that no further progress is possible, but only just before it prints a prompt. This is done so that it does not otherwise disturb your work. If, however, you set the shell variable notify, the shell will notify you immediately of changes of status in background jobs. There is also a shell command notify which marks a single process so that its status changes will be immediately reported. By default notify marks the current process; simply say ‘notify’ after starting a background job to mark it.

When you try to leave the shell while jobs are stopped, you will be warned that ‘You have stopped jobs.’ You may use the jobs command to see what they are. If you do this or immediately try to exit again, the shell will not warn you a second time, and the suspended jobs will be terminated.
File Name Completion

When the file name completion feature is enabled by setting the shell variable filec (see set), csh will interactively complete file names and user names from unique prefixes, when they are input from the terminal followed by the escape character (the escape key, or control-D). For example, if the current directory looks like

<table>
<thead>
<tr>
<th>Directory</th>
<th>File</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSC.OLD</td>
<td>bin</td>
</tr>
<tr>
<td>DSC.NEW</td>
<td>chaosnet</td>
</tr>
<tr>
<td>bench</td>
<td>class</td>
</tr>
<tr>
<td></td>
<td>cmd</td>
</tr>
<tr>
<td></td>
<td>lib</td>
</tr>
<tr>
<td></td>
<td>xmpl.c</td>
</tr>
<tr>
<td></td>
<td>cmdtest</td>
</tr>
<tr>
<td></td>
<td>mail</td>
</tr>
<tr>
<td></td>
<td>xmpl.o</td>
</tr>
<tr>
<td></td>
<td>dev</td>
</tr>
<tr>
<td></td>
<td>mbox</td>
</tr>
<tr>
<td></td>
<td>xmpl.out</td>
</tr>
</tbody>
</table>

and the input is

```
% vi ch<escape>
```

*csh* will complete the prefix "ch" to the only matching file name "chaosnet", changing the input line to

```
% vi chaosnet
```

However, given

```
% vi D<escape>
```

*csh* will only expand the input to

```
% vi DSC.
```

and will sound the terminal bell to indicate that the expansion is incomplete, since there are two file names matching the prefix "D".

If a partial file name is followed by the end-of-file character (usually control-D), then, instead of completing the name, *csh* will list all file names matching the prefix. For example, the input

```
% vi D<control-D>
```

causes all files beginning with "D" to be listed:

<table>
<thead>
<tr>
<th>Directory</th>
<th>File</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSC.NEW</td>
<td>DSC.OLD</td>
</tr>
</tbody>
</table>

while the input line remains unchanged.

The same system of escape and end-of-file can also be used to expand partial user names, if the word to be completed (or listed) begins with the character "~". For example, typing

```
   cd ~ro<control-D>
```

may produce the expansion

```
   cd ~root
```

The use of the terminal bell to signal errors or multiple matches can be inhibited by setting the variable no beep.

Normally, all files in the particular directory are candidates for name completion. Files with certain suffixes can be excluded from consideration by setting the variablefignore to the list of suffixes to be ignored. Thus, if *fignore* is set by the command

```
% set fignore = (.o .out)
```

then typing

```
   % vi x<escape>
```

would result in the completion to

```
   % vi xmpl.c
```

ignoring the files "xmpl.o" and "xmpl.out". However, if the only completion possible requires not ignoring these suffixes, then they are not ignored. In addition, *fignore* does not affect the listing of file names by control-D. All files are listed regardless of their suffixes.

Substitutions

We now describe the various transformations the shell performs on the input in the order in which they occur.
History substitutions

History substitutions place words from previous command input as portions of new commands, making it easy to repeat commands, repeat arguments of a previous command in the current command, or fix spelling mistakes in the previous command with little typing and a high degree of confidence. History substitutions begin with the character ‘’ and may begin anywhere in the input stream (with the proviso that they do not nest.) This ‘’ may be preceded by an ‘\’ to prevent its special meaning; for convenience, a ‘’ is passed unchanged when it is followed by a blank, tab, new-line, ‘=’ or ‘.’. (History substitutions also occur when an input line begins with ↑. This special abbreviation will be described later.) Any input line which contains history substitution is echoed on the terminal before it is executed as it could have been typed without history substitution.

Commands input from the terminal which consist of one or more words are saved on the history list. The history substitutions reintroduce sequences of words from these saved commands into the input stream. The size of which is controlled by the history variable; the previous command is always retained, regardless of its value. Commands are numbered sequentially from 1.

For definiteness, consider the following output from the history command:

```
9  write michael
10  ex write.c
11  cat oldwrite.c
12  diff *write.c
```

The commands are shown with their event numbers. It is not usually necessary to use event numbers, but the current event number can be made part of the prompt by placing an ‘’ in the prompt string.

With the current event 13 we can refer to previous events by event number ‘111’, relatively as in ‘!-2’ (referring to the same event), by a prefix of a command word as in ‘id’ for event 12 or ‘wri’ for event 9, or by a string contained in a word in the command as in ‘?mic?’ also referring to event 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; thus ‘!!’ alone is essentially a redo.

To select words from an event we can follow the event specification by a ‘:’ and a designator for the desired words. The words of an input line are numbered from 0, the first (usually command) word being 0, the second word (first argument) being 1, etc. The basic word designators are:

```
0   first (command) word
n   n'th argument
↑   first argument, i.e. ‘1’
$   last argument
%   word matched by (immediately preceding) ?$ ? search
x-y  range of words
-   abbreviates ‘0-y’
*   abbreviates ‘↑-$’, or nothing if only 1 word in event
x*   abbreviates ‘x-$’
x−   like ‘x*’ but omitting word ‘$’
```

The ‘:’ separating the event specification from the word designator can be omitted if the argument selector begins with a ‘↑’, ‘$’, ‘*‘ ‘−’ or ‘%’. After the optional word designator can be placed a sequence of modifiers, each preceded by a ‘.’. The following modifiers are defined:
h  Remove a trailing pathname component, leaving the head.
r  Remove a trailing ‘.xxx’ component, leaving the root name.
e  Remove all but the extension ‘.xxx’ part.
s/l/r/  Substitute l for r
t  Remove all leading pathname components, leaving the tail.
&  Repeat the previous substitution.
g  Apply the change globally, prefixing the above, e.g. ‘g&’.
p  Print the new command but do not execute it.
q  Quote the substituted words, preventing further substitutions.
x  Like q, but break into words at blanks, tabs and newlines.

Unless preceded by a ‘g’ the modification is applied only to the first modifiable word. With substitutions, it is an error for no word to be applicable.

The left hand side of substitutions are not regular expressions in the sense of the editors, but rather strings. Any character may be used as the delimiter in place of ‘/’; a ‘\’ quotes the delimiter into the l and r strings. The character ‘&’ in the right hand side is replaced by the text from the left. A ‘\’ quotes ‘&’ also. A null l uses the previous string either from a l or from a contextual scan string s in ‘?s?’.

A history reference may be given without an event specification, e.g. ‘!$’. In this case the reference is to the previous command unless a previous history reference occurred on the same line in which case this form repeats the previous reference. Thus ‘!?foo!$’ gives the first and last arguments from the command matching ‘?foo?’.

A special abbreviation of a history reference occurs when the first non-blank character of an input line is a ‘%’. This is equivalent to ‘!s%’ providing a convenient shorthand for substitutions on the text of the previous line. Thus ‘!lib%lib’ fixes the spelling of ‘lib’ in the previous command. Finally, a history substitution may be surrounded with ‘{‘ and ‘}’ if necessary to insulate it from the characters which follow. Thus, after ‘ls -ld ~paul’ we might do ‘{[l]a}’ to do ‘ls -ld ~paula’, while ‘[l]a’ would look for a command starting ‘la’.

Quotations with ‘ ’ and ‘”’

The quotation of strings by ‘”’ and ‘”’ can be used to prevent all or some of the remaining substitutions. Strings enclosed in ‘”’ are prevented any further interpretation. Strings enclosed in ‘”’ may be expanded as described below.

In both cases the resulting text becomes (all or part of) a single word; only in one special case (see Command Substitution below) does a ‘”’ quoted string yield parts of more than one word; ‘”’ quoted strings never do.

Alias substitution

The shell maintains a list of aliases which can be established, displayed and modified by the alias and unalias commands. After a command line is scanned, it is parsed into distinct commands and the first word of each command, left-to-right, is checked to see if it has an alias. If it does, then the text which is the alias for that command is reread with the history mechanism available as though that command were the previous input line. The resulting words replace the command and argument list. If no reference is made to the history list, then the argument list is left unchanged.

Thus if the alias for ‘ls’ is ‘ls –l’ the command ‘ls /usr’ would map to ‘ls –l /usr’, the argument list here being undisturbed. Similarly if the alias for ‘lookup’ was ‘grep !’ /etc/passwd’ then ‘lookup bill’ would map to ‘grep bill /etc/passwd’.
If an alias is found, the word transformation of the input text is performed and the aliasing process begins again on the reformed input line. Looping is prevented if the first word of the new text is the same as the old by flagging it to prevent further aliasing. Other loops are detected and cause an error.

Note that the mechanism allows aliases to introduce parser metasyntax. Thus we can ‘alias print ‘pr !+ 1 lpr’’ to make a command which pr’s its arguments to the line printer.

**Variable substitution**

The shell maintains a set of variables, each of which has as 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’s argument list, and words of this variable’s value are referred to in special ways.

The values of variables may be displayed and changed by using the `set` and `unset` commands. Of the variables referred to by the shell a number are toggles; the shell does not care what their value is, only whether they are set or not. For instance, the `verbose` variable is a toggle which causes command input to be echoed. The setting of this variable results from the `-v` command line option.

Other operations treat variables numerically. The `@` command permits numeric calculations to be performed and the result assigned to a variable. Variable values are, however, always represented as (zero or more) strings. For the purposes of numeric operations, the null string is considered to be zero, and the second and subsequent words of multiword values are ignored.

After the input line is aliased and parsed, and before each command is executed, variable substitution is performed key by `$` characters. This expansion can be prevented by preceding the `$` with a `'` except within `"`s where it *always* occurs, and within `"`s where it *never* occurs. Strings quoted by `''` are interpreted later (see Command substitution below) so `$` substitution does not occur there until later, if at all. A `$` is passed unchanged if followed by a blank, tab, or end-of-line.

Input/output redirections are recognized before variable expansion, and are variable expanded separately. Otherwise, the command name and entire argument list are expanded together. It is thus possible for the first (command) word to this point to generate more than one word, the first of which becomes the command name, and the rest of which become arguments.

Unless enclosed in `"` or given the `’q’` modifier the results of variable substitution may eventually be command and filename substituted. Within `"`, a variable whose value consists of multiple words expands to a (portion of) a single word, with the words of the variables value separated by blanks. When the `’q’` modifier is applied to a substitution the variable will expand to multiple words with each word separated by a blank and quoted to prevent later command or filename substitution.

The following metasequences are provided for introducing variable values into the shell input. Except as noted, it is an error to reference a variable which is not set.

```bash
$name
$[name]
```

Are replaced by the words of the value of variable *name*, each separated by a blank. Braces insulate *name* from following characters which would otherwise be part of it. Shell variables have names consisting of up to 20 letters and digits starting with a letter. The underscore character is considered a letter.

If *name* is not a shell variable, but is set in the environment, then that value is returned (but : modifiers and the other forms given below are not available in this case).
$\text{name}[\text{selector}]
$\{\text{name}[\text{selector}]\}

May be used to select only some of the words from the value of \text{name}. The selector is subjected to `$` substitution and may consist of a single number or two numbers separated by a `:`. The first word of a variables value is numbered `1`. If the first number of a range is omitted it defaults to `1`. If the last member of a range is omitted it defaults to `$\#\text{name}`. The selector `*` selects all words. It is not an error for a range to be empty if the second argument is omitted or in range.

$\#\text{name}
$\{\#\text{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.

$\text{number}
$\{\text{number}\}

Equivalent to `$\text{argv}[\text{number}]$`.

$*

Equivalent to `$\text{argv}[\ast]$`.

The modifiers `h`, `:t`, `:r`, `:q` and `:x` may be applied to the substitutions above as may `gh`, `gt` and `gr`. If braces `{` `}` appear in the command form then the modifiers must appear within the braces. The \textit{current implementation allows only one `:` modifier on each `$` expansion.}

The following substitutions may not be modified with `:` modifiers.

$?\text{name}
$\{?\text{name}\}

Substitutes the string `1` if name is set, `0` if it is not.

$?0

Substitutes `1` if the current input filename is known, `0` if it is not.

$$

Substitute the (decimal) process number of the (parent) shell.

$<$

Substitutes a line from the standard input, with no further interpretation thereafter. It can be used to read from the keyboard in a shell script.

\textbf{Command and filename substitution}

The remaining substitutions, command and filename substitution, are applied selectively to the arguments of built-in commands. This means that portions of expressions which are not evaluated are not subjected to these expansions. For commands which are not internal to the shell, the command name is substituted separately from the argument list. This occurs very late, after input-output redirection is performed, and in a child of the main shell.

\textbf{Command substitution}

Command substitution is indicated by a command enclosed in `^`. The output from such a command is normally broken into separate words at blanks, tabs and newlines, with null words being discarded, this text then replacing the original string. Within `"`'s, only newlines force new words; blanks and tabs are preserved.
In any case, the single final newline does not force a new word. Note that it is thus possible for a command substitution to yield only part of a word, even if the command outputs a complete line.

Filename substitution

If a word contains any of the characters ‘*’, ‘?’, ‘!’ or ‘ ‘ or begins with the character ‘~’, then that word is a candidate for filename substitution, also known as ‘globbing’. This word is then regarded as a pattern, and replaced with an alphabetically sorted list of file names which match the pattern. In a list of words specifying filename substitution it is an error for no pattern to match an existing file name, but it is not required for each pattern to match. Only the metacharacters ‘*’, ‘?’ and ‘!’ imply pattern matching, the characters ‘~’ and ‘ ‘ being more akin to abbreviations.

In matching filenames, the character ‘~’ at the beginning of a filename or immediately following a ‘/’, as well as the character ‘!’ must be matched explicitly. The character ‘*’ matches any string of characters, including the null string. The character ‘?’ matches any single character. The sequence ‘[...’ matches any one of the characters enclosed. Within ‘[...]’, a pair of characters separated by ‘~’ matches any character lexically between the two.

The character ‘~’ at the beginning of a filename is used to refer to home directories. Standing alone, i.e. ‘~’ it expands to the invokers home directory as reflected in the value of the variable home. When followed by a name consisting of letters, digits and ‘~’ 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 not at the beginning of a word, it is left undisturbed.

The metacharacter ‘[a,b,c,d]e’ is a shorthand for ‘abe ace ade’. Left to right order is preserved, with results of matches being sorted separately at a low level to preserve this order. This construct may be nested. Thus ‘~source/s1/(oldls,s).c’ expands to ‘/usr/source/s1/oldls.c /usr/source/s1ls.c’ whether or not these files exist without any chance of error if the home directory for ‘source’ is ‘/usr/source’. Similarly ‘/.../memo,*box’ might expand to ‘/.../memo .../box .../mbox’ (Note that ‘memo’ was not sorted with the results of matching ‘*box’). As a special case ‘[!’, ‘!’ and ‘!’ are passed undisturbed.

Input/output

The standard input and standard output of a command may be redirected with the following syntax:

```
< name
 Open file name (which is first variable, command and filename expanded) as
 the standard input.

<< word
 Read the shell input up to a line which is identical to word. Word is not
 subjected to variable, filename or command substitution, and each input line is
 compared to word before any substitutions are done on this input line. Unless a
 quoting ‘\’, ‘"’, ‘’’ or ‘’ appears in word variable and command substitution is
 performed on the intervening lines, allowing ‘\’ to quote ‘$’, ‘\’ and ‘’’. Commands
 which are substituted have all blanks, tabs, and newlines preserved, except for the
 final newline which is dropped. The resultant text is placed in an
 anonymous temporary file which is given to the command as standard input.

> name
```
The file _name_ is used as standard output. If the file does not exist then it is created; if the file exists, its is truncated, its previous contents being lost.

If the variable _noclobber_ is set, then the file must not exist or be a character special file (e.g. a terminal or '/dev/null') or an error results. This helps prevent accidental destruction of files. In this case the '!' forms can be used and suppress this check.

The forms involving '&' route the diagnostic output into the specified file as well as the standard output. _Name_ is expanded in the same way as '<' input filenames are.

Uses file _name_ as standard output like '>' but places output at the end of the file. If the variable _noclobber_ is set, then it is an error for the file not to exist unless one of the '!' forms is given. Otherwise similar to '>'.

A command receives the environment in which the shell was invoked as modified by the input-output parameters and the presence of the command in a pipeline. Thus, unlike some previous shells, commands run from a file of shell commands have no access to the text of the commands by default; rather they receive the original standard input of the shell. The '<< ' mechanism should be used to present inline data. This permits shell command scripts to function as components of pipelines and allows the shell to block read its input. Note that the default standard input for a command run detached is not modified to be the empty file '/dev/null'; rather the standard input remains as the original standard input of the shell. If this is a terminal and if the process attempts to read from the terminal, then the process will block and the user will be notified (see _Jobs_ above).

Diagnostic output may be directed through a pipe with the standard output. Simply use the form 'I &' rather than just 'I'.

**Expressions**

A number of the builtin commands (to be described subsequently) take expressions, in which the operators are similar to those of C, with the same precedence. These expressions appear in the @, _exit_, _if_, and _while_ commands. The following operators are available:

```
| | && | ! | += | -= | /= | %= | ! = ( )
```

Here the precedence increases to the right, '==' '!=' '=/=' and '!-=' and '!=-', '<=','=' '>=' '<' and '>', '<<=' and '>=', '+=' and '->', '+=' and '->/' and '%=' being, in groups, at the same level. The '==' '!==' '=/=' and '!-=' operators compare their arguments as strings; all others operate on numbers. The operators '==-' and '!-' are like '!=' and '==' except that the right hand side is a pattern (containing, e.g. '*s', '?s and instances of '[...]' against which the left hand operand is matched. This reduces the need for use of the _switch_ statement in shell scripts 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 no two components of an expression can appear in the same word; except when adjacent to components of expressions which are syntactically significant to the parser ('&' 'I' '<' '> '')('). They should be
surrounded by spaces.

Also available in expressions as primitive operands are command executions enclosed in '{' and '}' and file enquiries of the form '{-I name' where I is one of:

r read access
w write access
x execute access
e existence
o ownership
z zero size
f plain file
d directory
l symbolic link

The specified name is command and filename 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 enquiries return false, i.e. '0'. Command executions succeed, returning true, i.e. '1', if the command exits with status 0, otherwise they fail, returning false, i.e. '0'. If more detailed status information is required then the command should be executed outside of an expression and the variable status examined.

Control flow

The shell contains a number of commands which can be used to regulate the flow of control in command files (shell scripts) and (in limited but useful ways) from terminal input. These commands all operate by forcing the shell to reread or skip in its input and, due to the implementation, restrict the placement of some of the commands.

The foreach, switch, and while statements, as well as the if–then–else form of the if statement require that the major keywords appear in a single simple command on an input line as shown below.

If the shell's input is not seekable, the shell buffers up input whenever a loop is being read and performs seeks in this internal buffer to accomplish the rereading implied by the loop. (To the extent that this allows, backward goto's will succeed on non-seekable inputs.)

Built-in commands

Built-in commands are executed within the shell. If a built-in command occurs as any component of a pipeline except the last then it is executed in a subshell.

alias
alias name
alias name wordlist

The first form prints all aliases. The second form prints the alias for name. The final form assigns the specified wordlist as the alias of name; wordlist is command and filename substituted. Name is not allowed to be alias or unalias.

alloc

Shows the amount of dynamic memory acquired, broken down into used and free memory. With an argument shows the number of free and used blocks in each size category. The categories start at size 8 and double at each step. This command's output may vary across system types, since systems other than the VAX may use a different memory allocator.

bg
bg %job ...
   Puts the current or specified jobs into the background, continuing them if they were stopped.
break
   Causes execution to resume after the end of the nearest enclosing foreach or while. The remaining commands on the current line are executed. Multi-level breaks are thus possible by writing them all on one line.
breaksw
   Causes a break from a switch, resuming after the endsw.
case label:
   A label in a switch statement as discussed below.
cd
cd name
chdir
chdir name
   Change the shell's working directory to directory name. If no argument is given then change to the home directory of the user.
   If name is not found as a subdirectory of the current directory (and does not begin with '/','.' or '...'), then each component of the variable cpath is checked to see if it has a subdirectory name. Finally, if all else fails but name is a shell variable whose value begins with '/', then this is tried to see if it is a directory.
continue
   Continue execution of the nearest enclosing while or foreach. The rest of the commands on the current line are executed.
default:
   Labels the default case in a switch statement. The default should come after all case labels.
dirs
   Prints the directory stack; the top of the stack is at the left, the first directory in the stack being the current directory.
echo wordlist
echo -n wordlist
   The specified words are written to the shells standard output, separated by spaces, and terminated with a newline unless the -n option is specified.
else
end
endif
endsw
   See the description of the foreach, if, switch, and while statements below.
eval arg ...
   (As in sh(1).) The arguments are read as input to the shell and the resulting command(s) executed in the context of the current shell. This is usually used to execute commands generated as the result of command or variable substitution, since parsing occurs before these substitutions. See tset(1) for an example of using eval.
eexec command
   The specified command is executed in place of the current shell.
exit
exit(expr)
The shell exits either with the value of the status variable (first form) or with
the value of the specified expr (second form).

fg
fg %job ...
Brings the current or specified jobs into the foreground, continuing them if they
were stopped.

foreach name (wordlist)
...
end
The variable name is successively set to each member of wordlist and the
sequence of commands between this command and the matching end are ex-
cuted. (Both foreach and end must appear alone on separate lines.)
The builtin command continue may be used to continue the loop prematurely
and the builtin command break to terminate it prematurely. When this com-
mand is read from the terminal, the loop is read up once prompting with ‘?’
before any statements in the loop are executed. If you make a mistake typing in
a loop at the terminal you can rub it out.

glob wordlist
Like echo but no ‘\’ escapes are recognized and words are delimited by null
characters in the output. Useful for programs which wish to use the shell to
filename expand a list of words.

goto word
The specified word is filename 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. Execution contin-
ues after the specified line.

history
history n
history -r n
history -h n
Displays the history event list; if n is given only the n most recent events are
printed. The -r option reverses the order of printout to be most recent first
rather than oldest first. The -h option causes the history list to be printed
without leading numbers. This is used to produce files suitable for sourcere
using the -h option to source.

if (expr) command
If the specified expression evaluates true, then the single command with argu-
ments is executed. Variable substitution on command happens early, at the
same time it does for the rest of the if command. Command must be a simple
command, not a pipeline, a command list, or a parenthesized command list.
Input/output redirection occurs even if expr is false, when command is not exec-
cuted (this is a bug).

if (expr) then
...
else if (expr2) then
...
else
... 

`endif`

If the specified `expr` is true then the commands to the first `else` are executed; otherwise if `expr2` is true then the commands to the second `else` are executed, etc. Any number of `else-if` pairs are possible; only one `endif` is needed. The `else` part is likewise 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; given the `-l` options lists process id’s in addition to the normal information.

`kill %job`
`kill -sig %job ...`
`kill pid`
`kill -sig pid ...`
`kill -l`

Sends either the TERM (terminate) signal or the specified signal to the specified jobs or processes. Signals are either given by number or by names (as given in `/usr/include/signal.h`, stripped of the prefix “SIG”). The signal names are listed by “kill -l”. There is no default, saying just ‘kill’ does not send a signal to the current job. If the signal being sent is TERM (terminate) or HUP (hangup), then the job or process will be sent a CONT (continue) signal as well.

`limit`
`limit resource`
`limit resource maximum-use`
`limit -h`
`limit -h resource`
`limit -h resource maximum-use`

NOTE: This command is not supported on the Stardent 1500/3000. All limits are set to zero. You can set any of the limits, but you do nothing but modify some local variables.

Limits the consumption by the current process and each process it creates to not individually exceed `maximum-use` on the specified `resource`. If no `maximum-use` is given, then the current limit is printed; if no `resource` is given, then all limitations are given. If the `-h` flag is given, the hard limits are used instead of the current limits. The hard limits impose a ceiling on the values of the current limits. Only the super-user may raise the hard limits, but a user may lower or raise the current limits within the legal range.

Resources controllable currently include `cputime` (the maximum number of cpu-seconds to be used by each process), `filesize` (the largest single file which can be created), `datasize` (the maximum growth of the data+stack region via `sbrk(2)` beyond the end of the program text), `stacksize` (the maximum size of the automatically-extended stack region), and `coredumpsize` (the size of the largest core dump that will be created).

The `maximum-use` may be given as a (floating point or integer) number followed by a scale factor. For all limits other than `cputime` the default scale is ‘K’ or ‘kilo-bytes’ (1024 bytes); a scale factor of ‘m’ or ‘megabytes’ may also be used. For `cputime` the default scaling is ‘seconds’, while ‘m’ for minutes or ‘h’ for hours, or a time of the form ‘mm:ss’ giving minutes and seconds may be used.
For both resource names and scale factors, unambiguous prefixes of the names suffice.

**login**

Terminate a login shell, replacing it with an instance of `/bin/login`. This is one way to log off, included for compatibility with `sh(1)`.

**logout**

Terminate a login shell. Especially useful if `ignoreeof` is set.

**nice**

**nic**e **+number**

**nice** **command**

**nice** **+number** **command**

The first form sets the scheduling priority for this shell to 4. The second form sets the priority to the given number. The final two forms run command at priority 4 and `number` respectively. The greater the number, the less cpu the process will get. The super-user may specify negative priority by using `nice -number ...`. Command is always executed in a sub-shell, and the restrictions placed on commands in simple `if` statements apply.

**nohup**

**nohup** **command**

The first form can be used in shell scripts to cause hangups to be ignored for the remainder of the script. The second form causes the specified command to be run with hangups ignored. All processes detached with `&` are effectively `nohup`ed.

**notify**

**notify** %job ...  

Causes the shell to notify the user asynchronously when the status of the current or specified jobs changes; normally notification is presented before a prompt. This is automatic if the shell variable `notify` is set.

**onintr**

**onintr** –

**onintr label**

Control the action of the shell on interrupts. The first form restores the default action of the shell on interrupts which is to terminate shell scripts or to return to the terminal command input level. The second form `onintr -` causes all interrupts to be ignored. The final form causes the shell to execute a `goto label` when an interrupt is received or a child process terminates because it was interrupted.

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, returning to the new top directory. With an argument `'+n' discards the `n`th entry in the stack. The elements of the directory stack are numbered from 0 starting at the top.

**pushd**

**pushd** name

**pushd** **+n**

With no arguments, `pushd` exchanges the top two elements of the directory stack. Given a name argument, `pushd` changes to the new directory (ala `cd`) and
pushes the old current working directory (as in csw) 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 variable to be recomputed. This is needed if new commands are added to directories in the path while you are logged in. This should only be necessary if you add commands to one of your own directories, or if a systems programmer changes the contents of one of the system directories.

**repeat count command**

The specified command which is subject to the same restrictions as the command in the one line if statement above, is executed count times. I/O redirections occur exactly once, even if count is 0.

**set**

**set name**

**set name=word**

**set name[index]=word**

**set name=(wordlist)**

The first form of the command shows the value of all shell variables. Variables which have other than a single word as value print as a parenthesized word list. The second form sets name to the null string. The third form sets name to the single word. The fourth 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 wordlist. In all cases the value is command and filename expanded.

These arguments may be repeated to set multiple values in a single set command. Note however, that variable expansion happens for all arguments before any setting occurs.

**setenv**

**setenv name value**

**setenv name**

The first form lists all current environment variables. The last form sets the value of environment variable name to be value, a single string. The second form sets name to an empty string. The most commonly used environment variable 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.

**shift**

**shift variable**

The members of argv are shifted to the left, discarding argv[1]. It is an error for argv not to be set or to have less than one word as value. The second form performs the same function on the specified variable.

**source name**

**source -h name**

The shell reads commands from name. Source commands may be nested; if they are nested too deeply the shell may run out of file descriptors. An error in a source at any level terminates all nested source commands. Normally input during source commands is not placed on the history list; the -h option causes the commands to be placed in the history list without being executed.
stop
stop %job ...
   Stops the current or specified job which is executing in the background.

suspend
   Causes the shell to stop in its tracks, much as if it had been sent a stop signal
   with ^Z. This is most often used to stop shells started by su(1).

switch (string)
case str1:
   ...
breaksw
...
default:
   ...
breaksw
endsw
   Each case label is successively matched, against the specified string which is
   first command and filename expanded. The file metacharacters '*', '?' and '[]'
   may be used in the case labels, which are variable expanded. If none of the
   labels match before a 'default' label is found, then the execution begins after the
   default label. Each case label and the default label must appear at the begin-
   ning of a line. The command breaksw causes execution to continue after the
   endsw. Otherwise control may fall through case labels and default labels as in C.
   If no label matches and there is no default, execution continues after the endsw.

time
time command
   With no argument, a summary of time used by this shell and its children is
   printed. If arguments are given the specified simple command is timed and a
   time summary as described under the time variable is printed. If necessary, an
   extra shell is created to print the time statistic when the command completes.

umask
umask value
   The file creation mask is displayed (first form) or set to the specified value
   (second form). The mask is given in octal. Common values for the mask are
   002 giving all access to the group and read and execute access to others or 022
   giving all access except no write access for users in the group or others.

unalias pattern
   All aliases whose names match the specified pattern are discarded. Thus all
   aliases are removed by 'unalias *'. It is not an error for nothing to be unaliased.

unhash
   Use of the internal hash table to speed location of executed programs is dis-
   abled.

unlimit
unlimit resource
unlimit -h
unlimit -h resource
   Removes the limitation on resource. If no resource is specified, then all resource
   limitations are removed. If -h is given, the corresponding hard limits are
   removed. Only the super-user may do this.

unset pattern
   All variables whose names match the specified pattern are removed. Thus all
   variables are removed by 'unset *'; this has noticeably distasteful side-effects.
It is not an error for nothing to be unset.

**unsetenv pattern**

Removes all variables whose name match the specified pattern from the environment. See also the `setenv` command above and `printenv(1)`.

**wait**

All background jobs are waited for. If the shell is interactive, then an interrupt can disrupt the wait, at which time the shell prints names and job numbers of all jobs known to be outstanding.

**while (expr)**

```
... 
```

**end**

While the specified expression evaluates non-zero, the commands between the `while` and the matching end are evaluated. `break` and `continue` may be used to terminate or continue the loop prematurely. (The `while` and `end` must appear alone on their input lines.) Prompting occurs here the first time through the loop as for the `foreach` statement if the input is a terminal.

**%job**

Brings the specified job into the foreground.

**%job &**

Continues the specified job in the background.

**@**

```
@ name = expr
@ name[index] = expr
```

The first form prints the values of all the shell variables. 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 `'"' '```. 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.

The operators ` '*='`, ` '+='`, etc are available as in C. The space separating the name from the assignment operator is optional. Spaces are, however, mandatory in separating components of `expr` which would otherwise be single words.

Special postfixed ` '++'` and ` '--'` operators increment and decrement `name` respectively, i.e. ` '@ i++'`.

**Pre-defined and environment 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 will not then be modified unless this is done explicitly by the user.

This shell copies the environment variable `USER` into the variable `user`, `TERM` into `term`, and `HOME` into `home`, and copies these back into the environment whenever the normal shell variables are reset. The environment variable `PATH` is likewise handled; it is not necessary to worry about its setting other than in the file `.cshrc` as inferior `sh` processes will import the definition of `path` from the environment, and re-export it if you then change it.

**argv**

Set to the arguments to the shell, it is from this variable that positional parameters are substituted, i.e. `$1` is replaced by `$argv[1]`, etc.
cdpath
Gives a list of alternate directories searched to find subdirectories in chdir commands.

cwd
The full pathname of the current directory.

echo
Set when the \-x command line option is given. Causes each command and its arguments to be echoed just before it is executed. For non-built-in commands all expansions occur before echoing. Built-in commands are echoed before command and filename substitution, since these substitutions are then done selectively.

histchars
Can be given a string value to change the characters used in history substitution. The first character of its value is used as the history substitution character, replacing the default character !. The second character of its value replaces the character ↑ in quick substitutions.

history
Can be given a numeric value to control the size of the history list. Any command which has been referenced in this many events will not be discarded. Too large values of history may run the shell out of memory. The last executed command is always saved on the history list.

home
The home directory of the invoker, initialized from the environment. The filename expansion of ‘~’ refers to this variable.

ignoreeof
If set, the shell ignores end-of-file from input devices which are terminals. This prevents shells from accidentally being killed by control-D’s.

mail
The files where the shell checks for mail. This is done after each command completion which will result in a prompt, if a specified interval has elapsed. The shell says ‘You have new mail.’ if the file exists with an access time not greater than its modify time.

If the first word of the value of mail is numeric it specifies a different mail checking interval, in seconds, than the default, which is 10 minutes.

If multiple mail files are specified, then the shell says ‘New mail in name’ when there is mail in the file name.

noclobber
As described in the section on Input/output, restrictions are placed on output redirection to insure that files are not accidentally destroyed, and that ‘>>’ redirections refer to existing files.

noglob
If set, filename expansion is inhibited. This is most useful in shell scripts which are not dealing with filenames, or after a list of filenames has been obtained and further expansions are not desirable.

nonomatch
If set, it is not an error for a filename expansion to not match any existing files; rather the primitive pattern is returned. It is still an error for the primitive pattern to be malformed, i.e. ‘echo [’ still gives an error.

notify
If set, the shell notifies asynchronously of job completions. The default is to rather present job completions just before printing a 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 then only full path
names will execute. The usual search path is ", "/bin" and "/usr/bin", but this may vary from system to system. For the super-user the default search path is "/etc", "/bin" and "/usr/bin". A shell which is given neither the -c nor the -t option will normally hash 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 or the commands may not be found.

prompt
The string which is printed before each command is read from an interactive terminal input. If a "?" appears in the string it will be replaced by the current event number unless a preceding "\\" is given. Default is "%", or "#" for the super-user.

savehist
is given a numeric value to control the number of entries of the history list that are saved in ".history" when the user logs out. Any command which has been referenced in this many events will be saved. During start up the shell sources ".history" into the history list enabling history to be saved across logins. Too large values of savehist will slow down the shell during start up.

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 the description of Non-built in Command Execution below.) Initialized to the (system-dependent) home of the shell.

status
The status returned by the last command. If it terminated abnormally, then 0200 is added to the status. Built-in commands which fail return exit status '1', all other built-in commands set status '0'.

time
Controls automatic timing of commands. If set, then any command which takes more than this many cpu seconds will cause a line giving user, system, and real times and a utilization percentage which is the ratio of user plus system times to real time to be printed when it terminates.

verbose
Set by the -v command line option, causes the words of each command to be printed after history substitution.

Non-built in Command Execution

When a command to be executed is found to not be a built-in command the shell attempts to execute the command via execve(2). Each word in the variable path names a directory from which the shell will attempt to execute the command. If it is given neither a -c nor a -t option, the shell will hash the names in these directories into an internal table so that it will only try an exec in a directory if there is a possibility that the command resides there. This greatly speeds command location when a large number of directories are present in the search path. If this mechanism has been turned off (via unhash), or if the shell was given a -c or -t argument, and in any case for each directory component of path which 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 execute.

Parenthesized commands are always executed in a subshell. Thus ‘(cd ; pwd); pwd’ prints the home directory; leaving you where you were (printing this after the home directory), while ‘cd ; pwd’ leaves you in the home directory. Parenthesized commands are most often used to prevent 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 is spawned to read it.

If there is an alias for shell then the words of the alias will be prepended to the argument list to form the shell command. The first word of the alias should be the full path name of the shell (e.g. `$shell`). Note that this is a special, late occurring, case of alias substitution, and only allows words to be prepended to the argument list without modification.

**Argument list processing**

If argument 0 to the shell is `~` then this is a login shell. The flag arguments are interpreted as follows:

- **-b** This flag forces a ‘break’ from option processing, causing any further shell arguments to be treated as non-option arguments. The remaining arguments will not be interpreted as shell options. This may be used to pass options to a shell script without confusion or possible subterfuge. The shell will not run a set-user ID script without this option.

- **-c** Commands are read from the (single) following argument which must be present. Any remaining arguments are placed in `argv`.

- **-e** The shell exits if any invoked command terminates abnormally or yields a non-zero exit status.

- **-f** The shell will start faster, because it will neither search for nor execute commands from the file `.cshrc` in the invoker’s home directory.

- **-i** The shell is interactive and prompts for its top-level input, even if it appears to not be a terminal. Shells are interactive without this option if their inputs and outputs are terminals.

- **-n** Commands are parsed, but not executed. This aids in syntactic checking of shell scripts.

- **-s** Command input is taken from the standard input.

- **-t** A single line of input is read and executed. A `\` character may be used to escape the newline at the end of this line and continue onto another line.

- **-v** Causes the `verbose` variable to be set, with the effect that command input is echoed after history substitution.

- **-x** Causes the `echo` variable to be set, so that commands are echoed immediately before execution.

- **-V** Causes the `verbose` variable to be set even before `.cshrc` is executed.

- **-X** Is to `-x` as `-V` is to `-v`.

After processing of flag arguments, if arguments remain but none of the `-c, -i, -s, or -t` options was given, the first argument is taken as the name of a file of commands to be executed. The shell opens this file, and saves its name for possible resubstitution by `$0`. Since many systems use either the standard version 6 or version 7 shells whose shell scripts are not compatible with this shell, the shell will execute such a ‘standard’ shell if the first character of a script is not a `#`, i.e. if the script does not start with a comment. Remaining arguments initialize the variable `argv`.

Signal handling

The shell normally ignores quit signals. Jobs running detached (either by ‘&’ or the
bg or %...& commands) are immune to signals generated from the keyboard, includ-
ing hangups. Other signals have the values which the shell inherited from its parent.
The shell's handling of interrupts and terminate signals in shell scripts can be con-
trolled by omint. 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 interrupts
allowed when a login shell is reading the file '.logout'.

AUTHOR

William Joy. Job control and directory stack features first implemented by J.E. Kulp
of I.I.A.S.A., Luxenbug, Austria, with different syntax than that used now.

FILES

~/.cshrc
~/login
~/logout
/bin/sh
/tmp/sh*
/etc/passwd

Read at beginning of execution by each shell.
Read by login shell, after '.cshrc' at login.
Read by login shell, at logout.
Standard shell, for shell scripts not starting with a '#'.
Temporary file for '<<'.
Source of home directories for '~-name'.

LIMITATIONS

Words can be no longer than 1024 characters. The system limits argument lists to
10240 characters. The number of arguments to a command which involves filename
expansion is limited to 1/6th the number of characters allowed in an argument list.
Command substitutions may substitute no more characters than are allowed in an
argument list. To detect looping, the shell restricts the number of alias substitutions
on a single line to 20.

SEE ALSO

sh(1), stty(1B), access(2), execve(2), fork(2), killpg(2), pipe(2), sigvec(2), umask(2),
setrlimit(2), wait(2), a.out(4)

BUGS

When a command is restarted from a stop, the shell prints the directory it started in if
this is different from the current directory; this can be misleading (i.e. wrong) as the
job may have changed directories internally.

Shell builtin functions are not stoppable/restartable. Command sequences of the
form ‘a ; b ; c’ are also not handled gracefully when stopping is attempted. If you
suspend ‘b’, the shell will then immediately execute ‘c’. This is especially noticeable
if this expansion results from an alias. It suffices to place the sequence of commands
in (’s to force it to a subshell, i.e. ‘( a ; b ; c )’.

Control over tty output after processes are started is primitive; perhaps this will
inspire someone to work on a good virtual terminal interface. In a virtual terminal
interface much more interesting things could be done with output control.

Alias substitution is most often used to clumsily simulate shell procedures; shell pro-
cedures should be provided rather than aliases.

Commands within loops, prompted for by ‘?’, are not placed in the history list. Control
structure should be parsed rather than being recognized as built-in commands.
This would allow control commands to be placed anywhere, to be combined with ‘!’,
and to be used with ‘&’ and ‘!’ metasyntax.

It should be possible to use the ‘!’ modifiers on the output of command substitutions.
All and more than one ‘!’ modifier should be allowed on ‘$’ substitutions.
NAME
csplit – context split

SYNOPSIS
csplit [-s] [-k] [-f prefix] file arg1 [... argn]

DESCRIPTION
csplit reads file and separates it into n+1 sections, defined by the arguments arg1...argn. By default the sections are placed in xx00 ... xnn (n may not be greater than 99). These sections get the following pieces of file:
00: From the start of file up to (but not including) the line referenced by arg1.
01: From the line referenced by arg1 up to the line referenced by arg2.
  ...
n+1: From the line referenced by argn to the end of file.

If the file argument is a – then standard input is used.

The options to csplit are:
-s csplit normally prints the character counts for each file created. If the
-s option is present, csplit suppresses the printing of all character
counts.
-k csplit normally removes created files if an error occurs. If the -k
option is present, csplit leaves previously created files intact.
-f prefix If the -f option is used, the created files are named prefix00 ...
prefixn. The default is xx00 ... xnn.

The arguments (arg1 ... argn) to csplit can be a combination of the following:
/rexp/ A file is to be created for the section from the current line up to (but
not including) the line containing the regular expression rexp. The
current line becomes the line containing rexp. This argument may be
followed by an optional + or – some number of lines (e.g., /Page/-5).
%rexp% This argument is the same as /rexp/, except that no file is created for
the section.
lnno A file is to be created from the current line up to (but not including)
lnno. The current line becomes lnno.
{num} Repeat argument. This argument may follow any of the above argu-
ments. If it follows a rexp type argument, that argument is applied
num more times. If it follows lnno, the file will be split every lnno
lines (num times) from that point.

Enclose all rexp type arguments that contain blanks or other characters meaningful to
the shell in the appropriate quotes. Regular expressions may not contain embedded
new-lines. csplit does not affect the original file; it is the users responsibility to
remove it.

EXAMPLES
csplit -f cobol file '/procedure division/' /par5:/ /par16:/

This example creates four files, cobol00 ... cobol03. After editing the “split” files,
they can be recombined as follows:
cat cobol[0-3] > file

Note that this example overwrites the original file.
csplit -k file 100 [99]

This example would split the file at every 100 lines, up to 10,000 lines. The -k option causes the created files to be retained if there are less than 10,000 lines; however, an error message would still be printed.

csplit -k prog.c '%%main(%% '/^)/+1' [20]

Assuming that prog.c follows the normal C coding convention of ending routines with a ) at the beginning of the line, this example will create a file containing each separate C routine (up to 21) in prog.c.

SEE ALSO
ed(1), sh(1), regexp(5).

DIAGNOSTICS
Self-explanatory except for:
arg - out of range
which means that the given argument did not reference a line between the current position and the end of the file.
NAME

c - spawn getty to a remote terminal

SYNOPSIS

c [ -wn ] [ -xn ] [ -h ] [ -v ] [ -sspeed ] telno ...

DESCRIPTION

c dials the telephone number of a modem that is attached to a terminal, and spawns a getty process to that terminal. Telno is a telephone number, with equal signs for secondary dial tones and minus signs for delays at appropriate places. (The set of legal characters for telno is 0 thru 9, -, =, *, and #. The maximum length telno is 31 characters). If more than one telephone number is specified, c will try each in succession until one answers; this is useful for specifying alternate dialing paths.

c will try each line listed in the file /usr/lib/uucp/Devices until it finds an available line with appropriate attributes or runs out of entries. If there are no free lines, c will ask if it should wait for one, and if so, for how many minutes it should wait before it gives up. c will continue to try to open the dialers at one-minute intervals until the specified limit is exceeded. The dialogue may be overridden by specifying the -wn option, where n is the maximum number of minutes that c is to wait for a line.

The -xn option is used for debugging; it produces a detailed output of the program execution on stderr. The debugging level, n, is a single digit; -x9 is the most useful value.

Normally, c will hang up the current line, so the line can answer the incoming call. The -h option will prevent this action. The -h option will also wait for the termination of the specified c process before returning control to the user's terminal. If the -v option is used, c will send a running narrative to the standard error output stream.

The data rate may be set with the -s option, where speed is expressed in baud. The default rate is 1200.

After the user on the destination terminal logs out, there are two things that could occur depending on what type of getty is on the line (getty or uugetty). For the first case, c prompts, Reconnect? If the response begins with the letter n, the line will be dropped; otherwise, getty will be started again and the login: prompt will be printed. In the second case, there is already a getty (uugetty) on the line, so the login: message will appear.

To log out properly, the user must type control D.

Of course, the destination terminal must be attached to a modem that can answer the telephone.

FILES

/usr/lib/uucp/Devices
/usr/adm/ctlog

SEE ALSO

cu(1C), getty(1M), login(1), uucp(1C). uugetty(1M).

BUGS

For a shared port, one used for both dial-in and dial-out, the uugetty program running on the line must have the -r option specified (see uugetty(1M)).
NAME
ctags – create a tags file

SYNOPSIS
ctags [ -BFatuwx ] [ -f tagsfile ] name ...

DESCRIPTION
Ctags makes a tags file for ex(1) from the specified C, Pascal, Fortran, YACC, lex, and lisp sources. A tags file gives the locations of specified objects (in this case functions and typedefs) 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, typedefs with a line number. Specifiers are given in separate fields on the line, separated by blanks or tabs. Using the tags file, ex can quickly find these objects definitions.

If the -x flag is given, ctags produces a list of object names, the line number and file name on which each is defined, as well as the text of that line and prints this on the standard output. This is a simple index which can be printed out as an off-line readable function index.

If the -v flag is given, an index of the form expected by vgrind(1) is produced on the standard output. This listing contains the function name, file name, and page number (assuming 64 line pages). Since the output will be sorted into lexicographic order, it may be desired to run the output through sort -f. Sample use:
  ctags -v files 1 sort -f > index
  vgrind -x index

Normally ctags places the tag descriptions in a file called tags; this may be overridden with the -f option.

Files whose names end in .c or .h are assumed to be C source files and are searched for C routine and macro definitions. Files whose names end in .y are assumed to be YACC source files. Files whose names end in .j are assumed to be either lisp files if their first non-blank character is '(', '(', or '|', or lex files otherwise. Other files are first examined to see if they contain any Pascal or Fortran routine definitions; if not, they are processed again looking for C definitions.

Other options are:
-F use forward searching patterns (/.../) (default).
-B use backward searching patterns (?...?).
-a append to tags file.
-t create tags for typedefs.
-w suppressing warning diagnostics.
-u causing the specified files to be updated in tags, that is, all references to them are deleted, and the new values are appended to the file. (Beware: this option is implemented in a way which is rather slow; it is usually faster to simply rebuild the tags file.)

The tag main is treated specially in C programs. The tag formed is created by prepending M to the name of the file, with a trailing .c removed, if any, and leading pathname components also removed. This makes use of ctags practical in directories with more than one program.

FILES
tags output tags file
SEE ALSO

ex(1), vi(1)

AUTHOR

Ken Arnold; FORTRAN added by Jim Kleckner; Bill Joy added Pascal and -x, replacing cxref; C typedefs added by Ed Pelegri-Llopart.

BUGS

Recognition of functions, subroutines and procedures for FORTRAN and Pascal is done is a very simpleminded way. No attempt is made to deal with block structure; if you have two Pascal procedures in different blocks with the same name you lose.

The method of deciding whether to look for C or Pascal and FORTRAN functions is a hack.

Does not know about #ifdefs.

Should know about Pascal types. Relies on the input being well formed to detect typedefs. Use of -tx shows only the last line of typedefs.
NAME

cu – call another UNIX system

SYNOPSIS

cu [-s speed] [-h] [-d] [-o | -e] -l line

cu [-h] [-d] [-o | -e] systemname

DESCRIPTION

cu calls up another UNIX system, a terminal, or possibly a non-UNIX system. It manages an interactive conversation with possible transfers of ASCII files.

cu accepts the following options and arguments:

-sspeed
Specifies the transmission speed (300, 1200, 2400, 4800, 9600); The default value is "Any" speed which will depend on the order of the lines in the /usr/lib/uucp/Devices file. Most modems are either 300 or 1200 baud. Directly connected lines may be set to a speed higher than 1200 baud.

-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 having the right speed. When the -I option is used without the -s option, the speed of a line is taken from the Devices file. When the -I and -s options are both used together, cu will search the Devices file to check if the requested speed for the requested line is available. If so, the connection will be made at the requested speed; otherwise an error message will be printed and the call will not be made. The specified device is generally a directly connected asynchronous line (e.g., /dev/ttyab) in which case a telephone number (telno) is not required. The specified device need not be in the /dev directory. If the specified device is associated with an auto dialer, a telephone number must be provided. Use of this option with systemname rather than telno will not give the desired result (see systemname below).

-h
Emulates local echo, supporting calls to other computer systems which expect terminals to be set to half-duplex mode.

-t
Used to dial an ASCII terminal which has been set to auto answer. Appropriate mapping of carriage-return to carriage-return-line-feed pairs is set.

-d
Causes diagnostic traces to be printed.

-o
Designates that odd parity is to be generated for data sent to the remote system.

-n
For added security, will prompt the user to provide the telephone number to be dialed rather than taking it from the command line.

-e
Designates that even parity is to be generated for data sent to the remote system.

telno
When using an automatic dialer, the argument is the telephone number with equal signs for secondary dial tone or minus signs placed appropriately for delays of 4 seconds.

systemname
A uucp system name may be used rather than a telephone number; in this case, cu will obtain an appropriate direct line or telephone number from /usr/lib/uucp/System. Note: the systemname option should not be used in conjunction with the -l and -s options as cu will connect to the first available line for the system name specified, ignoring the requested
line and speed.

After making the connection, `cu` runs as two processes: the `transmit` process reads data from the standard input and, except for lines beginning with `~`, passes it to the remote system; the `receive` process accepts data from the remote system and, except for lines beginning with `~`, passes it to the standard output. Normally, an automatic DC3/DC1 protocol is used to control input from the remote so the buffer is not overrun. Lines beginning with `~` have special meanings.

The `transmit` process interprets the following user initiated commands:

```
~.                        terminate the conversation.
~!                        escape to an interactive shell on the local system.
~!cmd...                  run cmd on the local system (via sh -c).
~!$cmd...                 run cmd locally and send its output to the remote system.
~%cd                      change the directory on the local system. Note: ~!cd will
                          cause the command to be run by a sub-shell, probably not
                          what was intended.
~%take from [ to ]        copy file from (on the remote system) to file to on the local sys-
                          tem. If to is omitted, the from argument is used in both places.
~%put from [ to ]         copy file from (on local system) to file to on remote system. If
                          to is omitted, the from argument is used in both places.
```

For both `~%take` and `put` commands, as each block of the file is transferred, consecutive single digits are printed to the terminal.

```
~-- line                  send the line ~ line to the remote system.
~%break                   transmit a BREAK to the remote system (which can also be
                          specified as ~%b).
~%debug                   toggles the -d debugging option on or off (which can also be
                          specified as ~%d).
~t                         prints the values of the termio structure variables for the
                          user’s terminal (useful for debugging).
~l                         prints the values of the termio structure variables for the
                          remote communication line (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 which
does not respond properly to the DC3 and DC1 characters.
```

The `receive` process normally copies data from the remote system to its standard output. Internally the program accomplishes this by initiating an output diversion to a file when a line from the remote begins with `~`.

Data from the remote is diverted (or appended, if `>>` is used) to `file` on the local system. The trailing `~>` marks the end of the diversion.

The use of `~%put` requires `stty(1)` and `cat(1)` on the remote side. 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 the existence of echo(1) and cat(1) on the remote system. Also, tabs mode (See stty(1)) should be set on the remote system if tabs are to be copied without expansion to spaces.

When cu is used on system X to connect to system Y and subsequently used on system Y to connect to system Z, commands on system Y can be executed by using ~~. Executing a tilde command reminds the user of the local system uname. For example, uname can be executed on Z, X, and Y as follows:

```
uname
Z
~[X]!uname
X
~~[Y]!uname
Y
```

In general, ~ causes the command to be executed on the original machine, ~~~ causes the command to be executed on the next machine in the chain.

**EXAMPLES**

To dial a system whose telephone number is 9 201 555 1212 using 1200 baud (where dialtone is expected after the 9):
```
cu -s1200 9=12015551212
```
If the speed is not specified, "Any" is the default value.
To login to a system connected by a direct line:
```
cu -l /dev/ttyXX
```
or
```
cu -l ttyXX
```
To dial a system with the specific line and a specific speed:
```
cu -s1200 -l ttyXX
```
To dial a system using a specific line associated with an auto dialer:
```
cu -l cuIXX 9=12015551212
```
To use a system name:
```
cu systemname
```

**FILES**

```
/usr/lib/uucp/Systems
/usr/lib/uucp/Devices
/usr/spool/locks/LCK..(tty-device)
```

**SEE ALSO**

cat(1), ct(1C), echo(1), stty(1), uucp(1C), uname(1).

**DIAGNOSTICS**

Exit code is zero for normal exit, otherwise, one.

**WARNINGS**

The cu command does not do any integrity checking on data it transfers. Data fields with special cu characters may not be transmitted properly. Depending on the inter-connection hardware, it may be necessary to use a ~ to terminate the conversion even if stty 0 has been used. Non-printing characters are not dependably transmitted using either the ~%put or ~%take commands. cu between an IMBR1 and a penril modem will not return a login prompt immediately upon connection. A carriage
return will return the prompt.

**BUGS**

There is an artificial slowing of transmission by cu during the ~%put operation so that loss of data is unlikely.
Directory and File Management Utilities

NAME

cut – cut out selected fields of each line of a file

SYNOPSIS

cut –clist [ file ]
cut –flist [ –d char ] [ –s ] [ file ]

DESCRIPTION

Use cut to cut out columns from a table or fields from each line of a file; in data base parlance, it implements the projection of a relation. The fields as specified by list can be fixed length, i.e., character positions as on a punched card (–c option) or the length can vary from line to line and be marked with a field delimiter character like \texttt{tab} (–f option). cut can be used as a filter; if no files are given, the standard input is used. In addition, a file name of "\texttt{"}" explicitly refers to standard input.

The meanings of the options are:

\texttt{list} \quad A comma-separated list of integer field numbers (in increasing order), with optional – to indicate ranges [e.g., \texttt{1,4,7; 1–3,8; –5,10} (short for \texttt{1–5,10}; or \texttt{3–} (short for third through last field)].

\texttt{–clist} \quad The list following –c (no space) specifies character positions (e.g., \texttt{–c1–72} would pass the first 72 characters of each line).

\texttt{–flist} \quad The list following –f is a list of fields assumed to be separated in the file by a delimiter character (see –d); e.g., \texttt{–f1,7 copies the first and seventh field only. Lines with no field delimiters will be passed through intact (useful for table subheadings), unless –s is specified.}

\texttt{–d char} \quad The character following –d is the field delimiter (–f option only). Default is \texttt{tab}. Space or other characters with special meaning to the shell must be quoted.

\texttt{–s} \quad Suppresses lines with no delimiter characters in case of –f option. Unless specified, lines with no delimiters will be passed through untouched.

Either the –c or –f option must be specified.

Use \texttt{grep(1)} to make horizontal “cuts” (by context) through a file, or \texttt{paste(1)} to put files together column-wise (i.e., horizontally). To reorder columns in a table, use cut and paste.

EXAMPLES

cut –d: –f1,5 /etc/passwd \quad \text{mapping of user IDs to names}
name='who am i | cut –f1 –d" "' \quad \text{to set name to current login name.}

DIAGNOSTICS

ERROR: line too long

A line can have no more than 1023 characters or fields, or there is no new-line character.

ERROR: bad list for c / f option
Missed –c or –f option or incorrectly specified list. No error occurs if a line has fewer fields than the list calls for.

ERROR: no fields \quad The list is empty.

ERROR: no delimiter
Missing char on –d option.

ERROR: cannot handle multiple adjacent backspaces
Adjacent backspaces cannot be processed correctly.
WARNING: cannot open <filename>
Either filename cannot be read or does not exist. If multiple filenames are present, processing continues.

SEE ALSO

grep(1), paste(1).
NAME
date – print and set the date

SYNOPSIS
date [ mmddhhmm[yy] ] | +format |

DESCRIPTION
If no argument is given, or if the argument begins with +, the current date and time are printed. Otherwise, the current date is set. The first mm is the month number; dd is the day number in the month; hh is the hour number (24 hour system); the second mm is the minute number; yy is the last 2 digits of the year number and is optional. For example:

date 10080045

sets the date to Oct 8, 12:45 AM. The current year is the default if no year is mentioned. The system operates in GMT. date takes care of the conversion to and from local standard and daylight time. Only the superuser may change the date.

If the argument begins with +, the output of date is under the control of the user. All output fields are of fixed size (zero padded if necessary). Each field descriptor is preceded by % and will be replaced in the output by its corresponding value. A single % is encoded by %%%. All other characters are copied to the output without change. The string is always terminated with a new-line character.

Field Descriptors:

n insert a new-line character
t insert a tab character
m month of year – 01 to 12
d day of month – 01 to 31
y last 2 digits of year – 00 to 99
D date as mm/dd/yy
H hour – 00 to 23
M minute – 00 to 59
S second – 00 to 59
T time as HH:MM:SS
j day of year – 001 to 366
w day of week – Sunday = 0
a abbreviated weekday – Sun to Sat
h abbreviated month – Jan to Dec
r time in AM/PM notation

EXAMPLE
date ‘+DATE: %m/%d/%y%TIME: %H:%M:%S’
would have generated as output:
DATE: 08/01/76
TIME: 14:45:05

DIAGNOSTICS
No permission if you are not the super-user and you try to change the date;
bad conversion if the date set is syntactically incorrect;
bad format character if the field descriptor is not recognizable.
NAME
dbg – debugger

SYNOPSIS
dbg [a.outfile] [corefile]

DESCRIPTION
The $dbg$ command calls the debugger. It can be used with C and Fortran programs to examine their object files and core files and to provide a controlled environment for their execution.

$a.outfile$ defaults to $a.out$ which is produced by compilation or link editing. This may be changed to reflect your own file name. $corefile$ defaults to $core$ and may be changed to another corefile name.

FILES
a.out
core

SEE ALSO
cc(1), a.out(4), core(4), sh(1), syms(4)
The Programmer's Guide.
NAME

dc – desk calculator

SYNOPSIS

dc [ file ]

DESCRIPTION

`dc` is an arbitrary precision arithmetic package. Ordinarily it operates on decimal integers, but one may specify an input base, output base, and a number of fractional digits to be maintained. (See `bc(1)`, a preprocessor for `dc` that provides infix notation and a C-like syntax that implements functions. `Bc` also provides reasonable control structures for programs.) The overall structure of `dc` is a stacking (reverse Polish) calculator. If an argument is given, input is taken from that file until its end, then from the standard input. The following constructions are recognized:

`number`

The value of the number is pushed on the stack. A number is an unbroken string of the digits 0-9. It may be preceded by an underscore (_) to input a negative number. Numbers may contain decimal points.

`+ - / * % ^`

The top two values on the stack are added (+), subtracted (-), multiplied (*), divided (/), remaindered (%), or exponentiated (^). The two entries are popped off the stack; the result is pushed on the stack in their place. Any fractional part of an exponent is ignored.

`s` The top of the stack is popped and stored into a register named `x`, where `x` may be any character. If the `s` is capitalized, `x` is treated as a stack and the value is pushed on it.

`l` The value in register `x` is pushed on the stack. The register `x` is not altered. All registers start with zero value. If the `l` is capitalized, register `x` is treated as a stack and its top value is popped onto the main stack.

`d` The top value on the stack is duplicated.

`p` The top value on the stack is printed. The top value remains unchanged.

`P` Interprets the top of the stack as an ASCII string, removes it, and prints it.

`f` All values on the stack are printed.

`q` Exits the program. If executing a string, the recursion level is popped by two.

`Q` Exits the program. The top value on the stack is popped and the string execution level is popped 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.

` [...]` Puts the bracketed ASCII string onto the top of the stack.

`<x >x =x`

The top two elements of the stack are popped and compared. Register `x` is evaluated if they obey 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 UNIX system command.
c  All values on the stack are popped.
i  The top value on the stack is popped and used as the number radix for further input. i pushes the input base on the top of the stack.
o  The top value on the stack is popped and used as the number radix for further output.
o  Pushes the output base on the top of the stack.
k  The top of the stack is popped, and that value is used as a non-negative scale factor: the appropriate number of places are printed on output, and maintained during multiplication, division, and exponentiation. The interaction of scale factor, input base, and output base will be reasonable if all are changed together.
z  The stack level is pushed onto the stack.
Z  Replaces the number on the top of the stack with its length.
?  A line of input is taken from the input source (usually the terminal) and executed.
;:  are used by bc(1) for array operations.

EXAMPLE

This example prints the first ten values of n!:

[1a1+d$a+p$a10>y]sy
0sa1
lyx

SEE ALSO

bc(1).

DIAGNOSTICS

x is unimplemented
where x is an octal number.

stack empty
for not enough elements on the stack to do what was asked.

Out of space
when the free list is exhausted (too many digits).

Out of headers
for too many numbers being kept around.

Out of pushdown
for too many items on the stack.

Nesting Depth
for too many levels of nested execution.
NAME
dd – convert and copy a file

SYNOPSIS
dd [option=value] ...

DESCRIPTION

`dd` copies the specified input file to the specified output with possible conversions. The standard input and output are used by default. The input and output block size may be specified to take advantage of raw physical I/O.

<table>
<thead>
<tr>
<th>option</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>if=file</td>
<td>input file name; standard input is default</td>
</tr>
<tr>
<td>of=file</td>
<td>output file name; standard output is default</td>
</tr>
<tr>
<td>ibs=n</td>
<td>input block size n bytes (default 512)</td>
</tr>
<tr>
<td>obs=n</td>
<td>output block size (default 512)</td>
</tr>
<tr>
<td>bs=n</td>
<td>set both input and output block size, superseding <code>ibs</code> and <code>obs</code>; also, if no conversion is specified, it is particularly efficient since no in-core copy need be done</td>
</tr>
<tr>
<td>cbs=n</td>
<td>conversion buffer size</td>
</tr>
<tr>
<td>skip=n</td>
<td>skip n input blocks before starting copy</td>
</tr>
<tr>
<td>seek=n</td>
<td>seek n blocks from beginning of output file before copying</td>
</tr>
<tr>
<td>count=n</td>
<td>copy only n input blocks</td>
</tr>
<tr>
<td>conv=ascii</td>
<td>convert EBCDIC to ASCII</td>
</tr>
<tr>
<td></td>
<td>ebcdis</td>
</tr>
<tr>
<td>block</td>
<td>slightly different map of ASCII to EBCDIC</td>
</tr>
<tr>
<td>unblock</td>
<td>convert fixed length records to variable length</td>
</tr>
<tr>
<td>lcase</td>
<td>map alphabetics to lower case</td>
</tr>
<tr>
<td>ucase</td>
<td>map alphabetics to upper case</td>
</tr>
<tr>
<td>swab</td>
<td>swap every pair of bytes</td>
</tr>
<tr>
<td>noerror</td>
<td>do not stop processing on an error</td>
</tr>
<tr>
<td>sync</td>
<td>pad every input block to <code>ibs</code></td>
</tr>
<tr>
<td>... , ...</td>
<td>several comma-separated conversions</td>
</tr>
</tbody>
</table>

Where sizes are specified, a number of bytes is expected. A number may end with k, b, or w to specify multiplication by 1024, 512, or 2, respectively; a pair of numbers may be separated by x to indicate multiplication.

`cbs` is used only if `conv=ascii`, `conv=unblock`, `conv=ebcdic`, or `conv=block` is specified. In the first two cases, `cbs` characters are placed into the conversion buffer (converted to ASCII). Trailing blanks are trimmed and a new-line added before sending the line to the output. In the latter two cases, ASCII characters are read into the conversion buffer (converted to EBCDIC). Blanks are added to make up an output block of size `cbs`.

After completion, `dd` reports the number of whole and partial input and output blocks.

DIAGNOSTICS

`f+p blocks in(out)` numbers of full and partial blocks read(written)
delta – make a delta (change) to an SCCS file

SYNOPSIS

delta [-rSID] [-s] [-n] [-glist] [-m[mrlist]] [-y[comment]] [-p] files

DESCRIPTION

delta is used to permanently introduce into the named SCCS file changes that were made to the file retrieved by get(1) (called the g-file, or generated file).

delta makes a delta to each named SCCS file. If a directory is named, delta behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.) and unreadable files are silently ignored. If a name of – is given, the standard input is read (see WARNINGS); each line of the standard input is taken to be the name of an SCCS file to be processed.

delta may issue prompts on the standard output depending upon certain keyletters specified and flags [see admin(1)] that may be present in the SCCS file (see –m and –y keyletters below).

Keyletter arguments apply independently to each named file.

-rSID  Uniquely identifies which delta is to be made to the SCCS file. The use of this keyletter is necessary only if two or more outstanding gets for editing (get –e) on the same SCCS file were done by the same person (login name). The SID value specified with the –r keyletter can be either the SID specified on the get command line or the SID to be made as reported by the get command [see get(1)]. A diagnostic results if the specified SID is ambiguous, or, if necessary and omitted on the command line.

-s     Suppresses the issue, on the standard output, of the created delta’s SID, as well as the number of lines inserted, deleted and unchanged in the SCCS file.

-n     Specifies retention of the edited g-file (normally removed at completion of delta processing).

-glist a list (see get(1) for the definition of list) of deltas which are to be ignored when the file is accessed at the change level (SID) created by this delta.

-m[mrlist] If the SCCS file has the v flag set [see admin(1)] then a Modification Request (MR) number must be supplied as the reason for creating the new delta.

If –m is not used and the standard input is a terminal, the prompt MRs? is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. The MRs? prompt always precedes the comments? prompt (see –y keyletter).

MRs in a list are separated by blanks and/or tab characters. An unescaped new-line character terminates the MR list.

Note that if the v flag has a value [see admin(1)], it is taken to be the name of a program (or shell procedure) which will validate the correctness of the MR numbers. If a non-zero exit status is returned from the MR number validation program, delta terminates. (It is assumed that the MR numbers were not
all valid.)

-\texttt{y}[\texttt{comment}] Arbitrary text used to describe the reason for making the delta. A null string is considered a valid comment.

If -\texttt{y} is not specified and the standard input is a terminal, the prompt comments? is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. An unescaped new-line character terminates the comment text.

-\texttt{p} Causes \texttt{delta} to print (on the standard output) the SCCS file differences before and after the delta is applied in a \texttt{diff(1)} format.

\textbf{FILES}

\begin{itemize}
    \item \texttt{g-file} Existed before the execution of \texttt{delta}; removed after completion of \texttt{delta}.
    \item \texttt{p-file} Existed before the execution of \texttt{delta}; may exist after completion of \texttt{delta}.
    \item \texttt{q-file} Created during the execution of \texttt{delta}; removed after completion of \texttt{delta}.
    \item \texttt{x-file} Created during the execution of \texttt{delta}; renamed to SCCS file after completion of \texttt{delta}.
    \item \texttt{z-file} Created during the execution of \texttt{delta}; removed during the execution of \texttt{delta}.
    \item \texttt{d-file} Created during the execution of \texttt{delta}; removed after completion of \texttt{delta}.
    \item \texttt{/usr/bin/bdiff} Program to compute differences between the “gotten” file and the \texttt{g-file}.
\end{itemize}

\textbf{WARNINGS}

Lines beginning with an SOH ASCII character (binary 001) cannot be placed in the SCCS file unless the SOH is escaped. This character has special meaning to SCCS [see \texttt{sccsfile(4)} (5)] and will cause an error.

A get of many SCCS files, followed by a \texttt{delta} of those files, should be avoided when the get generates a large amount of data. Instead, multiple get/\texttt{delta} sequences should be used.

If the standard input (-) is specified on the \texttt{delta} command line, the \texttt{-m} (if necessary) and \texttt{-y} keyletters must also be present. Omission of these keyletters causes an error to occur.

Comments are limited to text strings of at most 512 characters.

\textbf{SEE ALSO}

\texttt{admin(1), bdiff(1), cdc(1), get(1), help(1), prs(1), rmdel(1), sccsfile(4)}.

\textbf{DIAGNOSTICS}

Use \texttt{help(1)} for explanations.
NAME
deroff – remove nroff/troff, tbl, and eqn constructs

SYNOPSIS
deroff [-mx] [-w] [ files ]

DESCRIPTION
deroff reads each of the files in sequence and removes all troff(1) requests, macro calls, backslash constructs, eqn(1) constructs (between .EQ and .EN lines, and between delimiters), and tbl(1) descriptions, perhaps replacing them with white space (blanks and blank lines), and writes the remainder of the file on the standard output. deroff follows chains of included files (.so and .nx troff commands); if a file has already been included, a .so naming that file is ignored and a .nx naming that file terminates execution. If no input file is given, deroff reads the standard input.

The -m option may be followed by an m, s, or l. The -mm option causes the macros to be interpreted so that only running text is output (i.e., no text from macro lines.) The -ml option forces the -mm option and also causes deletion of lists associated with the mm macros.

If the -w option is given, the output is a word list, one "word" per line, with all other characters deleted. Otherwise, the output follows the original, with the deletions mentioned above. 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, however, 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."

SEE ALSO
eqn(1), nroff(1), tbl(1), troff(1).

BUGS
deroff is not a complete troff interpreter, so it can be confused by subtle constructs. Most such errors result in too much rather than too little output.
The -ml option does not handle nested lists correctly.
NAME

devnm – device name

SYNOPSIS

/etc/devnm [ names ]

DESCRIPTION

devnm identifies the special file associated with the mounted file system where the argument name resides.

EXAMPLE

The command:

    /etc/devnm /usr

produces

    /dev/dsk/c1d0s2 usr

if /usr is mounted on /dev/dsk/c1d0s2.

FILES

    /dev/dsk/*
    /etc/mnttab
NAME

df – report number of free disk blocks and i-nodes

SYNOPSIS

df [-l] [-f] [-b] [file-system | directory | mounted-resource]

DESCRIPTION

The df command prints out the number of free blocks and free i-nodes in mounted file systems, directories, or mounted resources by examining the counts kept in the super-blocks.

file-system may be specified either by device name (e.g., /dev/dsk/c1d0s2) or by mount point directory name (e.g., /usr).

directory can be a directory name. The report presents information for the device that contains the directory.

mounted-resource can be a remote resource name. The report presents information for the remote device that contains the resource.

If no arguments are used, the free space on all locally and remotely mounted file systems is printed.

The df command uses the following options:

-1 only reports on local file systems.

-t causes the figures for total allocated blocks and i-nodes to be reported as well as the free blocks and i-nodes.

-f an actual count of the blocks in the free list is made, rather than taking the figure from the super-block (free i-nodes are not reported). This option will not print any information about mounted remote resources.

-b produce output in 4.3BSD format.

NOTE

If multiple remote resources are listed that reside on the same file system on a remote machine, each listing after the first one will be marked with an asterisk.

FILES

/dev/dsk/*

/etc/mnttab

SEE ALSO

fs(4), mnttab(4), mount(1M).
**NAME**

dconf, dflist, dfreset – disk farms and administration

**SYNOPSIS**

dconf [-c] ctrldev secs cnt dev ...
dlist dev [ dev ... ]
dreset ctrldev [ ctrldev ... ]

**DESCRIPTION**

Disk farming allows one or more disk partitions on one or more disk drives to be combined into one logical disk partition. Each disk farm is represented by three devices: a block access device (found in /dev/farm), a raw access device (found in /dev/rfarm), and a controlling device (found in /dev/farmctrl). The controlling device is used when configuring, resetting, or listing a farm; the raw access device can also be used to list the farm. Once configured, the block and raw devices behave as any other disk device.

Each disk partition in the farm is divided into one or more equal sized groups of logically consecutive sectors called _leaves_. A farm is arranged such that sectors 0 through N-1 map to the 0th leaf of the first partition, sectors N through 2N-1 map to the 0th leaf of the second partition, etc., each disk in turn contributing the next block of N sectors. Each partition contributes the same number of leaves, and all leaves are the same number of sectors.

`dconf` configures a disk farm for use. If `-c` is the first argument, the disk farm configuration will be listed after a successful completion. `ctrldev` must be the `/dev/farmctrl` corresponding to the farm to be configured. `secs` is how many sectors (NBPSCTR bytes per sector) in a leaf. The only logical constraint on this number is that the number of bytes in a leaf must be a multiple of the access size. For example, if the farm will be read and written in 4Kbyte pieces, `secs` must be a multiple of (4K/NBPSCTR). `cnt` is the total number of _leaves_ contributed by each partition. The remaining arguments are the block special device file(s) which make up the farm.

**N.B.** No checking is done during configuration, so it is possible to have a successful configuration and not be able to use the farm.

`dfreset` clears the state of the named farm(s), as in preparation for system shutdown or to reconfigure a farm. The specified device(s) must be the `/dev/farmctrl` corresponding to the farm to be reset. If the farm is currently open by a process or mounted, the reset will fail.

`dflist` lists the configuration for the named farm(s). The specified device(s) may be either the corresponding `/dev/farmctrl` entry or the appropriate `/dev/rfarm` character special file.

Both `dconf` and `dfreset` may only be invoked by the super user. `dflist` may be invoked by any user, provided they also have read/write permission for the necessary farm device(s).
NAME
diff – differential file comparator

SYNOPSIS
diff [ -efbh ] file1 file2

DESCRIPTION
diff tells what lines must be changed in two files to bring them into agreement. If file1 (file2) is ~, the standard input is used. If file1 (file2) is a directory, then a file in that directory with the name file2 (file1) is used. The normal output contains lines of these forms:

n1 a n3,n4
n1,n2 d n3
n1,n2 c n3,n4

These lines resemble ed commands to convert file1 into file2. The numbers after the letters pertain to file2. In fact, by exchanging a for d and reading backward one may ascertain equally how to convert file2 into file1. As in ed, identical pairs, where n1 = n2 or n3 = n4, are abbreviated as a single number.

Following each of these lines come all the lines that are affected in the first file flagged by <, then all the lines that are affected in the second file flagged by >.

The -b option causes trailing blanks (spaces and tabs) to be ignored and other strings of blanks to compare equal.

The -e option produces a script of a, c, and d commands for the editor ed, which will recreate file2 from file1. The -f option produces a similar script, not useful with ed, in the opposite order. In connection with -e, the following shell program may help maintain multiple versions of a file. Only an ancestral file ($1) and a chain of version-to-version ed scripts ($2,$3,...) made by diff need be on hand. A “latest version” appears on the standard output.

(shift; cat $*; echo '1,$p') | ed $1

Except in rare circumstances, diff finds a smallest sufficient set of file differences.

Option -h does a fast, half-hearted job. It works only when changed stretches are short and well separated, but does work on files of unlimited length. Options -e and -f are unavailable with -h.

FILES
/tmp/d?????
/usr/lib/diffh for -h

SEE ALSO
bdiff(1), cmp(1), comm(1), ed(1).

DIAGNOSTICS
Exit status is 0 for no differences, 1 for some differences, 2 for trouble.

BUGS
Editing scripts produced under the -e or -f option are naive about creating lines consisting of a single period (.)

WARNINGS
Missing newline at end of file X indicates that the last line of file X did not have a new-line. If the lines are different, they will be flagged and output; although the output will seem to indicate they are the same.
NAME
diff – differential file and directory comparator

SYNOPSIS
diff [-l] [-r] [-s] [-cefhn] [-biwt] dir1 dir2
diff [-cefhn] [-biwt] file1 file2
diff [-Dstring] [-biw] file1 file2

DESCRIPTION
If both arguments are directories, diff sorts the contents of the directories by name, and then runs the regular file diff algorithm (described below) on text files which are different. Binary files which differ, common subdirectories, and files which appear in only one directory are listed. Options when comparing directories are:

-\l long output format; each text file diff is piped through pr(1) to paginate it, other differences are remembered and summarized after all text file differences are reported.
-\r causes application of diff recursively to common subdirectories encountered.
-\s causes diff to report files which are the same, which are otherwise not mentioned.
-\Sname starts a directory diff in the middle beginning with file name.

When run on regular files, and when comparing text files which differ during directory comparison, diff tells what lines must be changed in the files to bring them into agreement. Except in rare circumstances, diff finds a smallest sufficient set of file differences. If neither file1 nor file2 is a directory, then either may be given as ‘-’, in which case the standard input is used. If file1 is a directory, then a file in that directory whose file-name is the same as the file-name of file2 is used (and vice versa).

There are several options for output format; the default output format contains lines of these forms:

\[ n1 a n3,n4 \]
\[ n1,n2 d n3 \]
\[ n1,n2 c n3,n4 \]

These lines resemble ed commands to convert file1 into file2. The numbers after the letters pertain to file2. In fact, by exchanging ‘a’ for ‘d’ and reading backward one may ascertain equally how to convert file2 into file1. As in ed, identical pairs where \( n1 = n2 \) or \( n3 = n4 \) are abbreviated as a single number.

Following each of these lines come all the lines that are affected in the first file flagged by ‘<’, then all the lines that are affected in the second file flagged by ‘>’.

Except for -b, -w, -i or -t which may be given with any of the others, the following options are mutually exclusive:

-\e produces a script of a, c and d commands for the editor ed, which will recreate file2 from file1. In connection with -e, the following shell program may help maintain multiple versions of a file. Only an ancestral file ($1) and a chain of version-to-version ed scripts ($2,$3,...) made by diff need be on hand. A ‘latest version’ appears on the standard output.

\[(\text{shift; cat $*; echo '1,$p'} ) | \text{ed - } 1\]

Extra commands are added to the output when comparing directories with -e, so that the result is a sh(1) script for converting text files which are common to the two directories from their state in dir1 to their state in dir2.
-f produces a script similar to that of -e, not useful with ed, and in the opposite order.

-n produces a script similar to that of -e, but in the opposite order and with a count of changed lines on each insert or delete command. This is the form used by rcsdiff(1).

-c produces a diff with lines of context. The default is to present 3 lines of context and may be changed, e.g. to 10, by -c10. With -c the output format is modified slightly: the output beginning with identification of the files involved and their creation dates and then each change is separated by a line with a dozen *'s. The lines removed from file1 are marked with '−'; those added to file2 are marked '+' . Lines which are changed from one file to the other are marked in both files with with '!' .

Changes which lie within <context> lines of each other are grouped together on output. (This is a change from the previous "diff -c" but the resulting output is usually much easier to interpret.)

-h does a fast, half-hearted job. It works only when changed stretches are short and well separated, but does work on files of unlimited length.

-Dstring causes diff to create a merged version of file1 and file2 on the standard output, with C preprocessor controls included so that a compilation of the result without defining string is equivalent to compiling file1, while defining string will yield file2.

-b causes trailing blanks (spaces and tabs) to be ignored, and other strings of blanks to compare equal.

-w is similar to -b but causes whitespace (blanks and tabs) to be totally ignored. E.g., "if ( a == b )" will compare equal to "if(a==b)".

-i ignores the case of letters. E.g., "A" will compare equal to "a".

-t will expand tabs in output lines. Normal or -c output adds character(s) to the front of each line which may screw up the indentation of the original source lines and make the output listing difficult to interpret. This option will preserve the original source's indentation.

FILES

/tmp/d?????
/usr/lib/diffh for -h
/bin/diff for directory diffs
/bin/pr

SEE ALSO

cmp(1), cc(1), comm(1), ed(1), diff3(1)

DIAGNOSTICS

Exit status is 0 for no differences, 1 for some, 2 for trouble.

BUGS

Editing scripts produced under the -e or -f option are naive about creating lines consisting of a single '·'.

When comparing directories with the -b, -w or -i options specified, diff first compares the files a la cmp, and then decides to run the diff algorithm if they are not equal. This may cause a small amount of spurious output if the files then turn out to be identical because the only differences are insignificant blank string or case differences.
NAME

diff3 – 3-way differential file comparison

SYNOPSIS

diff3 [-e3] file1 file2 file3

DESCRIPTION

diff3 compares three versions of a file, and publishes disagreeing ranges of text flagged with these codes:

```
====          all three files differ
====1         file1 is different
====2         file2 is different
====3         file3 is different
```

The type of change suffered in converting a given range of a given file to some other is indicated in one of these ways:

```
f : n1 a      Text is to be appended after line number n1 in file f, where f = 1, 2, or 3.

f : n1 , n2 c Text is to be changed in the range line n1 to line n2. 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, the contents of the lower-numbered file is suppressed.

Under the -e option, diff3 publishes a script for the editor ed that will incorporate into file1 all changes between file2 and file3, i.e., the changes that normally would be flagged ==== and ====3. Option -x (-3) produces a script to incorporate only changes flagged ==== (= ====3). The following command will apply the resulting script to file1.

```
(cat script; echo '1,$p') | ed - file1
```

FILES

```
/tmp/d3*
/usr/lib/diff3prog
```

SEE ALSO

diff(1).

BUGS

Text lines that consist of a single . will defeat -e. Files longer than 64K bytes will not work.
NAME
dircmp – directory comparison

SYNOPSIS
dircmp [-d] [-s] [-wn] dir1 dir2

DESCRIPTION
dircmp examines dir1 and dir2 and generates various tabulated information about the contents of the directories. Listings of files that are unique to each directory are generated for all the options. If no option is entered, a list is output indicating whether the file names common to both directories have the same contents.

-d  Compare the contents of files with the same name in both directories and output a list telling what must be changed in the two files to bring them into agreement. The list format is described in diff(1).

-s  Suppress messages about identical files.

-wn  Change the width of the output line to n characters. The default width is 72.

SEE ALSO
cmp(1), diff(1).
NAME
diskusg – generate disk accounting data by user ID

SYNOPSIS
diskusg [options] [files]

DESCRIPTION
diskusg generates intermediate disk accounting information from data in files, or the
standard input if omitted. diskusg output lines on the standard output, one per user,
in the following format: uid login #blocks

where
uid the numerical user ID of the user.
login the login name of the user; and
#blocks the total number of disk blocks allocated to this user.
diskusg normally reads only the inodes of file systems for disk accounting. In this
case, files are the special filenames of these devices.
diskusg recognizes the following options:

- s the input data is already in diskusg output format. diskusg combines all
  lines for a single user into a single line.
- v verbose. Print a list on standard error of all files that are charged to no
  one.
- f fnmlist ignore the data on those file systems whose file system name is in
  fnmlist. fnmlist is a list of file system names separated by commas or
  enclosed within quotes. diskusg compares each name in this list with
  the file system name stored in the volume ID (see labelit(1M)).
- p file use file as the name of the password file to generate login names.
  /etc/passwd is used by default.
- u file write records to file of files that are charged to no one. Records consist
  of the special file name, the inode number, and the user ID.

The output of diskusg is normally the input to acctdisk (see acct(1M)) which generates
total accounting records that can be merged with other accounting records. diskusg is
normally run in dodisk (see acctsh(1M)).

EXAMPLES
The following will generate daily disk accounting information for root on
/dev/dsk/c1d0s0:
diskusg /dev/dsk/c1d0s0 | acctdisk > disktacct

FILES
/etc/passwd used for user ID to login name conversions

SEE ALSO
acct(1M), acct(4), acctsh(1M)
NAME

du – summarize disk usage

SYNOPSIS

du [ -sar ] [ names ]

DESCRIPTION

du reports the number of 1024-byte logical blocks contained in all files and (recursively) directories within each directory and file specified by the names argument. The block count includes the indirect blocks of the file. If names is missing, the current directory is used.

The optional arguments are as follows:
- -s causes only the grand total (for each of the specified names) to be given.
- -a causes an output line to be generated for each file.

If neither -s or -a is specified, an output line is generated for each directory only.
- -r will cause du to generate messages about directories that cannot be be read, files that cannot be opened, etc., rather than being silent (the default).

A file with two or more links is only counted once.

BUGS

If the -a option is not used, non-directories given as arguments are not listed.
If there are links between files in different directories where the directories are on separate branches of the file system hierarchy, du will count the excess files more than once.
Files with holes in them will get an incorrect block count.
NAME
dvhtool - modify disk volume header information

SYNOPSIS
/etc/dvhtool [ modify part nblk 1st_blk type ] [ list ]
    [-v [ add unix_file doh_file ] [ creat unix_file doh_file ]
    [ delete doh_file ] [ list ]]
    [-d [ modify name_value ] [ list ]]
    [ doh_file -m partition_no filesystem_type ]

DESCRIPTION
dvhtool allows modification of the disk volume header information, where the disk
volume header is a block located at the beginning of all disk media. The disk volume
header consists of three main parts: the device parameters, the partition table, and
the volume directory. The volume directory is used to locate such things as the boot
block and the bad sector table. The partition table describes the logical device parti-
tions. The device parameters describe the specifics of a particular disk drive.

Invoked with no arguments, dvhtool allows the user to examine and modify the disk
volume header interactively. The read command reads the volume header from the
specified device. The vd, pt, and dp commands first list their respective portions of
the volume header and then prompt for modifications. The write command writes
the possibly modified volume header to the device.

Invoked with arguments, dvhtool reads the volume header, performs the specified
operations, and then writes the volume header. The following paragraphs describe
dvhtool’s command line arguments.

The -v flag provides four options for modifying the volume directory information in
the disk volume header. The creat option allows creation of a volume directory with
the name doh_name and the contents of unix_file. If an entry already exists with the
name doh_file, it is overwritten with the new contents. The add option adds a volume
directory entry with the name doh_file and the contents of unix_file. Unlike the creat
option, the add options do not overwrite an existing entry. The delete option
removes the entry named doh_file, if it exists, from the volume directory. The list
option lists the current volume directory contents.

The -m flag allows dvhtool to invoke mkfs and make a file system of the type
filesystem_type (either 4K for System V or AFFS for Ardent Fast File System) on parti-
tion partition_no on the drive named by doh_file. Mkfs will be invoked with
default parameters which covers the entire partition.

The -d flag provides two options for modifying the device parameter information in
the disk volume header. The modify options sets the name device parameter to the
specified value1. The list option lists the current device parameters.

SEE ALSO
mkfs(1M)
**NAME**

`echo` – echo arguments

**SYNOPSIS**

```
/bin/echo [ arg ] ...
```

**DESCRIPTION**

`echo` writes its arguments separated by blanks and terminated by a new-line on the standard output. It also understands C-like escape conventions; beware of conflicts with the shell's use of `\`:

```
\b   backspace
\c   print line without new-line
\f   form-feed
\n   new-line
\r   carriage return
\t   tab
\v   vertical tab
\\  backslash
\0n  where `n` is the 8-bit character whose ASCII code is the 1-, 2- or 3-digit octal number representing that character.
```

`echo` is useful for producing diagnostics in command files and for sending known data into a pipe.

**SEE ALSO**

`sh(1), csh(1)`

**CAVEATS**

When representing an 8-bit character by using the escape convention `\0n`, the `n` must always be preceded by the digit zero (0).

For example, typing: `echo 'WARNING:\07'` will print the phrase WARNING: and sound the “bell” on your terminal. The use of single (or double) quotes (or two backslashes) is required to protect the “\” that precedes the “07”.

For the octal equivalents of each character, see `ascii(5)`.

**WARNING**

`echo` is a built-in command to both `sh(1)` and `csh(1)` and the syntax varies.
NAME

ed, red – text editor

SYNOPSIS

ed [-s] [-p string ] [-x] [file]
red [-s] [-p string ] [-x] [file]

DESCRIPTION

ed is the standard text editor. If the file argument is given, ed simulates an e command (see below) on the named file; that is to say, the file is read into ed's buffer so that it can be edited.

-s Suppresses the printing of character counts by e, r, and w commands, of diagnostics from e and q commands, and of the ! prompt after a !sh command. Also, see the WARNING section at the end of this manual page.

-p Allows the user to specify a prompt string.

-x Encryption option; when this option is used, the file will be encrypted as it is being written and will require an encryption key to be read (see crypt(1)). Also, see the WARNING section at the end of this manual page.

ed operates on a copy of the file it is editing; changes made to the copy have no effect on the file until a w (write) command is given. The copy of the text being edited resides in a temporary file called the buffer. There is only one buffer.

red is a restricted version of ed. It will only allow editing of files in the current directory. It prohibits executing shell commands via !sh command. Attempts to bypass these restrictions result in an error message (restricted shell).

Both ed and red support the fspec(4) formatting capability. After including a format specification as the first line of file and invoking ed with your terminal in stty -tabs or stty tab3 mode (see stty(1)), the specified tab stops will automatically be used when scanning file. For example, if the first line of a file contained: <t5,10,15 872:> tab stops would be set at columns 5, 10, and 15, and a maximum line length of 72 would be imposed. NOTE: while inputing text, tab characters when typed are expanded to every eighth column as is the default.

Commands to ed have a simple and regular structure: zero, one, or two addresses followed by a single-character command, possibly followed by parameters to that command. These addresses specify one or more lines in the buffer. Every command that requires addresses has default addresses, so that the addresses can very often be omitted.

In general, only one command may appear on a line. Certain commands allow the input of text. This text is placed in the appropriate place in the buffer. While ed is accepting text, it is said to be in input mode. In this mode, no commands are recognized; all input is merely collected. Input mode is left by typing a period (.) alone at the beginning of a line, followed immediately by a carriage return.

ed supports a limited form of regular expression notation; regular expressions are used in addresses to specify lines and in some commands (e.g., s) to specify portions of a line that are to be substituted. A regular expression (RE) specifies a set of character strings. A member of this set of strings is said to be matched by the RE. The REs allowed by ed are constructed as follows:

The following one-character REs match a single character:

1.1 An ordinary character (not one of those discussed in 1.2 below) is a one-character RE that matches itself.
1.2 A backslash (\) followed by any special character is a one-character RE that matches the special character itself. The special characters are:

a. ., *, [ and \ (period, asterisk, left square bracket, and backslash, respectively), which are always special, except when they appear within square brackets ([)]; see 1.4 below).

b. \^ (caret or circumflex), which is special at the beginning of an entire RE (see 3.1 and 3.2 below), or when it immediately follows the left of a pair of square brackets ([)] (see 1.4 below).

c. \$ (dollar sign), which is special at the end of an entire RE (see 3.2 below).

d. The character used to bound (i.e., delimit) an entire RE, which is special for that RE (for example, see how slash (/) is used in the g command, below.)

1.3 A period (.) is a one-character RE that matches any character except new-line.

1.4 A non-empty string of characters enclosed in square brackets ([]) is a one-character RE that matches any one character in that string. If, however, the first character of the string is a circumflex (^), the one-character RE matches any character except new-line and the remaining characters in the string. The ^ has this special meaning only if it occurs first in the string. The minus (-) may be used to indicate a range of consecutive ASCII characters; for example, [0-9] is equivalent to [0123456789]. The – loses this special meaning if it occurs first (after an initial ^, if any) or last in the string. The right square bracket (]) does not terminate such a string when it is the first character within it (after an initial ^, if any); e.g., [a-f] matches either a right square bracket (]) or one of the letters a through f inclusive. The four characters listed in 1.2.a above stand for themselves within such a string of characters.

The following rules may be used to construct REs from one-character REs:

2.1 A one-character RE is a RE that matches whatever the one-character RE matches.

2.2 A one-character RE followed by an asterisk (*) is a RE that matches zero or more occurrences of the one-character RE. If there is any choice, the longest leftmost string that permits a match is chosen.

2.3 A one-character RE followed by \{m\}, \{m,n\}, or \{m,n\} is a RE that matches a range of occurrences of the one-character RE. The values of m and n must be non-negative integers less than 256; \{m\} matches exactly m occurrences; \{m,n\} matches at least m occurrences; \{m,n\} matches any number of occurrences between m and n inclusive. Whenever a choice exists, the RE matches as many occurrences as possible.

2.4 The concatenation of REs is a RE that matches the concatenation of the strings matched by each component of the RE.

2.5 A RE enclosed between the character sequences \ and \ is a RE that matches whatever the unadorned RE matches.

2.6 The expression \n matches the same string of characters as was matched by an expression enclosed between \ and \ earlier in the same RE. Here n is a digit; the sub-expression specified is that beginning with the n-th occurrence of \ counting from the left. For example, the expression ^{.*}\\1$ matches a line consisting of two repeated appearances of the same string.

Finally, an entire RE may be constrained to match only an initial segment or final segment of a line (or both).
3.1 A circumflex ($\wedge$) at the beginning of an entire RE constrains that RE to match an *initial* segment of a line.

3.2 A dollar sign ($) at the end of an entire RE constrains that RE to match a *final* segment of a line.

The construction $\wedge$ *entire RE* $\$*$ constrains the entire RE to match the entire line.

The null RE (e.g., /*/) is equivalent to the last RE encountered. See also the last paragraph before FILES below.

To understand addressing in ed it is necessary to know that at any time there is a *current line*. Generally speaking, the current line is the last line affected by a command; the exact effect on the current line is discussed under the description of each command. *Addresses* are constructed as follows:

1. The character . addresses the current line.
2. The character $\$*$ addresses the last line of the buffer.
3. A decimal number $n$ addresses the $n$-th line of the buffer.
4. 'x addresses the line marked with the mark name character x, which must be a lower-case letter. Lines are marked with the k command described below.
5. A RE enclosed by slashes (/) addresses the first line found by searching *forward* from the line following the current line toward the end of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the beginning of the buffer and continues up to and including the current line, so that the entire buffer is searched. See also the last paragraph before FILES below.
6. A RE enclosed in question marks (?) addresses the first line found by searching *backward* from the line preceding the current line toward the beginning of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the end of the buffer and continues up to and including the current line. See also the last paragraph before FILES below.
7. An address followed by a plus sign (+) or a minus sign (−) followed by a decimal number specifies that address plus (respectively minus) the indicated number of lines. The plus sign may be omitted.
8. If an address begins with + or −, the addition or subtraction is taken with respect to the current line; e.g., −5 is understood to mean −5.
9. If an address ends with + or −, then 1 is added to or subtracted from the address, respectively. As a consequence of this rule and of Rule 8, immediately above, the address − refers to the line preceding the current line. (To maintain compatibility with earlier versions of the editor, the character $\wedge$ in addresses is entirely equivalent to −.) Moreover, trailing + and − characters have a cumulative effect, so −− refers to the current LINE LESS 2.
10. For convenience, a comma (,) stands for the address pair 1,$\$*$, while a semicolon (;) stands for the pair ,$\$*$.

Commands may require zero, one, or two addresses. Commands that require no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when an insufficient number of addresses is given; if more addresses are given than such a command requires, the last one(s) are used.

Typically, addresses are separated from each other by a comma (,). They may also be separated by a semicolon (;). In the latter case, the current line (.) is set to the first address, and only then is the second address calculated. This feature can be used to
determine the starting line for forward and backward searches (see Rules 5 and 6, above). The second address of any two-address sequence must correspond to a line that follows, in the buffer, the line corresponding to the first address.

In the following list of ed commands, the default addresses are shown in parentheses. The parentheses are not part of the address; they show that the given addresses are the default.

It is generally illegal for more than one command to appear on a line. However, any command (except e, f, r, or w) may be suffixed by l, n, or p in which case the current line is either listed, numbered or printed, respectively, as discussed below under the l, n, and p commands.

\(.\)a
\(<\text{text}>\)

The append command reads the given text and appends it after the addressed line; . is left at the last inserted line, or, if there were none, at the addressed line. Address 0 is legal for this command: it causes the appended text to be placed at the beginning of the buffer. The maximum number of characters that may be entered from a terminal is 256 per line (including the new-line character).

\(.\)c
\(<\text{text}>\)

The change command deletes the addressed lines, then accepts input text that replaces these lines; . is left at the last line input, or, if there were none, at the first line that was not deleted.

\(\ldots\)d

The delete command deletes the addressed lines from the buffer. The line after the last line deleted becomes the current line; if the lines deleted were originally at the end of the buffer, the new last line becomes the current line.

e file

The edit command causes the entire contents of the buffer to be deleted, and then the named file to be read in; . is set to the last line of the buffer. If no file name is given, the currently-remembered file name, if any, is used (see the f command). The number of characters read is typed; file is remembered for possible use as a default file name in subsequent e, r, and w commands. If file is replaced by \(1\), the rest of the line is taken to be a shell \((sh(1))\) command whose output is to be read. Such a shell command is not remembered as the current file name. See also DIAGNOSTICS below.

E file

The Edit command is like e, except that the editor does not check to see if any changes have been made to the buffer since the last w command.

f file

If file is given, the f file-name command changes the currently-remembered file name to file; otherwise, it prints the currently-remembered file name.

\(1,\$g/RE/command list\)

In the global command, the first step is to mark every line that matches the given RE. Then, for every such line, the given command list is executed with . initially set to that line. A single command or the first of a list of commands appears on the same line as the global command. All lines of a multi-line list except the last line must be ended with a \(\backslash; a, i,\) and \(c\) commands and associated input are permitted. The . terminating input mode may be omitted if it would be the last line of the command list. An empty command list is equivalent
to the \textit{p} command. The \textit{g}, \textit{G}, \textit{v}, and \textit{V} commands are \textit{not} permitted in the \textit{command list}. See also \textit{BUGS} and the last paragraph before \textit{FILES} below.

\texttt{(1,§)G/RE/}

In the interactive \textit{Global} command, the first step is to mark every line that matches the given \textit{RE}. Then, for every such line, that line is printed, . is changed to that line, and any \textit{one} command (other than one of the \textit{a}, \textit{c}, \textit{i}, \textit{g}, \textit{G}, \textit{v}, and \textit{V} commands) may be input and is executed. After the execution of that command, the next marked line is printed, and so on; a new-line acts as a null command; an \& causes the re-execution of the most recent command executed within the current invocation of \textit{G}. Note that the commands input as part of the execution of the \textit{G} command may address and affect any lines in the buffer. The \textit{G} command can be terminated by an interrupt signal (ASCII DEL or BREAK).

\texttt{h}

The \textit{help} command gives a short error message that explains the reason for the most recent \textit{?} diagnostic.

\texttt{H}

The \textit{Help} command causes \textit{ed} to enter a mode in which error messages are printed for all subsequent \textit{?} diagnostics. It will also explain the previous \textit{?} if there was one. The \textit{H} command alternately turns this mode on and off; it is initially off.

\texttt{(.)j}

\texttt{<text> .}

The insert command inserts the given text before the addressed line; . is left at the last inserted line, or, if there were none, at the addressed line. This command differs from the \textit{a} command only in the placement of the input text. Address 0 is not legal for this command. The maximum number of characters that may be entered from a terminal is 256 per line (including the new-line character).

\texttt{(.,+1)}

The \textit{join} command joins contiguous lines by removing the appropriate new-line characters. If exactly one address is given, this command does nothing.

\texttt{(.,)kx}

The mark command marks the addressed line with name \textit{x}, which must be a lower-case letter. The address \textit{'x} then addresses this line; . is unchanged.

\texttt{(.,.)l}

The list command prints the addressed lines in an unambiguous way: a few non-printing characters (e.g., \textit{tab}, \textit{backspace}) are represented by visually mnemonic overstrikes. All other non-printing characters are printed in octal, and long lines are folded. An \textit{l} command may be appended to any other command other than \textit{e}, \textit{f}, \textit{r}, or \textit{w}.

\texttt{(.,.)m}

The move command repositions the addressed line(s) after the line addressed by \textit{a}. Address 0 is legal for \textit{a} and causes the addressed line(s) to be moved to the beginning of the file. It is an error if address \textit{a} falls within the range of moved lines; . is left at the last line moved.

\texttt{(.,.)n}

The number command prints the addressed lines, preceding each line by its line number and a tab character; . is left at the last line printed. The \textit{n} command may be appended to any other command other than \textit{e}, \textit{f}, \textit{r}, or \textit{w}.
The print command prints the addressed lines; . is left at the last line printed. The p command may be appended to any other command other than e, f, r, or w. For example, dp deletes the current line and prints the new current line.

The editor will prompt with a * for all subsequent commands. The P command alternately turns this mode on and off; it is initially off.

The quit command causes ed to exit. No automatic write of a file is done; however, see DIAGNOSTICS, below.

The editor exits without checking if changes have been made in the buffer since the last w command.

($)r file
The read command reads in the given file after the addressed line. If no file name is given, the currently-remembered file name, if any, is used (see e and f commands). The currently-remembered file name is not changed unless file is the very first file name mentioned since ed was invoked. Address 0 is legal for r and causes the file to be read at the beginning of the buffer. If the read is successful, the number of characters read is typed; . is set to the last line read in. If file is replaced by !, the rest of the line is taken to be a shell (sh(1)) command whose output is to be read. For example, "$! ls" appends current directory to the end of the file being edited. Such a shell command is not remembered as the current file name.

(...)/s/RE/replacement/
(...)/s/RE/replacement/g
(...)/s/RE/replacement\n
The substitute command searches each addressed line for an occurrence of the specified RE. In each line in which a match is found, all (non-overlapped) matched strings are replaced by the replacement if the global replacement indicator g appears after the command. If the global indicator does not appear, only the first occurrence of the matched string is replaced. If a number n appears after the command, only the n-th occurrence of the matched string on each addressed line is replaced. It is an error for the substitution to fail on all addressed lines. Any character other than space or new-line may be used instead of / to delimit the RE and the replacement; . is left at the last line on which a substitution occurred. See also the last paragraph before FILES below.

An ampersand (&) appearing in the replacement is replaced by the string matching the RE on the current line. The special meaning of & in this context may be suppressed by preceding it by \. As a more general feature, the characters \n, where n is a digit, are replaced by the text matched by the n-th regular subexpression of the specified RE enclosed between \ and \. When nested parenthesized subexpressions are present, n is determined by counting occurrences of \ starting from the left. When the character % is the only character in the replacement, the replacement used in the most recent substitute command is used as the replacement in the current substitute command. The % loses its special meaning when it is in a replacement string of more than one character or is preceded by a \.

A line may be split by substituting a new-line character into it. The new-line in the replacement must be escaped by preceding it by \. Such substitution cannot be done as part of a g or v command list.
This command acts just like the m command, except that a copy of the addressed lines is placed after address a (which may be 0); . is left at the last line of the copy.

The undo command nullifies the effect of the most recent command that modified anything in the buffer, namely the most recent a, c, d, g, i, j, m, r, s, t, v, G, or V command.

This command is the same as the global command g except that the command list is executed with . initially set to every line that does not match the RE.

This command is the same as the interactive global command G except that the lines that are marked during the first step are those that do not match the RE.

The write command writes the addressed lines into the named file. If the file does not exist, it is created with mode 666 (readable and writable by everyone), unless your umask setting (see umask(1)) dictates otherwise. The currently-remembered file name is not changed unless file is the very first file name mentioned since ed was invoked. If no file name is given, the currently-remembered file name, if any, is used (see e and f commands); . is unchanged. If the command is successful, the number of characters written is typed. If file is replaced by !, the rest of the line is taken to be a shell (sh(1)) command whose standard input is the addressed lines. Such a shell command is not remembered as the current file name.

An encryption key is requested from the standard input. Subsequent e, r, and w commands will use this key to encrypt or decrypt the text (see crypt(1)). An explicitly empty key turns off encryption. Also, see the --x option of ed.

The line number of the addressed line is typed; . is unchanged by this command.

The remainder of the line after the ! is sent to the UNIX system shell (sh(1)) to be interpreted as a command. Within the text of that command, the unescaped character % is replaced with the remembered file name; if a ! appears as the first character of the shell command, it is replaced with the text of the previous shell command. Thus, !! will repeat the last shell command. If any expansion is performed, the expanded line is echoed; . is unchanged.

An address alone on a line causes the addressed line to be printed. A new-line alone is equivalent to .+1p; it is useful for stepping forward through the buffer.

If an interrupt signal (ASCII DEL or BREAK) is sent, ed prints a ? and returns to its command level.

Some size limitations: 512 characters per line, 256 characters per global command list, and 64 characters per file name. The limit on the number of lines depends on the amount of user memory: each line takes 1 word.

When reading a file, ed discards ASCII NUL characters. Files (e.g., a.out) that contain characters not in the ASCII set (bit 8 on) cannot be edited by ed.
If a file is not terminated by a new-line character, ed adds one and outputs a message explaining what it did.

If the closing delimiter of a RE or of a replacement string (e.g., /) would be the last character before a new-line, that delimiter may be omitted, in which case the addressed line is printed. The following pairs of commands are equivalent:

- s/s1/s2  s/s1/s2/p
- g/s1    g/s1/p
- ?s1     ?s1?

FILES

/usr/tmp default directory for temporary work file.
$TMPDIR if this environmental variable is not null, its value is used in place of
/usr/tmp as the directory name for the temporary work file.
ed.hup work is saved here if the terminal is hung up.

DIAGNOSTICS

? for command errors.
?file for an inaccessible file.
(use the help and Help commands for detailed explanations).

If changes have been made in the buffer since the last w command that wrote the entire buffer, ed warns the user if an attempt is made to destroy ed’s buffer via the e or q commands. It prints ? and allows one to continue editing. A second e or q command at this point will take effect. The –s command-line option inhibits this feature.

SEE ALSO

edit(1), ex(1), grep(1), sed(1), sh(1), stty(1), umask(1), vi(1), fspec(4), regexp(5).

BUGS

A ! command cannot be subject to a g or a v command.
The ! command and the ! escape from the e, r, and w commands cannot be used if the editor is invoked from a restricted shell (see sh(1)).
The sequence \n in a RE does not match a new-line character.
Characters are masked to 7 bits on input.
If the editor input is coming from a command file (e.g., ed file < ed-cmd-file), the editor will exit at the first failure.

WARNINGS

The –x option is provided with the Security Administration Utilities, which is available only in the United States.
The – option, although supported in this release for upward compatibility, will no longer be supported in the next major release of the system. Convert shell scripts that use the – option to use the –s option, instead.
NAME

edit – text editor (variant of ex for casual users)

SYNOPSIS

edit [-r] [-x] name ...

DESCRIPTION

dit is a variant of the text editor ex recommended for new or casual users who wish to use a command-oriented editor.

- r Recover file after an editor or system crash.

- x Encryption option; when this option is used, the file will be encrypted as it is being written and will require an encryption key to be read (see crypt(1)). Also, see the WARNING section at the end of this manual page.

The following brief introduction should help you get started with edit. If you are using a CRT terminal you may want to learn about the display editor vi.

To edit the contents of an existing file you begin with the command “edit name” to the shell. edit makes a copy of the file which you can then edit, and tells you how many lines and characters are in the file. To create a new file, just make up a name for the file and try to run edit on it; you will cause an error diagnostic, but do not worry.

dit prompts for commands with the character ‘’, which you should see after starting the editor. If you are editing an existing file, then you will have some lines in edit’s buffer (its name for the copy of the file you are editing). Most commands to edit use its “current line” if you do not tell them which line to use. Thus if you say print (which can be abbreviated p) and hit carriage return (as you should after all edit commands) this current line will be printed. If you delete (d) the current line, edit will print the new current line. When you start editing, edit makes the last line of the file the current line. If you delete this last line, then the new last line becomes the current one. In general, after a delete, the next line in the file becomes the current line. (Deleting the last line is a special case.)

If you start with an empty file or wish to add some new lines, then the append (a) command can be used. After you give this command (typing a carriage return after the word append) edit will read lines from your terminal until you give a line consisting of just a “’”, placing these lines after the current line. The last line you type then becomes the current line. The command insert (i) is like append but places the lines you give before, rather than after, the current line.

dit numbers the lines in the buffer, with the first line having number 1. If you give the command “1” then edit will type this first line. If you then give the command delete edit will delete the first line, line 2 will become line 1, and edit will print the current line (the new line 1) so you can see where you are. In general, the current line will always be the last line affected by a command.

You can make a change to some text within the current line by using the substitute (s) command. You say “s/old/new/” where old is replaced by the old characters you want to get rid of and new is the new characters you want to replace it with.

The command file (f) will tell you how many lines there are in the buffer you are editing and will say “[Modified]” if you have changed it. After modifying a file you can put the buffer text back to replace the file by giving a write (w) command. You can then leave the editor by issuing a quit (q) command. If you run edit on a file, but do not change it, it is not necessary (but does no harm) to write the file back. If you try to quit from edit after modifying the buffer without writing it out, you will be warned that there has been “No write since last change” and edit will await another command. If you wish not to write the buffer out then you can issue another quit
command. The buffer is then irretrievably discarded, and you return to the shell.

By using the delete and append commands, and giving line numbers to see lines in
the file you can make any changes you desire. You should learn at least a few more
things, however, if you are to use edit more than a few times.

The change (c) command will change the current line to a sequence of lines you sup-
ply (as in append you give lines up to a line consisting of only a “.”). You can tell
change to change more than one line by giving the line numbers of the lines you
want to change, i.e., “3,5change”. You can print lines this way too. Thus “1,23p”
prints the first 23 lines of the file.

The undo (u) command will reverse the effect of the last command you gave which
changed the buffer. Thus if you give a substitute command which does not do what
you want, you can say undo and the old contents of the line will be restored. You
can also undo an undo command so that you can continue to change your mind. edit
will give you a warning message when commands you do affect more than one line
of the buffer. If the amount of change seems unreasonable, you should consider
doing an undo and looking to see what happened. If you decide that the change is
ok, then you can undo again to get it back. Note that commands such as write and
quit cannot be undone.

To look at the next line in the buffer you can just hit carriage return. To look at a
number of lines hit ^D (control key and, while it is held down D key, then let up
both) rather than carriage return. This will show you a half screen of lines on a CRT
or 12 lines on a hardcopy terminal. You can look at the text around where you are by
giving the command "z.". The current line will then be the last line printed; you can
get back to the line where you were before the "z." command by saying "z". The z
command can also be given other following characters "z-n" prints a screen of text (or
24 lines) ending where you are; "z+" prints the next screenful. If you want less than
a screenful of lines, type in "z.12" to get 12 lines total. This method of giving counts
works in general; thus you can delete 5 lines starting with the current line with the
command "delete 5".

To find things in the file, you can use line numbers if you happen to know them;
since the line numbers change when you insert and delete lines this is somewhat
unreliable. You can search backwards and forwards in the file for strings by giving
commands of the form /text/ to search forward for text or ?text? to search backward
for text. If a search reaches the end of the file without finding the text it wraps, end
around, and continues to search back to the line where you are. A useful feature here
is a search of the form /^text/ which searches for text at the beginning of a line. Similarly
/text$/ searches for text at the end of a line. You can leave off the trailing / or ?
in these commands.

The current line has a symbolic name "."; this is most useful in a range of lines as in
".,$print" which prints the rest of the lines in the file. To get to the last line in the file
you can refer to it by its symbolic name "$". Thus the command "$ delete" or "$d"
deletes the last line in the file, no matter which line was the current line before.
Arithmetic with line references is also possible. Thus the line "$-5" is the fifth before
the last, and "$+20" is 20 lines after the present.

You can find out which line you are at by doing "=.". This is useful if you wish to
move or copy a section of text within a file or between files. Find out the first and
last line numbers you wish to copy or move (say 10 to 20). For a move you can then
say "10,20delete a" which deletes these lines from the file and places them in a buffer
named a. edit has 26 such buffers named a through z. You can later get these lines
back by doing "put a" to put the contents of buffer a after the current line. If you
want to move or copy these lines between files you can give an edit (e) command.
after copying the lines, following it with the name of the other file you wish to edit, i.e., "edit chapter2". By changing delete to yank above you can get a pattern for copying lines. If the text you wish to move or copy is all within one file then you can just say "10,20move $" for example. It is not necessary to use named buffers in this case (but you can if you wish).

**SEE ALSO**
ed(1), ex(1), vi(1).

**WARNING**
The -x option is provided with the Security Administration Utilities, which is available only in the United States.
egrep – search a file for a pattern using full regular expressions

**SYNOPSIS**

egrep [options] full regular expression [file ...]

**DESCRIPTION**

egrep (expression grep) searches files for a pattern of characters and prints all lines that contain that pattern. egrep uses full regular expressions (expressions that have string values that use the full set of alphanumerics and special characters) to match the patterns. It uses a fast deterministic algorithm that sometimes needs exponential space.

egrep accepts full regular expressions as in ed(1), except for \( \) and \( \), with the addition of:

1. A full regular expression followed by + that matches one or more occurrences of the full regular expression.
2. A full regular expression followed by ? that matches 0 or 1 occurrences of the full regular expression.
3. Full regular expressions separated by | or by a new-line that match strings that are matched by any of the expressions.
4. A full regular expression that may be enclosed in parentheses () for grouping.

Be careful using the characters $, *, [, ^, |, (,), and \ in full regular expression, because they are also meaningful to the shell. It is safest to enclose the entire full regular expression in single quotes ‘...’.

The order of precedence of operators is [], then *?+, then concatenation, then | and new-line.

If no files are specified, egrep assumes standard input. Normally, each line found is copied to the standard output. The file name is printed before each line found if there is more than one input file.

Command line options are:

- **-b** Precede each line by the block number on which it was found. This can be useful in locating block numbers by context (first block is 0).
- **-c** Print only a count of the lines that contain the pattern.
- **-i** Ignore upper/lower case distinction during comparisons.
- **-l** Print the names of files with matching lines once, separated by new-lines. Does not repeat the names of files when the pattern is found more than once.
- **-n** Precede each line by its line number in the file (first line is 1).
- **-v** Print all lines except those that contain the pattern.
- **e special_expression**

  Search for a special expression (full regular expression that begins with a –).
- **-f file** Take the list of full regular expressions from file.

**SEE ALSO**

ed(1), fgrep(1), grep(1), sed(1), sh(1).

**DIAGNOSTICS**

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files (even if matches were found).

**BUGS**

Ideally there should be only one grep command, but there is not a single algorithm that spans a wide enough range of space-time tradeoffs. Lines are limited to BUFSIZE characters; longer lines are truncated. BUFSIZE is defined in /usr/include/stdio.h.
NAME

enable, disable – enable/disable LP printers

SYNOPSIS

enable printers
disable [-c] [-r{ reason }] printers

DESCRIPTION

enable activates the named printers, enabling them to print requests taken by lp(1). Use lpstat(1) to find the status of printers.

Disable deactivates the named printers, disabling them from printing requests taken by lp(1). By default, any requests that are currently printing on the designated printers will be reprinted in their entirety either on the same printer or on another member of the same class. Use lpstat(1) to find the status of printers. Options useful with disable are:

- c Cancel any requests that are currently printing on any of the designated printers.

- r{ reason } Associates a reason with the deactivation of the printers. This reason applies to all printers mentioned up to the next -r option. If the -r option is not present or the -r option is given without a reason, then a default reason will be used. Reason is reported by lpstat(1).

FILES

/usr/spool/lp/*

SEE ALSO

lp(1), lpstat(1).
NAME
env – set environment for command execution

SYNOPSIS
env [-] [ name=value ] ... [ command args ]

DESCRIPTION
env obtains the current environment, modifies it according to its arguments, then executes the command with the modified environment. Arguments of the form name=value are merged into the inherited environment before the command is executed. The - flag causes the inherited environment to be ignored completely, so that the command is executed with exactly the environment specified by the arguments.
If no command is specified, the resulting environment is printed, one name-value pair per line.

SEE ALSO
sh(1), exec(2), profile(4), environ(5).
NAME

error – analyze and disperse compiler error messages

SYNOPSIS

error [-n] [-s] [-q] [-v] [-t suffixlist] [-I ignorefile] [ name ]

DESCRIPTION

Error analyzes and optionally disperses the diagnostic error messages produced by a number of compilers and language processors to the source file and line where the errors occurred. It can replace the painful, traditional methods of scribbling abbreviations of errors on paper, and permits error messages and source code to be viewed simultaneously without machinations of multiple windows in a screen editor.

Error looks at the error messages, either from the specified file name or from the standard input, and attempts to determine which language processor produced each error message, determines the source file and line number to which the error message refers, determines if the error message is to be ignored or not, and inserts the (possibly slightly modified) error message into the source file as a comment on the line preceding to which the line the error message refers. Error messages which can’t be categorized by language processor or content are not inserted into any file, but are sent to the standard output. Error touches source files only after all input has been read. By specifying the -q query option, the user is asked to confirm any potentially dangerous (such as touching a file) or verbose action. Otherwise error proceeds on its merry business. If the -t touch option and associated suffix list is given, error will restrict itself to touch only those files with suffixes in the suffix list. Error also can be asked (by specifying -v) to invoke vi(1) on the files in which error messages were inserted; this obviates the need to remember the names of the files with errors.

Error is intended to be run with its standard input connected via a pipe to the error message source. Some language processors put error messages on their standard error file; others put their messages on the standard output. Hence, both error sources should be piped together into error. For example, when using the csh syntax,

    make -s lint | & error -q -v

will analyze all the error messages produced by whatever programs make runs when making lint.

Error knows about the error messages produced by: make, cc, ccpp, ccom, as, ld, lint, pi, pc, fj7, and DEC Western Research Modula-2. Error knows a standard format for error messages produced by the language processors, so is sensitive to changes in these formats. For all languages except Pascal, error messages are restricted to be on one line. Some error messages refer to more than one line in more than one files; error will duplicate the error message and insert it at all of the places referenced.

Error will do one of six things with error messages.

synchronize

Some language processors produce short errors describing which file it is processing. Error uses these to determine the file name for languages that don’t include the file name in each error message. These synchronization messages are consumed entirely by error.

discard

Error messages from lint that refer to one of the two lint libraries, /usr/lib/lib-lc and /usr/lib/lib-port are discarded, to prevent accidently touching these libraries. Again, these error messages are consumed entirely by error.

nullify

Error messages from lint can be nullified if they refer to a specific function, which is known to generate diagnostics which are not interesting. Nullified error messages are not inserted into the source file, but are
written to the standard output. The names of functions to ignore are
taken from either the file named .errorsrc in the user's home directory, or
from the file named by the -I option. If the file does not exist, no error
messages are nullified. If the file does exist, there must be one function
name per line.

not file specific
Error messages that can't be intuited are grouped together, and written to
the standard output before any files are touched. They will not be
inserted into any source file.

file specific Error message that refer to a specific file, but to no specific line, are written
to the standard output when that file is touched.

true errors Error messages that can be intuited are candidates for insertion into the
file to which they refer.

Only true error messages are candidates for inserting into the file they refer to. Other
error messages are consumed entirely by error or are written to the standard output.
Error inserts the error messages into the source file on the line preceding the line the
language processor found in error. Each error message is turned into a one line com-
ment for the language, and is internally flagged with the string "###" at the begin-
ing of the error, and "%%%" at the end of the error. This makes pattern searching
for errors easier with an editor, and allows the messages to be easily removed. In
addition, each error message contains the source line number for the line the message
refers to. A reasonably formatted source program can be recompiled with the error
messages still in it, without having the error messages themselves cause future
errors. For poorly formatted source programs in free format languages, such as C or
Pascal, it is possible to insert a comment into another comment, which can wreak
havoc with a future compilation. To avoid this, programs with comments and source
on the same line should be formatted so that language statements appear before com-
ments.

Options available with error are:

-n Do not touch any files; all error messages are sent to the standard output.

-q The user is queried whether s/he wants to touch the file. A "y" or "n" to the
question is necessary to continue. Absence of the -q option implies that all
referenced files (except those referring to discarded error messages) are to be
touched.

-v After all files have been touched, overlay the visual editor vi with it set up to
edit all files touched, and positioned in the first touched file at the first error. If
vi can't be found, try ex or ed from standard places.

-t Take the following argument as a suffix list. Files whose suffixes do not appear
in the suffix list are not touched. The suffix list is dot separated, and "*" wild-
cards work. Thus the suffix list:

".c,.y,.foo*.h"

allows error to touch files ending with "*.c", "*.y", "*.foo*" and "*.y".

-s Print out statistics regarding the error categorization. Not too useful.

Error catches interrupt and terminate signals, and if in the insertion phase, will ordi-
erly terminate what it is doing.

AUTHOR

Robert Henry
**FILES**

- `~/.errorrc` function names to ignore for lint error messages
- `/dev/tty` user's teletype

**BUGS**

Opens the teletype directly to do user querying.

Source files with links make a new copy of the file with only one link to it.

Changing a language processor's format of error messages may cause `error` to not understand the error message.

`Error`, since it is purely mechanical, will not filter out subsequent errors caused by 'floodgating' initiated by one syntactically trivial error. Humans are still much better at discarding these related errors.

Pascal error messages belong after the lines affected (error puts them before). The alignment of the ' 1 ' marking the point of error is also disturbed by `error`.

`Error` was designed for work on CRT's at reasonably high speed. It is less pleasant on slow speed terminals, and has never been used on hardcopy terminals.
NAME

ex – text editor

SYNOPSIS

ex [-][ -v ] [-t tag ] [-r ][ -R ][ -x ] [ +command ] name ...

DESCRIPTION

ex is the root of a family of editors: ex and vi. ex is a superset of ed, with the most
notable extension being a display editing facility. Display based editing is the focus
of vi.

If you have a CRT terminal, you may wish to use a display based editor; in this case
see vi(1), which is a command which focuses on the display editing portion of ex.

For ed Users

If you have used ed you will find that ex has a number of new features useful on CRT
terminals. Intelligent terminals and high speed terminals are very pleasant to use
with vi. Generally, the editor uses far more of the capabilities of terminals than ed
does, and uses the terminal capability data base (see Terminal Information Utilities
Guide) and the type of the terminal you are using from the variable TERM in the
environment to determine how to drive your terminal efficiently. The editor makes
use of features such as insert and delete character and line in its visual command
(which can be abbreviated vi) and which is the central mode of editing when using
vi(1).

ex contains a number of new features for easily viewing the text of the file. The z
command gives easy access to windows of text. Hitting ^D causes the editor to scroll
a half-window of text and is more useful for quickly stepping through a file than just
hitting return. Of course, the screen-oriented visual mode gives constant access to
editing context.

ex gives you more help when you make mistakes. The undo (u) command allows
you to reverse any single change which goes astray. ex gives you a lot of feedback,
normally printing changed lines, and indicates when more than a few lines are
affected by a command so that it is easy to detect when a command has affected
more lines than it should have.

The editor also normally prevents overwriting existing files unless you edited them
so that you do not accidentally clobber with a write a file other than the one you are
editing. If the system (or editor) crashes, or you accidentally hang up the telephone,
you can use the editor recover command to retrieve your work. This will get you
back to within a few lines of where you left off.

ex has several features for dealing with more than one file at a time. You can give it a
list of files on the command line and use the next (n) command to deal with each in
turn. The next command can also be given a list of file names, or a pattern as used by
the shell to specify a new set of files to be dealt with. In general, file names in the
editor may be formed with full shell metasyntax. The metacharacter ‘%’ is also avail-
able in forming file names and is replaced by the name of the current file.

For moving text between files and within a file the editor has a group of buffers,
named a through z. You can place text in these named buffers and carry it over when
you edit another file.

There is a command & in ex which repeats the last substitute command. In addition
there is a confirmed substitute command. You give a range of substitutions to be
done and the editor interactively asks whether each substitution is desired.

It is possible to ignore case of letters in searches and substitutions. ex also allows reg-
ular expressions which match words to be constructed. This is convenient, for exam-
ple, in searching for the word “edit” if your document also contains the word
"editor."

ex has a set of options which you can set to tailor it to your liking. One option which is very useful is the autoindent option which allows the editor to automatically supply leading white space to align text. You can then use the ^D key as a backtab and space and tab forward to align new code easily.

Miscellaneous new useful features include an intelligent join (j) command which supplies white space between joined lines automatically, commands < and > which shift groups of lines, and the ability to filter portions of the buffer through commands such as sort.

INVOCATION OPTIONS
The following invocation options are interpreted by ex:

- Suppress all interactive-user feedback. This is useful in processing editor scripts.

-v Invokes vi

-t tagfr Edit the file containing the tag and position the editor at its definition.

-r file Recover file after an editor or system crash. If file is not specified a list of all saved files will be printed.

-R Readonly mode set, prevents accidentally overwriting the file.

-x Encryption option; when this option is used, the file will be encrypted as it is being written and will require an encryption key to be read (see crypt(1)). Also, see the WARNING section at the end of this manual page.

+command Begin editing by executing the specified editor search or positioning command.

The name argument indicates files to be edited.

ex States
Command Normal and initial state. Input prompted for by . Your kill character cancels partial command.

Insert Entered by a, i, or c. Arbitrary text may be entered. Insert is normally terminated by a line having only . on it, or abnormally with an interrupt.

Visual Entered by vi, terminates with Q or ^\.

ex command names and abbreviations

<table>
<thead>
<tr>
<th>abbrev</th>
<th>ab</th>
<th>next</th>
<th>n</th>
<th>unabbrev</th>
<th>una</th>
</tr>
</thead>
<tbody>
<tr>
<td>append</td>
<td>a</td>
<td>number</td>
<td>nu</td>
<td>undo</td>
<td>u</td>
</tr>
<tr>
<td>args</td>
<td>ar</td>
<td></td>
<td></td>
<td>unmap</td>
<td>unm</td>
</tr>
<tr>
<td>change</td>
<td>c</td>
<td>preserve</td>
<td>pre</td>
<td>version</td>
<td>ve</td>
</tr>
<tr>
<td>copy</td>
<td>co</td>
<td>print</td>
<td>p</td>
<td>visual</td>
<td>vi</td>
</tr>
<tr>
<td>delete</td>
<td>d</td>
<td>put</td>
<td>pu</td>
<td>write</td>
<td>w</td>
</tr>
<tr>
<td>edit</td>
<td>e</td>
<td>quit</td>
<td>q</td>
<td>xit</td>
<td>x</td>
</tr>
<tr>
<td>file</td>
<td>f</td>
<td>read</td>
<td>re</td>
<td>yank</td>
<td>ya</td>
</tr>
<tr>
<td>global</td>
<td>g</td>
<td>recover</td>
<td>rec</td>
<td>window</td>
<td>z</td>
</tr>
<tr>
<td>insert</td>
<td>i</td>
<td>rewind</td>
<td>rew</td>
<td>escape</td>
<td>!</td>
</tr>
<tr>
<td>join</td>
<td>j</td>
<td>set</td>
<td>se</td>
<td>lshift</td>
<td>&lt;</td>
</tr>
<tr>
<td>list</td>
<td>l</td>
<td>shell</td>
<td>sh</td>
<td>print next</td>
<td>CR</td>
</tr>
</tbody>
</table>
Editing Utilities

EX(1)

map ma source so resubst &
mark ma stop st rshift >
move m substitute s scroll ^D

ex Command Addresses

n line n /pat next with pat
. current ?pat previous with pat
$ last x-n n before x
+ next x,y x through y
- previous 'x marked with x
+n n forward " previous context
% 1,$

Initializing options

EXINIT place set's here in environment var.
$HOME/.exrc editor initialization file
./exrc editor initialization file
set x enable option
set nox disable option
set x=val give value val
set show changed options
set all show all options
set x? show value of option x

Most useful options

autoindent ai supply indent
autowrite aw write before changing files
ignorecase ic in scanning
list print ^I for tab, $ at end
magic . [* special in patterns
number nu number lines
paragraphs para macro names which start ...
redraw simulate smart terminal
scroll command mode lines
sections sect macro names ...
shiftwidth sw for <, >, and input ^D
showmatch sm to ) and ) as typed
showmode smd show insert mode in vi
slowopen slow stop updates during insert
window visual mode lines
wrapscan ws around end of buffer?
wrapmargin wm automatic line splitting

Scanning pattern formation

^ beginning of line
$ end of line
. any character
\< beginning of word
\> end of word
[st] any char in str
[^[st]) ... not in str
[x-y] ... between x and y
* any number of preceding

AUTHOR

Vi and ex are based on software developed by The University of California, Berkeley
California, Computer Science Division, Department of Electrical Engineering and

162

Stardent 1500/3000
Computer Science.

FILES

/etc/lib/ex?strings error messages
/usr/lib/ex?recover recover command
/usr/lib/ex?preserve preserve command
/usr/lib/*/* describes capabilities of terminals
$HOME/.exrc editor startup file
./.exrc editor startup file
/tmp/Exnmnnn editor temporary
/tmp/Rxnmnnn named buffer temporary
/usr/preserve/login preservation directory
(where login is the user's login)

SEE ALSO

awk(1), ed(1), edit(1), grep(1), sed(1), vi(1).
The Terminal Information Utilities Guide.

WARNING

The -x option is provided with the Security Administration Utilities, which is available only in the United States.

BUGS

The undo command causes all marks to be lost on lines changed and then restored if the marked lines were changed.

Undo never clears the buffer modify condition.

The z command prints a number of logical rather than physical lines. More than a screen full of output may result if long lines are present.

File input/output errors do not print a name if the command line `-f' option is used.

There is no easy way to do a single scan ignoring case.

The editor does not warn if text is placed in named buffers and not used before exiting the editor.

Null characters are discarded in input files and cannot appear in resultant files.
NAME

expand, unexpand – expand tabs to spaces, and vice versa

SYNOPSIS

expand [-tabstop] [-tab1,tab2,...,tabn] [ file ... ]
unexpand [-a] [ file ... ]

DESCRIPTION

Expand processes the named files or the standard input writing the standard output with tabs changed into blanks. Backspace characters are preserved into the output and decrement the column count for tab calculations. Expand is useful for preprocessing character files (before sorting, looking at specific columns, etc.) that contain tabs.

If a single tabstop argument is given, then tabs are set tabstop spaces apart instead of the default 8. If multiple tabstops are given then the tabs are set at those specific columns.

Unexpand puts tabs back into the data from the standard input or the named files and writes the result on the standard output. By default, only leading blanks and tabs are reconverted to maximal strings of tabs. If the -a option is given, then tabs are inserted whenever they would compress the resultant file by replacing two or more characters.
EXPR(1)

NAME

expr – evaluate arguments as an expression

SYNOPSIS

expr arguments

DESCRIPTION

The arguments are taken as an expression. After evaluation, the result is written on
the standard output. Terms of the expression must be separated by blanks. Charac-
ters special to the shell must be escaped. Note that 0 is returned to indicate a zero
value, rather than the null string. Strings containing blanks or other special charac-
ters should be quoted. Integer-valued arguments may be preceded by a unary minus
sign. Internally, integers are treated as 32-bit, 2s complement numbers.

The operators and keywords are listed below. Characters that need to be escaped are
preceded by \. The list is in order of increasing precedence, with equal precedence
operators grouped within {} symbols.

expr \| expr
    returns the first expr if it is neither null nor 0, otherwise returns the second
    expr.
expr \& expr
    returns the first expr if neither expr is null or 0, otherwise returns 0.
expr { =, \>, \>=, \<, \<=, != } expr
    returns the result of an integer comparison if both arguments are integers,
    otherwise returns the result of a lexical comparison.
expr { +, - } expr
    addition or subtraction of integer-valued arguments.
expr { *, /, % } expr
    multiplication, division, or remainder of the integer-valued arguments.
expr : expr
    The matching operator : compares the first argument with the second argu-
ment which must be a regular expression. Regular expression syntax is the
same as that of ed(1), except that all patterns are "anchored" (i.e., begin with
^) and, therefore, ^ is not a special character, in that context. Normally, the
matching operator returns the number of characters matched (0 on failure).
Alternatively, the \(...\) pattern symbols can be used to return a portion of
the first argument.

EXAMPLES

1. a=`expr $a + 1`
   adds 1 to the shell variable a.
2. # 'For $a equal to either "/usr/abc/file" or just "file"
   expr $a : ^.*\/(.*\s)\s* \s* $a
   returns the last segment of a path name (i.e., file). Watch out for / alone as
   an argument: expr will take it as the division operator (see BUGS below).
3. # A better representation of example 2.
   expr \!/\$a : ^.*\/(.*\s)\s* \s* $a
   The addition of the // characters eliminates any ambiguity about the divi-
   sion operator and simplifies the whole expression.
4. expr $VAR : ^.*
returns the number of characters in $VAR.

SEE ALSO
ed(1), sh(1).

DIAGNOSTICS
As a side effect of expression evaluation, expr returns the following exit values:

- 0  if the expression is neither null nor 0
- 1  if the expression is null or 0
- 2  for invalid expressions.

- syntax error     for operator/operand errors
- non-numeric argument  if arithmetic is attempted on such a string

BUGS
After argument processing by the shell, expr cannot tell the difference between an
operator and an operand except by the value. If $a is an =, the command:

    expr $a = '=='

looks like:

    expr = = =

as the arguments are passed to expr (and they will all be taken as the = operator).
The following works:

    expr X$a = X=
NAME

factor – obtain the prime factors of a number

SYNOPSIS

factor [ integer ]

DESCRIPTION

When you use factor without an argument, it waits for you to give it an integer. After you give it a positive integer less than or equal to \(10^{14}\), it factors the integer, prints its prime factors the proper number of times, and then waits for another integer. factor exits if it encounters a zero or any non-numeric character.

If you invoke factor with an argument, it factors the integer as described above, and then it exits.

The maximum time to factor an integer is proportional to \(\sqrt{n}\). factor will take this time when \(n\) is prime or the square of a prime.

DIAGNOSTICS

factor prints the error message, "Ouch," for input out of range or for garbage input.
NAME  
fcc – Fortran compiler
SYNOPSIS  
fcc [ options ] [ files ] [ options ] [ files ]
DESCRIPTION  
The fcc command is an interface to the Stardent 1500/3000 Compilation System. The compilation tools consist of a preprocessor, compiler, assembler, and link editor. The fcc command processes the supplied options and then executes the various tools with the proper arguments. The fcc command accepts several types of files as arguments:

Files whose names end with .f are taken to be Fortran source programs and may be preprocessed, compiled, optimized, assembled, and linked edited. The compilation process may be stopped after the completion of any pass if the appropriate options are supplied. If the compilation process runs through the assembler then an object program is produced and is left in the file whose name is that of the source with .o substituted for .f. However, the .o file is normally deleted if a single Fortran program is compiled and then immediately linked edited. In the same way, files whose names end in .s are taken to be assembly source programs, and may be assembled and linked edited; and files whose names end in .i are taken to be preprocessed Fortran source programs and may be compiled, optimized, assembled and linked edited. Files whose names do not end in .f, .s or .i are handed to the link editor.

Since the fcc command usually creates files in the current directory during the compilation process, it is necessary to run the fcc command in a directory in which a file can be created.

The following options are interpreted by fcc:

-all doubles  
Set all undeclared floating point variables, all declared floating point variables, and floating point constants to be as if they were declared REAL*8. If they were COMPLEX*8, they are to COMPLEX*16. Integers are allocated 8 bytes of storage instead of 4 bytes.

-ast=number  
Allow the compiler to set the initial allocation for the program internal representation to the user defined number bytes. Normally, users should not invoke this flag unless the program is terminated with the internal error.

-blanks72  
Pad with blanks on right to column 72.

-c  
Suppress the link editing phase of the compilation, and do not remove any produced object files.

-case_sensitive  
Insist that user defined variables must be case sensitive. The default is -nocase_sensitive.

-catalog=name.in  
Create a database of functions available to be inlined. This option creates two files: the database name.in and an associated file name.id. Each use of catalog adds to the catalog; it does not replace previous versions.

-continuations=n  
Set the number n of continuation lines the compiler accepts for any statement. The default is 19.
-cpp Invokes the C preprocessor on the source file before compiling.

-cross_reference
  Generate a cross-reference, if a listing is generated.

-Dname
  Define name to have the value of 1, to the preprocessor.

-Dname=val
  Define name to have the value of val, to the preprocessor.

-debug
  Add debug data to the object file. Force optimization level to zero. This is
  synonymous with -g.

-double_precision
  Set all undeclared floating point variables and all declared floating point vari-
  ables to be as if they were declared REAL*8. If they were COMPLEX*8, they are
  then set to COMPLEX*16. -nodouble_precision is the default.

-d_lines
  Compile source file lines with a 'D' or 'd' in column 1.

-E
  Run only cpp(1) on the named C programs, and send the result to the standard
  output.

-extend_source
  Extend the statement field of a source line from columns 1 through 72 to
  columns 1 through 132.

-fast Perform some optimization which may lose small amounts of precision.

-full_report
  Produce a detailed vectorizer report.

-fullsubcheck
  Generate code to check that every subscript in every array reference is within
  the bounds of the appropriate array dimensions.

-g Generate additional information needed for the use of dbg(1). Force optimiza-
  tion level to zero. This option is synonymous with -debug.

-I Suppress the default searching for preprocessor included files in /usr/include.

-Ldir Search for preprocessor include files in dir. Neither -I option affects the Fortran
  INCLUDE statement.

-i Suppress the automatic production of #ident information.

-implicit
  Make all variables in a program untyped.

-include_listing
  List included source files as well as the original source file when generating a
  listing.

-inline
  Instruct the compiler to enable function inlining.

-i4 Interpret INTEGER and LOGICAL declarations as if they had been written
  INTEGER*4 and LOGICAL*4.

-list Generate a listing from the compilation.

-messages
  Allow warning messages to be printed.
-Npaths=\text{name.in}
Instruct the compiler to make use of the database of functions listed in the catalog name.in as the source for inlining.

-no_assoc
Assume floating point is not associative. Force the floating point operation to be in the same order as the program specifies (e.g. no vectorized sum or dot product reductions).

-no_directive
Do not apply compiler directives.

-novector
Not generate no vector code for any loop. This option is especially useful at optimization level 03 when you desire parallel code but no vector code.

-O0
Turn off all optimizations.

-O1
Perform common subexpression elimination and instruction scheduling. If nothing is specified, this -O1 is the default setting of compiler optimization level.

-O2
Perform -O1 and vectorization.

-O3
Perform -O2 and parallelization.

-O
This is synonymous with -O1.

-o filename
Place the output into filename.

-object
Generate object files.

-onetrip
Generate code to guarantee that all DO loops execute at least once. This is a compatibility feature for programs originally written for use with Fortran 66. The default is -noonetrip.

-P
Run only \texttt{cpp}(1) on the named Fortran programs and leave the result in corresponding files suffixed .i. This option is passed to \texttt{cpp}(1).

-p
Generate code to profile the loaded program during execution. (See \texttt{prof}(1) and \texttt{mkprof}(1).)

-ploop
Generate code which allows loops within a single routine to be profiled separately.

-r
Produce a relocatable output file.

-S
Compile and do not assemble the named Fortran programs, and leave the assembler output in corresponding files suffixed .s.

-safe_strings
Generate code that corrects for mismatched string parameters. If the type of parameters being passed are truly matched, avoid this option because it causes some performance degradation.

-safe=procs
Cause the compiler to compile a procedure for parallel execution.

-save
All variables declared are saved.
-standard
  Check for standard Fortran 77 usage.

-subcheck
  Produce code to check at runtime to ensure that each array element accessed is
  actually part of the appropriate array. However, at optimization level 02 and
  higher, this option ignores the vector mask. This means that some operations
  may generate subscriptranges that are not actually in the code.

-Unname
  Undefine name.

-V
  Prints a single line of version information with the release number of the com-
  pilation system. Version numbers corresponding to the release number of each
  executing component are also printed. Version numbers vary independently of
  each other and of the release number.

-verbose
  Use verbose message output.

-vreport
  Invoke the vector reporting facility and tell the user what vectorization has
  been done. A detailed listing is provided for each loop nest and suggestions for
  achieving better performance are included.

-vsummary
  Invoke the vector reporting facility and tell the user what vectorization has
  been done. Print out what statements are and are not vectorized in each loop.
  This output is in Fortran-like notation.

-w
  Suppress warning messages during compilation.

-yname
  Print uses and definitions of name.

-43
  Cause the compiler to use /usr/lib/bsd/libc.a instead of /usr/lib/libc.a. In another
  word, this option cause the compiler to use BSD library files instead of System
  V library files.

The fc command recognizes -B hhhhhhh, -D hhhhhhh, -esym, -L, -Ldir, -lflag, -m, -n,
-ofilename, -opct, -p, -r, -s, -T hhhhhhh, -t, -uname, -V, and -yname and passes these
options and their arguments directly to the loader. See the manual pages for cpp(1)
and ld(1) for descriptions.

Other arguments are taken to be Fortran compatible object programs, typically pro-
duced by an earlier fc run, or perhaps libraries of Fortran compatible routines and are
passed directly to the link editor. These programs, together with the results of any
compilations specified, are link edited (in the order given) to produce an executable
program with name a.out.

FILES

file.f     Fortran source file
file.i     preprocessed Fortran source file
file.o     object file
file.s     assembly language file
a.out      link edited output
LIBDIR/fct0.o  start-up routine
LIBDIR/file.i inlined functions file
LIBDIR/file.V vector reporting facility file
LIBDIR/file.L listing file
TMPDIR/* temporary files
LIBDIR/cpp preprocessor, cpp(1)
BINDIR/as assembler, as(1)
BINDIR/lPd link editor, ld(1)
LIBDIR/libc.a standard C library

LIBDIR is usually /lib
BINDIR is usually /bin

TMPDIR is usually /tmp but can be redefined by setting the environment variable TMPDIR [see tempnam() in tmpnam(3S)], and exporting it to the environment in which the programs run.

ENVIRONMENT VARIABLES

Setting a couple of shell environment variables to either VMS or BSD allows you to decide how files should be read and written. These variables are:

    UNFORMATTED_IO
    UNFORMATTED_INPUT
    UNFORMATTED_OUTPUT

The setting of UNFORMATTED_IO takes precedence over the other two variables. If it is set, any setting of UNFORMATTED_INPUT or UNFORMATTED_OUTPUT is ignored. The available settings are described below:

UNFORMATTED_INPUT BSD
    On input, the record length is measured in bytes; there is no padding.

UNFORMATTED_INPUT VMS
    Default.
    On input, the length is in words; padding rounds up to even 4-byte boundaries.

UNFORMATTED_OUTPUT BSD
    On output, the record length is written in bytes; there is no padding.

UNFORMATTED_OUTPUT VMS
    Default.
    On output, the record length is written in words (length rounded up to nearest 4-byte boundary) and padding is up to the nearest 4-byte boundary.

UNFORMATTED_IO BSD
    On input and output, the record length is both read and written in bytes; there is no padding.

UNFORMATTED_IO VMS
    Default.
    On input and output, the record length is read and written in words (rounded up to the nearest 4-byte boundary); padding is up to the nearest 4-byte boundary.

SEE ALSO

as(1), ld(1), cpp(1), prof(1), dbg(1)
Programmer’s Guide
Fortran Reference Manual
NAME

fgrep – search a file for a character string

SYNOPSIS

fgrep [options] string [file ...]

DESCRIPTION

fgrep (fast grep) searches files for a character string and prints all lines that contain
that string. fgrep is different from grep(1) and egrep(1) because it searches for a string,
instead of searching for a pattern that matches an expression. It uses a fast and com-
 pact algorithm.

The characters $, *, [^, |, ), and \ are interpreted literally by fgrep, that is, fgrep
does not recognize full regular expressions as does egrep. Since these characters have
special meaning to the shell, it is safest to enclose the entire string in single quotes
'...'.

If no files are specified, fgrep assumes standard input. Normally, each line found is
copied to the standard output. The file name is printed before each line found if
there is more than one input file.

Command line options are:

-b   Precede each line by the block number on which it was found. This can be
     useful in locating block numbers by context (first block is 0).
-c   Print only a count of the lines that contain the pattern.
-i   Ignore upper/lower case distinction during comparisons.
-l   Print the names of files with matching lines once, separated by new-lines.
     Does not repeat the names of files when the pattern is found more than once.
-n   Precede each line by its line number in the file (first line is 1).
-v   Print all lines except those that contain the pattern.
-x   Print only lines matched entirely.
-e special_string
     Search for a special string (string begins with a -).
-f file  Take the list of strings from file.

SEE ALSO

ed(1), egrep(1), grep(1), sed(1), sh(1).

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible
files (even if matches were found).

BUGS

Ideally there should be only one grep command, but there is not a single algorithm
that spans a wide enough range of space-time tradeoffs. Lines are limited to BUFSIZ
characters; longer lines are truncated. BUFSIZ is defined in /usr/include/stdio.h.
NAME
file – determine file type

SYNOPSIS
file [ -c ] [ -f ffile ] [ -m mfile ] arg ...

DESCRIPTION
file performs a series of tests on each argument in an attempt to classify it. If an argument appears to be ASCII, file examines the first 512 bytes and tries to guess its language. If an argument is an executable a.out, file will print the version stamp, provided it is greater than 0.

-c The -c option causes file to check the magic file for format errors. This validation is not normally carried out for reasons of efficiency. No file typing is done under -c.

-f If the -f option is given, the next argument is taken to be a file containing the names of the files to be examined.

-m The -m option instructs file to use an alternate magic file.

file uses the file /etc/magic to identify files that have some sort of magic number, that is, any file containing a numeric or string constant that indicates its type. Commentary at the beginning of /etc/magic explains its format.

FILES
/etc/magic

SEE ALSO
filehdr(4).
NAME

find – find files

SYNOPSIS

find path-name-list expression

DESCRIPTION

find recursively descends the directory hierarchy for each path name in the path-name-list (that is, one or more path names) seeking files that match a boolean expression written in the primaries given below. In the descriptions, the argument n is used as a decimal integer where +n means more than n, −n means less than n and n means exactly n. Valid expressions are:

−name file True if file matches the current file name. Normal shell argument syntax may be used if escaped (watch out for {, ?, and *).

[−perm]−onum True if the file permission flags exactly match the octal number onum (see chmod(1)). If onum is prefixed by a minus sign, only the bits that are set in onum are compared with the file permission flags, and the expression evaluates true if they match.

−type c True if the type of the file is c, where c is b, c, d, l, p, or f for block special file, character special file, directory, symbolic link, fifo (a.k.a named pipe), or plain file respectively.

−links n True if the file has n links.

−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 taken 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 taken as a group ID.

−size n[c] True if the file is n blocks long (512 bytes per block). If n is followed by a c, the size is in characters.

−atime n True if the file has been accessed in n days. The access time of directories in path-name-list is changed by find itself.

−mtime n True if the file has been modified in n days.

−ctime n True if the file has been changed in n days.

−exec cmd True if the executed cmd returns a zero value as exit status. The end of cmd must be punctuated by an escaped semicolon. A command argument () is replaced by the current path name.

−ok cmd Like −exec except that the generated command line is printed with a question mark first, and is executed only if the user responds by typing y.

−print Always true; causes the current path name to be printed.

−cpio device Always true; write the current file on device in cpio (1) format (5120-byte records).

−newer file True if the current file has been modified more recently than the argument file.

−depth Always true; causes descent of the directory hierarchy to be done so that all entries in a directory are acted on before the directory itself. This can be useful when find is used with cpio(1) to transfer files that are contained in directories without write permission.
-mount  Always true; restricts the search to the file system containing the directory specified, or if no directory was specified, the current directory.

-local  True if the file physically resides on the local system.

(expression)  True if the parenthesized expression is true (parentheses are special to the shell and must be escaped).

The primaries may be combined using the following operators (in order of decreasing precedence):

1)  The negation of a primary (! is the unary not operator).

2)  Concatenation of primaries (the and operation is implied by the juxtaposition of two primaries).

3)  Alternation of primaries (~o is the or operator).

EXAMPLE

To remove all files named a.out or *.o that have not been accessed for a week:

find / \(-name a.out -o -name ‘*.o’ \) -atime +7 -exec rm \;

FILES

/etc/passwd, /etc/group

SEE ALSO

chmod(1), cpio(1), sh(1), test(1), stat(2), umask(2), fs(4).

BUGS

find / -depth always fails with the message:

"find: stat failed: : No such file or directory".

find does not follow symbolic links to directories.
NAME

finger – user information lookup program

SYNOPSIS

finger [ options ] name ...

DESCRIPTION

By default finger lists the login name, full name, terminal name and write status (as a ‘w’ before the terminal name if write permission is denied), idle time, login time, and office location and phone number (if they are known) for each current UNIX user. (Idle time is minutes if it is a single integer, hours and minutes if a ‘:’ is present, or days and hours if a ‘d’ is present.)

A longer format also exists and is used by finger whenever a list of people’s names is given. (Account names as well as first and last names of users are accepted.) This format is multi-line, and includes all the information described above as well as the user’s home directory and login shell, any plan which the person has placed in the file .plan in their home directory, and the project on which they are working from the file .project also in the home directory.

Finger may be used to lookup users on a remote machine. The format is to specify the user as “user@host.” If the user name is left off, the standard format listing is provided on the remote machine.

Finger options include:

-m Match arguments only on user name.
-l Force long output format.
-p Suppress printing of the .plan files
-s Force short output format.

FILES

/etc/utmp who file
/etc/passwd for users’ names, offices, ...
/usr/adm/lastlog last login times
~/.plan plans
~/.project projects

SEE ALSO

chfn(1), w(1), who(1)

AUTHOR

Earl T. Cohen

BUGS

Only the first line of the .project file is printed.

There is no way to pass arguments to the remote machine as finger uses an internet standard port.
NAME
fold – fold long lines for finite width output device

SYNOPSIS
fold [ -width ] [ file ... ]

DESCRIPTION
Fold is a filter which will fold the contents of the specified files, or the standard input if no files are specified, breaking the lines to have maximum width width. The default for width is 80. Width should be a multiple of 8 if tabs are present, or the tabs should be expanded using expand(1) before coming to fold.

SEE ALSO
expand(1)

BUGS
If underlining is present it may be messed up by folding.
NAME

fpr, asa – print Fortran file

SYNOPSIS

fpr
asa

DESCRIPTION

fpr (asa) is a filter that transforms files formatted according to Fortran’s carriage control conventions into files formatted according to UNIX line printer conventions. fpr (asa) copies its input onto its output, replacing the carriage control characters with characters that produce the intended effects when printed. The first character of each line determines the vertical spacing as follows:

<table>
<thead>
<tr>
<th>Character</th>
<th>Vertical Space Before Printing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>One line</td>
</tr>
<tr>
<td>0</td>
<td>Two lines</td>
</tr>
<tr>
<td>1</td>
<td>To first line of next page</td>
</tr>
<tr>
<td>+</td>
<td>No advance</td>
</tr>
</tbody>
</table>

A blank line is treated as if its first character is a blank. A blank that appears as a carriage control character is deleted. A zero is changed to a newline. A one is changed to a form feed. The effects of a "+" are simulated using backspaces.

EXAMPLES

a.out | fpr | lp
a.out | asa | lp
fpr < fc.output | lp

BUGS

Results are undefined for input lines longer than 170 characters.
NAME

from – who is my mail from?

SYNOPSIS

from [ -s sender ] [ user ]

DESCRIPTION

from prints out the mail header lines in your mailbox file to show you who your mail is from. If user is specified, then user's mailbox is examined instead of your own. If the -s option is given, then only headers for mail sent by sender are printed.

FILES

/usr/spool/mail/*

SEE ALSO

biff(1), mail(1)
NAME

fsck – check and repair file systems

SYNOPSIS


DESCRIPTION

fsck audits and interactively repairs inconsistent conditions for file systems. If the file system is found to be consistent, the number of files, blocks used, and blocks free are reported. If the file system is inconsistent the user is prompted for concurrence before each correction is attempted. It should be noted that some corrective actions result in loss of some data. The amount and severity of data loss may be determined from the diagnostic output. The default action for each correction is to wait for the user to respond yes or no. If the user does not have write permission fsck defaults to a -n action.

The following options are accepted by fsck.

- y  Assume a yes response to all questions asked by fsck.
- n  Assume a no response to all questions asked by fsck; do not open the file system for writing.
- sX Ignore the actual free list and (unconditionally) reconstruct a new one by rewriting the super-block of the file system. The file system should be unmounted while this is done; if this is not possible, care should be taken that the system is quiescent and that it is rebooted immediately afterwards. This precaution is necessary so that the old, bad, in-core copy of the superblock will not continue to be used, or written on the file system.

The -sX option allows for creating an optimal free-list organization.

If X is not given, the values used when the file system was created are used.

The format of X is cylinder size:gap size.

NOTE: This option has no effect on the Fast File System.

- SX Conditionally reconstruct the free list. This option is like -sX above except that the free list is rebuilt only if there were no discrepancies discovered in the file system. Using -S will force a no response to all questions asked by fsck. This option is useful for forcing free list reorganization on uncontaminated file systems.

NOTE: This option has no effect on the Fast File System.

- t  If fsck cannot obtain enough memory to keep its tables, it uses a scratch file. If the -t option is specified, the file named in the next argument is used as the scratch file, if needed. Without the -t flag, fsck will prompt the user for the name of the scratch file. The file chosen should not be on the file system being checked, and if it is not a special file or did not already exist, it is removed when fsck completes.

- q  Quiet fsck. Do not print size-check messages. Unreferenced fifos will silently be removed. If fsck requires it, counts in the superblock will be automatically fixed and the free list salvaged.

- D  Directories are checked for bad blocks. Useful after system crashes.

- f  Fast check. Check block and sizes and check the free list. The free list will be reconstructed if it is necessary.

- b  Reboot. If the file system being checked is the root file system and modifications have been made, then either remount the root file system or reboot the system. A remount is done only if there was minor damage.
If no `file-systems` are specified, `fsck` will read a list of default file systems from the file 
`/etc/checklist`.

Inconsistencies checked are as follows:
1. Blocks claimed by more than one i-node or the free list.
2. Blocks claimed by an i-node or the free list outside the range of the file system.
3. Incorrect link counts.
4. Size checks:
   - Incorrect number of blocks.
   - Directory size not 16-byte aligned.
5. Bad i-node format.
6. Blocks not accounted for anywhere.
7. Directory checks:
   - File pointing to unallocated i-node.
   - I-node number out of range.
8. Super Block checks:
   - More than 65536 i-nodes.
   - More blocks for i-nodes than there are in the file system.
9. Bad free block list format. (System V file system only.)
10. Total free block and/or free i-node count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with the user's concurrence, reconnected by placing them in the `lost+found` directory, if the files are nonempty. The user will be notified if the file or directory is empty or not. Empty files or directories are removed, as long as the `-n` option is not specified. `fsck` will force the reconnection of nonempty directories. The name assigned is the i-node number. The only restriction is that the directory `lost+found` must preexist in the root of the file system being checked and must have empty slots in which entries can be made. This is accomplished by making `lost+found`, copying a number of files to the directory, and then removing them (before `fsck` is executed).

Checking the raw device is almost always faster and should be used with everything but the `root` file system.

**WARNING** Do not use the raw device on the `root` file system or serious damage to the file system will result.

**FILES**

`/etc/checklist` contains default list of file systems to check.

**SEE ALSO**

`checklist(4), crash(1M), fs(4), mkfs(1M), ncheck(1M), uadmin(2)`.

**BUGS**

I-node numbers for . and .. in each directory are not checked for validity.
NAME

fsdb – file system debugger

SYNOPSIS

/etc/fsdb special [-]

DESCRIPTION

fsdb can be used to patch up a damaged file system after a crash. It has conversions to translate block and i-numbers into their corresponding disk addresses. Also included are mnemonic offsets to access different parts of an i-node. These greatly simplify the process of correcting control block entries or descending the file system tree.

fsdb automatically determines the type of file system you are trying to repair and invokes the appropriate module:

fsdb.s54k
  to repair System V file systems

fsdb.afs
  to repair Ardent Fast File Systems

fsdb contains several error-checking routines to verify i-node and block addresses. These can be disabled if necessary by invoking fsdb with the optional – argument or by the use of the O symbol. (fsdb reads the i-size and f-size entries from the superblock of the file system as the basis for these checks.)

Numbers are considered decimal by default. Octal numbers must be prefixed with a zero. During any assignment operation, numbers are checked for a possible truncation error due to a size mismatch between source and destination.

fsdb reads a block at a time and will therefore work with raw as well as block I/O. A buffer management routine is used to retain commonly used blocks of data in order to reduce the number of read system calls. All assignment operations result in an immediate write-through of the corresponding block.

The symbols recognized by fsdb are:

#    absolute address
i    convert from i-number to i-node address
b    convert to block address
d    directory slot offset
e    extent slot offset (only available for the Ardent Fast File System)
+,-  address arithmetic
q    quit
>,<  save, restore an address
=    numerical assignment
+=   incremental assignment
-=   decremental assignment
"="  character string assignment
O    error checking flip flop
P    general print facilities
F    file print facility
B    byte mode
W    word mode
D    double word mode
!    escape to shell

The print facilities generate a formatted output in various styles. The current address is normalized to an appropriate boundary before printing begins. It advances with the printing and is left at the address of the last item printed. The output can be
terminated at any time by typing the delete character. If a number follows the p symbol, that many entries are printed. A check is made to detect block boundary overflows since logically sequential blocks are generally not physically sequential. If a count of zero is used, all entries to the end of the current block are printed. The print options available are:

- i: print as i-nodes
- d: print as directories
- o: print as octal words
- e: print as decimal words
- c: print as characters
- b: print as octal bytes
- x: print as hexadecimal bytes

The f symbol is used to print data blocks associated with the current i-node. If followed by a number, that block of the file is printed. (Blocks are numbered from zero.) The desired print option letter follows the block number, if present, or the f symbol. This print facility works for small as well as large files. It checks for special devices and that the block pointers used to find the data are not zero.

Dots, tabs, and spaces may be used as function delimiters but are not necessary. A line with just a new-line character will increment the current address by the size of the data type last printed. That is, the address is set to the next byte, word, double word, directory entry or i-node, allowing the user to step through a region of a file system. Information is printed in a format appropriate to the data type. Bytes, words and double words are displayed with the octal address followed by the value in octal and decimal. A .B or .D is appended to the address for byte and double word values, respectively. Directories are printed as a directory slot offset followed by the decimal i-number and the character representation of the entry name. I-nodes are printed with labeled fields describing each element.

The following mnemonics are used for i-node examination and refer to the current working i-node:

- md: mode
- ln: link count
- uid: user ID number
- gid: group ID number
- sz: file size
- a#: data block numbers (0 – 12)
- at: access time
- mt: modification time
- maj: major device number
- min: minor device number

**Examples**

386i prints i-number 386 in an i-node format. This now becomes the current working i-node.

ln=4 changes the link count for the working i-node to 4.

ln=+1 increments the link count by 1.

fc prints, in ASCII, block zero of the file associated with the working i-node.

2i.fd prints the first 32 directory entries for the root i-node of this file system.
d5i.fc changes the current i-node to that associated with the 5th directory entry (numbered from zero) found from the above command. The first logical block of the file is then printed in ASCII.

512B:p0o prints the superblock of this file system in octal.

2i.a0b.d7=3 changes the i-number for the seventh directory slot in the root directory to 3. This example also shows how several operations can be combined on one command line.

d7.nm="name" changes the name field in the directory slot to the given string. Quotes are optional when used with nm if the first character is alphabetic.

a2b.p0d prints the third block of the current i-node as directory entries.

e0b.p0x prints the first block of the current extent (for an Ardent Fast File System) in hexadecimal format

SEE ALSO
dir(4), fs(4), fsck(1M)
NAME
fsplit – split a multi-routine Fortran file into individual files

SYNOPSIS
fsplit [-e efile] ... [ file ]

DESCRIPTION
fsplit takes as input either a file or standard input containing Fortran source code. It attempts to split the input into separate routine files of the form name.f, where name is the name of the program unit (e.g., function, subroutine, block data or program). The name for unnamed block data subprograms has the form blkdataNNN.f where NNN is three digits and a file of this name does not already exist. For unnamed main programs the name has the form mainNNN.f. If there is an error in classifying a program unit, or if name.f already exists, the program unit will be put in a file of the form zzzNNN.f where zzzNNN.f does not already exist.

Normally each subprogram unit is split into a separate file. When the -e option is used, only the specified subprogram units are split into separate files. E.g.:

    fsplit -e readit -e doit prog.f

will split readit and doit subprograms into separate files.

DIAGNOSTICS
If names specified via the -e option are not found, a diagnostic is written to standard error.

AUTHOR
Asa Romberger and Jerry Berkman

BUGS
Fsplit assumes the subprogram name is on the first noncomment line of the subprogram unit. Nonstandard source formats may confuse fsplit.

It is hard to use -e for unnamed main programs and block data subprograms since you must predict the created file name.
NAME

fsstat – report file system status

SYNOPSIS

/etc/fsstat special_file

DESCRIPTION

fsstat reports on the status of the file system on special_file. During startup, this command is used to determine if the file system needs checking before it is mounted. fsstat succeeds if the file system is unmounted and appears okay. For the root file system, it succeeds if the file system is active and not marked bad.

SEE ALSO

fs(4).

DIAGNOSTICS

The command has the following exit codes:

0 – the file system is not mounted and appears okay,
   (except for root where 0 means mounted and okay).
1 – the file system is not mounted and needs to be checked.
2 – the file system is mounted.
3 – the command failed.

BUGS

Does not work with the Fast File System.
fstyp – determine file system identifier

**SYNOPSIS**

`fstyp special`

**DESCRIPTION**

`fstyp` allows the user to determine the file system identifier of mounted or unmounted file systems using heuristic programs. The file system type is required by `mount(2)` and sometimes by `mount(1M)` to mount file systems of different types.

The directory `/etc/fstyp.d` contains a program for each file system type to be checked; each of these programs applies some appropriate heuristic to determine whether the supplied `special` file is of the type for which it checks. If it is, the program prints on standard output the usual file-system identifier for that type and exits with a return code of 0; otherwise it prints error messages on standard error and exits with a non-zero return code. `fstyp` runs the programs in `/etc/fstyp.d` in alphabetical order, passing `special` as an argument; if any program succeeds, its file-system type identifier is printed and `fstyp` exits immediately. If no program succeeds, `fstyp` prints "Unknown_fstyp" to indicate failure.

**WARNING**

The use of heuristics implies that the result of `fstyp` is not guaranteed to be accurate.

**SEE ALSO**

`mount(1M)`, `mount(2)` , `sysfs(2)`.

**BUGS**

Does not work with the Fast File System.
NAME

ftp - ARPANET file transfer program

SYNOPSIS

ftp [-v] [-d] [-i] [-n] [-g] [host]

DESCRIPTION

ftp is the user interface to the ARPANET standard File Transfer Protocol. The pro-
gram allows a user to transfer files to and from a remote network site.

The client host with which ftp is to communicate may be specified on the command
line. If this is done, ftp will immediately attempt to establish a connection to an FTP
server on that host; otherwise, ftp will enter its command interpreter and await
instructions from the user. When ftp is awaiting commands from the user the
prompt "ftp>" is provided to the user. The following commands are recognized by
ftp:

! [ command [ args ] ]

Invoke an interactive shell on the local machine. If there are arguments, the
first is taken to be a command to execute directly, with the rest of the argu-
ments as its arguments.

$ macro-name [ args ]

Execute the macro macro-name that was defined with the macdef command.
Arguments are passed to the macro unglobbed.

account [ passwd ]

Supply a supplemental password required by a remote system for access to
resources once a login has been successfully completed. If no argument is
included, the user will be prompted for an account password in a non-echoing
input mode.

append local-file [ remote-file ]

Append a local file to a file on the remote machine. If remote-file is left
unspecified, the local file name is used in naming the remote file after being
altered by any ntrans or nmap setting. File transfer uses the current settings for
type, format, mode, and structure.

ascii

Set the file transfer type to network ASCII. This is the default type.

bell

Arrange that a bell be sounded after each file transfer command is completed.

binary

Set the file transfer type to support binary image transfer.

bye

Terminate the FTP session with the remote server and exit ftp. An end of file
will also terminate the session and exit.

case

Toggle remote computer file name case mapping during mget commands.
When case is on (default is off), remote computer file names with all letters in
upper case are written in the local directory with the letters mapped to lower
case.

cd remote-directory

Change the working directory on the remote machine to remote-directory.

cdup

Change the remote machine working directory to the parent of the current
remote machine working directory.

close

Terminate the FTP session with the remote server, and return to the command
interpreter. Any defined macros are erased.
cr  Toggle carriage return stripping during ascii type file retrieval. Records are
denoted by a carriage return/linefeed sequence during ascii type file transfer.
When cr is on (the default), carriage returns are stripped from this sequence to
conform with the UNIX single linefeed record delimiter. Records on non-UNIX
remote systems may contain single linefeeds; when an ascii type transfer is
made, these linefeeds may be distinguished from a record delimiter only when
cr is off.

delete remote-file
Delete the file remote-file on the remote machine.

debug [ debug-value ]
Toggle debugging mode. If an optional debug-value is specified it is used to set
the debugging level. When debugging is on, ftp prints each command sent to
the remote machine, preceded by the string "-->".

dir [ remote-directory ] [ local-file ]
Print a listing of the directory contents in the directory, remote-directory, and,
optionally, placing the output in local-file. If no directory is specified, the
current working directory on the remote machine is used. If no local file is
specified, or local-file is -, output comes to the terminal.

disconnect
A synonym for close.

form format
Set the file transfer form to format. The default format is "file".

get remote-file [ local-file ]
Retrieve the remote-file and store it on the local machine. If the local file name is
not specified, it is given the same name it has on the remote machine, subject to
alteration by the current case, ntrans, and nmap settings. The current settings
for type, form, mode, and structure are used while transferring the file.

glob  Toggle filename expansion for mdelete, mget and mput. If globbing is turned
off with glob, the file name arguments are taken literally and not expanded.
Globbing for mput is done as in csh(1). For mdelete and mget, each remote file
name is expanded separately on the remote machine and the lists are not
merged. Expansion of a directory name is likely to be different from expansion
of the name of an ordinary file: the exact result depends on the foreign operating
system and ftp server, and can be previewed by doing 'mls remote-files -'.
Note: mget and mput are not meant to transfer entire directory subtrees of
files. That can be done by transferring a tar(1) archive of the subtree (in binary
mode).

hash  Toggle hash-sign ("#") printing for each data block transferred. The size of a
data block is 1024 bytes.

help [ command ]
Print an informative message about the meaning of command. If no argument is
given, ftp prints a list of the known commands.

lcd [ directory ]
Change the working directory on the local machine. If no directory is specified,
the user's home directory is used.

ls [ remote-directory ] [ local-file ]
Print an abbreviated listing of the contents of a directory on the remote
machine. If remote-directory is left unspecified, the current working directory is
used. If no local file is specified, or if local-file is -, the output is sent to the ter-
minal.
**macdef** *macro-name*
Define a macro. Subsequent lines are stored as the macro *macro-name*; a null line (consecutive newline characters in a file or carriage returns from the terminal) terminates macro input mode. There is a limit of 16 macros and 4096 total characters in all defined macros. Macros remain defined until a close command is executed. The macro processor interprets '$' and 'v' as special characters. A '$' followed by a number (or numbers) is replaced by the corresponding argument on the macro invocation command line. A '$' followed by an 'v' signals that macro processor that the executing macro is to be looped. On the first pass '$v' is replaced by the first argument on the macro invocation command line, on the second pass it is replaced by the second argument, and so on. A 'v' followed by any character is replaced by that character. Use the 'v' to prevent special treatment of the '$'.

**mdelete** [ *remote-files* ]
Delete the *remote-files* on the remote machine.

**mdir** *remote-files* *local-file*
Like *dir*, except multiple remote files may be specified. If interactive prompting is on, *ftp* will prompt the user to verify that the last argument is indeed the target local file for receiving *mdir* output.

**mget** *remote-files*
Expand the *remote-files* on the remote machine and do a *get* for each file name thus produced. See *glob* for details on the filename expansion. Resulting file names will then be processed according to *case*, *ntrans*, and *nmmap* settings. Files are transferred into the local working directory, which can be changed with 'cd directory'; new local directories can be created with 'mkdir directory'.

**mkdir** *directory-name*
Make a directory on the remote machine.

**mls** *remote-files* *local-file*
Like *ls*, except multiple remote files may be specified. If interactive prompting is on, *ftp* will prompt the user to verify that the last argument is indeed the target local file for receiving *mls* output.

**mode** [ *mode-name* ]
Set the file transfer *mode* to *mode-name*. The default mode is "stream" mode.

**mput** *local-files*
Expand wild cards in the list of local files given as arguments and do a *put* for each file in the resulting list. See *glob* for details of filename expansion. Resulting file names will then be processed according to *ntrans* and *nmmap* settings.

**nmap** [ *inpattern* *outpattern* ]
Set or unset the filename mapping mechanism. If no arguments are specified, the filename mapping mechanism is unset. If arguments are specified, remote filenames are mapped during *mput* commands and *put* commands issued without a specified remote target filename. If arguments are specified, local filenames are mapped during *mget* commands and *get* commands issued without a specified local target filename. This command is useful when connecting to a non-UNIX remote computer with different file naming conventions or practices. The mapping follows the pattern set by *inpattern* and *outpattern*. *Inpattern* is a template for incoming filenames (which may have already been processed according to the *ntrans* and *case* settings). Variable templating is accomplished by including the sequences '$1', '$2', ..., '$9' in *inpattern*. Use 'v' to prevent this special treatment of the '$' character. All other characters are treated literally, and are used to determine the *nmap* *inpattern* variable values.
For example, given \texttt{inpatter} $1.\$2$ and the remote file name "mydata.data", $1$ would have the value "mydata", and $2$ would have the value "data". The \texttt{out-pattern} determines the resulting mapped filename. The sequences '$_1$', '$_2$', ..., '$_9$' are replaced by any value resulting from the \texttt{inpatter} template. The sequence '$_0$' is replaced by the original filename. Additionally, the sequence '{seq1,seq2}' is replaced by \texttt{seq1} if seq1 is not a null string; otherwise it is replaced by seq2. For example, the command "\texttt{nmap}$1.\$2.\$3$ [\$1,\$2],[\$2,\$3]$" would yield the output filename "myfile.data" for input filenames "myfile.data" and "myfile.data.old", "myfile.file" for the input filename "myfile", and "myfile.myfile" for the input filename "myfile". Spaces may be included in \texttt{out-pattern}, as in the example: \texttt{nmap}$1$ !sed "s/ +$//" > $1$. Use the '\' character to prevent special treatment of the '$', '[', ']', and ',' characters.

\texttt{ntans} [ \texttt{inchar} [ \texttt{outchar} ] ]

Set or unset the filename character translation mechanism. If no arguments are specified, the filename character translation mechanism is unset. If arguments are specified, characters in remote filenames are translated during mput commands and put commands issued without a specified remote target filename. If arguments are specified, characters in local filenames are translated during mget commands and get commands issued without a specified local target filename. This command is useful when connecting to a non-UNIX remote computer with different file naming conventions or practices. Characters in a filename matching a character in \texttt{inchar} are replaced with the corresponding character in \texttt{outchar}. If the character's position in \texttt{inchar} is longer than the length of \texttt{outchar}, the character is deleted from the file name.

\texttt{open} host [ port ]

Establish a connection to the specified \texttt{host} FTP server. An optional port number may be supplied, in which case, ftp will attempt to contact an FTP server at that port. If the auto-login option is on (default), ftp will also attempt to automatically log the user in to the FTP server (see below).

\texttt{prompt}

Toggle interactive prompting. Interactive prompting occurs during multiple file transfers to allow the user to selectively retrieve or store files. If prompting is turned off (default is on), any mget or mput will transfer all files, and any mdelete will delete all files.

\texttt{proxy} ftp-command

Execute an ftp command on a secondary control connection. This command allows simultaneous connection to two remote ftp servers for transferring files between the two servers. The first proxy command should be an open, to establish the secondary control connection. Enter the command "proxy ?" to see other ftp commands executable on the secondary connection. The following commands behave differently when prefixed by proxy: open will not define new macros during the auto-login process, close will not erase existing macro definitions, get and mget transfer files from the host on the primary control connection to the host on the secondary control connection, and put, mput, and append transfer files from the host on the secondary control connection to the host on the primary control connection. Third party file transfers depend upon support of the ftp protocol PASV command by the server on the secondary control connection.

\texttt{put} local-file [ remote-file ]

Store a local file on the remote machine. If remote-file is left unspecified, the local file name is used after processing according to any \texttt{ntans} or \texttt{nmap} settings in naming the remote file. File transfer uses the current settings for type, format,
mode, and structure.

**pwd** Print the name of the current working directory on the remote machine.

**quit** A synonym for bye.

**quote** *arg1 arg2* ...
The arguments specified are sent, verbatim, to the remote FTP server.

**recv** remote-file [local-file]
A synonym for get.

**remotelnlp** [command-name]
Request help from the remote FTP server. If a command-name is specified it is supplied to the server as well.

**rename** [from] [to]
Rename the file from on the remote machine, to the file to.

**reset** Clear reply queue. This command re-synchronizes command/reply sequencing with the remote ftp server. Resynchronization may be necessary following a violation of the ftp protocol by the remote server.

**rmdir** directory-name
Delete a directory on the remote machine.

**runique**
Toggle storing of files on the local system with unique filenames. If a file already exists with a name equal to the target local filename for a get or mget command, a "1" is appended to the name. If the resulting name matches another existing file, a "2" is appended to the original name. If this process continues up to "99", an error message is printed, and the transfer does not take place. The generated unique filename will be reported. Note that runique will not affect local files generated from a shell command (see below). The default value is off.

**send** local-file [remote-file]
A synonym for put.

**sendport**
Toggle the use of PORT commands. By default, ftp will attempt to use a PORT command when establishing a connection for each data transfer. The use of PORT commands can prevent delays when performing multiple file transfers. If the PORT command fails, ftp will use the default data port. When the use of PORT commands is disabled, no attempt will be made to use PORT commands for each data transfer. This is useful for certain FTP implementations which do ignore PORT commands but, incorrectly, indicate they’ve been accepted.

**status**
Show the current status of ftp.

**struct** [struct-name]
Set the file transfer structure to struct-name. By default "stream" structure is used.

**sunique**
Toggle storing of files on remote machine under unique file names. Remote ftp server must support ftp protocol STOU command for successful completion. The remote server will report unique name. Default value is off.

**tenex**
Set the file transfer type to that needed to talk to TENEX machines.
trace  Toggle packet tracing.

**type [ type-name ]**
Set the file transfer type to type-name. If no type is specified, the current type is printed. The default type is network ASCII.

**user [ user-name [ password ] [ account ]**
Identify yourself to the remote FTP server. If the password is not specified and the server requires it, *ftp* will prompt the user for it (after disabling local echo). If an account field is not specified, and the FTP server requires it, the user will be prompted for it. If an account field is specified, an account command will be relayed to the remote server after the login sequence is completed if the remote server did not require it for logging in. Unless *ftp* is invoked with “auto-login” disabled, this process is done automatically on initial connection to the FTP server.

**verbose**
Toggle verbose mode. In verbose mode, all responses from the FTP server are displayed to the user. In addition, if verbose is on, when a file transfer completes, statistics regarding the efficiency of the transfer are reported. By default, verbose is on.

? [ command ]
A synonym for help.

Command arguments which have embedded spaces may be quoted with quote (") marks.

**ABORTING A FILE TRANSFER**
To abort a file transfer, use the terminal interrupt key (usually Ctrl-C). Sending transfers will be immediately halted. Receiving transfers will be halted by sending an *ftp* protocol ABOR command to the remote server, and discarding any further data received. The speed at which this is accomplished depends upon the remote server’s support for ABOR processing. If the remote server does not support the ABOR command, an "ftp>" prompt will not appear until the remote server has completed sending the requested file.

The terminal interrupt key sequence will be ignored when *ftp* has completed any local processing and is awaiting a reply from the remote server. A long delay in this mode may result from the ABOR processing described above, or from unexpected behavior by the remote server, including violations of the *ftp* protocol. If the delay results from unexpected remote server behavior, the local *ftp* program must be killed by hand.

**FILE NAMING CONVENTIONS**
Files specified as arguments to *ftp* commands are processed according to the following rules.

1) If the file name "-" is specified, the stdin (for reading) or stdout (for writing) is used.

2) If the first character of the file name is "!", the remainder of the argument is interpreted as a shell command. *Ftp* then forks a shell, using *popen*(3) with the argument supplied, and reads (writes) from the stdout (stdin). If the shell command includes spaces, the argument must be quoted; e.g. ""! Is -It"". A particularly useful example of this mechanism is: "dir | more".

3) Failing the above checks, if "globbing" is enabled, local file names are expanded according to the rules used in the *csh*(1); c.f. the *glob* command. If the *ftp* command expects a single local file (.e.g. put), only the first filename generated by the "globbing" operation is used.
4) For **mget** commands and **get** commands with unspecified local file names, the local filename is the remote filename, which may be altered by a **case**, **ntrans**, or **nmap** setting. The resulting filename may then be altered if **runique** is on.

5) For **mput** commands and **put** commands with unspecified remote file names, the remote filename is the local filename, which may be altered by a **ntrans** or **nmap** setting. The resulting filename may then be altered by the remote server if **runique** is on.

**FILE TRANSFER PARAMETERS**

The FTP specification specifies many parameters which may affect a file transfer. The **type** may be one of “ascii”, “image” (binary), “ebcdic”, and “local byte size” (for PDP-10’s and PDP-20’s mostly). **Ftp** supports the ascii and image types of file transfer, plus local byte size 8 for **tenex** mode transfers.

**Ftp** supports only the default values for the remaining file transfer parameters: **mode**, **form**, and **struct**.

**OPTIONS**

Options may be specified at the command line, or to the command interpreter.

The **-v** (verbose on) option forces **ftp** to show all responses from the remote server, as well as report on data transfer statistics.

The **-n** option restrains **ftp** from attempting “auto-login” upon initial connection. If auto-login is enabled, **ftp** will check the **.netrc** (see below) file in the user’s home directory for an entry describing an account on the remote machine. If no entry exists, **ftp** will prompt for the remote machine login name (default is the user identity on the local machine), and, if necessary, prompt for a password and an account with which to login.

The **-i** option turns off interactive prompting during multiple file transfers.

The **-d** option enables debugging.

The **-g** option disables file name globbing.

**THE .netrc FILE**

The **.netrc** file contains login and initialization information used by the auto-login process. It resides in the user’s home directory. The following tokens are recognized; they may be separated by spaces, tabs, or new-lines:

- machine **name**
  Identify a remote machine name. The auto-login process searches the **.netrc** file for a **machine** token that matches the remote machine specified on the **ftp** command line or as an **open** command argument. Once a match is made, the subsequent **.netrc** tokens are processed, stopping when the end of file is reached or another **machine** token is encountered.

- login **name**
  Identify a user on the remote machine. If this token is present, the auto-login process will initiate a login using the specified name.

- password **string**
  Supply a password. If this token is present, the auto-login process will supply the specified string if the remote server requires a password as part of the login process. Note that if this token is present in the **.netrc** file, **ftp** will abort the auto-login process if the **.netrc** is readable by anyone besides the user.

- account **string**
  Supply an additional account password. If this token is present, the auto-login process will supply the specified string if the remote server requires an
additional account password, or the auto-login process will initiate an ACCT command if it does not.

**macdef name**

Define a macro. This token functions like the *ftp macdef* command functions. A macro is defined with the specified name; its contents begin with the next .netrc line and continue until a null line (consecutive new-line characters) is encountered. If a macro named *init* is defined, it is automatically executed as the last step in the auto-login process.

**BUGS**

Correct execution of many commands depends upon proper behavior by the remote server.

An error in the treatment of carriage returns in the 4.2BSD UNIX ascii-mode transfer code has been corrected. This correction may result in incorrect transfers of binary files to and from 4.2BSD servers using the ascii type. Avoid this problem by using the binary image type.
NAME

fuser – identify processes using a file or file structure

SYNOPSIS

/etc/fuser [-ku] files | resources [-] [[-ku] files | resources]

DESCRIPTION

fuser outputs the process IDs of the processes that are using the files specified as arguments. Each process ID is followed by a letter code, interpreted as follows: if the process is using the file as 1) its current directory, the code is c, 2) the parent of its current directory (only when the file is being used by the system), the code is p, or 3) its root directory, the code is r. For block special devices with mounted file systems, all processes using any file on that device are listed. For all other types of files (text files, executables, directories, devices, etc.) only the processes using that file are reported.

The following options may be used with fuser:

- `u` the user login name, in parentheses, also follows the process ID.
- `k` the SIGKILL signal is sent to each process. Since this option spawns kills for each process, the kill messages may not show up immediately [see kill(2)].

If more than one group of files are specified, the options may be respecified for each additional group of files. A lone dash cancels the options currently in force; then, the new set of options applies to the next group of files.

The process IDs are printed as a single line on the standard output, separated by spaces and terminated with a single new line. All other output is written on standard error.

Any user with permission to read /dev/kmem and /dev/mem can use fuser. Only the super-user can terminate another user’s process.

FILES

/unicorn for system namelist
/dev/kmem for system image
/dev/mem also for system image

SEE ALSO

kill(2), mount(1M), ps(1), signal(2).
fwtmp, wtmpfix – manipulate connect accounting records

/usr/lib/acct/fwtmp [-ic]
/usr/lib/acct/wtmpfix [files]

fwtmp reads from the standard input and writes to the standard output, converting binary records of the type found in wtmp to formatted ASCII records. The ASCII version is useful to enable editing, via ed(1), bad records or general purpose maintenance of the file.

The argument –ic is used to denote that input is in ASCII form, and output is to be written in binary form.

wtmpfix examines the standard input or named files in wtmp format, corrects the time/date stamps to make the entries consistent, and writes to the standard output. A – can be used in place of files to indicate the standard input. If time/date corrections are not performed, acctcon(1) will fail when it encounters certain date-change records.

Each time the date is set, a pair of date change records are written to /etc/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 of the <utmp.h> structure. The second record specifies the new date and is denoted by the string new time placed in the line field and the flag NEW_TIME placed in the type field. wtmpfix uses these records to synchronize all time stamps in the file.

In addition to correcting time/date stamps, wtmpfix will check the validity of the name field to ensure that it consists solely of alphanumeric characters or spaces. If it encounters a name that is considered invalid, it will change the login name to INVALID and write a diagnostic to the standard error. In this way, wtmpfix reduces the chance that acctcon(1) will fail when processing connect accounting records.

/files
/etc/wtmp
/usr/include/utmp.h

SEE ALSO
acct(1M), acct(2), acct(4), acctcms(1M), acctcom(1), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), ed(1), runacct(1M), utmp(4)
NAME

get – get a version of an SCCS file

SYNOPSIS


DESCRIPTION

get generates an ASCII text file from each named SCCS file according to the specifications given by its keyletter arguments, which begin with –. The arguments may be specified in any order, but all keyletter arguments apply to all named SCCS files. If a directory is named, get behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.) and unreadable files are silently ignored. If a name of – is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The generated text is normally written into a file called the g-file whose name is derived from the SCCS file name by simply removing the leading s; (see also FILES, below).

Each of the keyletter arguments is explained below as though only one SCCS file is to be processed, but the effects of any keyletter argument applies independently to each named file.

-rSID  The SCCS IDentification string (SID) of the version (delta) of an SCCS file to be retrieved. Table-1 below shows, for the most useful cases, what version of an SCCS file is retrieved (as well as the SID of the version to be eventually created by delta(1) if the –e keyletter is also used), as a function of the SID specified.

-cutoff  Cutoff date-time, in the form:

YY[MM][DD][HH][MM][SS]]

No changes (deltas) to the SCCS file which were created after the specified cutoff date-time are included in the generated ASCII text file. Units omitted from the date-time default to their maximum possible values; that is, –c7502 is equivalent to –c750228235959. Any number of non-numeric characters may separate the various 2-digit pieces of the cutoff date-time. This feature allows one to specify a cutoff date in the form: “–c77/2/2 9:22:25”. Note that this implies that one may use the %E% and %U% identification keywords (see below) for nested gets within, say the input to a send(1C) command:

~!get "–c%E% %U%" s.file

-ilist  A list of deltas to be included (forced to be applied) in the creation of the generated file. The list has the following syntax:

<list> ::= <range> | <list>, <range>

<range> ::= SID | SID – SID

SID, the SCCS Identification of a delta, may be in any form shown in the “SID Specified” column of Table 1.

-xlist  A list of deltas to be excluded in the creation of the generated file. See the –l keyletter for the list format.
-e Indicates that the get is for the purpose of editing or making a change (delta) to the SCCS file via a subsequent use of delta(1). The -e keyletter used in a get for a particular version (SID) of the SCCS file prevents further gets for editing on the same SID until delta is executed or the j (joint edit) flag is set in the SCCS file [see admin(1)]. Concurrent use of get -e for different SIDs is always allowed.

If the g-file generated by get with an -e keyletter is accidentally ruined in the process of editing it, it may be regenerated by re-executing the get command with the -k keyletter in place of the -e keyletter.

SCCS file protection specified via the ceiling, floor, and authorized user list stored in the SCCS file [see admin(1)] are enforced when the -e keyletter is used.

-b Used with the -e keyletter to indicate that the new delta should have an SID in a new branch as shown in Table 1. This keyletter is ignored if the b flag is not present in the file [see admin(1)] or if the retrieved delta is not a leaf delta. (A leaf delta is one that has no successors on the SCCS file tree.)

Note: A branch delta may always be created from a non-leaf delta. Partial SIDs are interpreted as shown in the "SID Retrieved" column of Table 1.

-k Suppresses replacement of identification keywords (see below) in the retrieved text by their value. The -k keyletter is implied by the -e keyletter.

-l[p] Causes a delta summary to be written into an l-file. If -lp is used then an l-file is not created; the delta summary is written on the standard output instead. See FILES for the format of the l-file.

-p Causes the text retrieved from the SCCS file to be written on the standard output. No g-file is created. All output which normally goes to the standard output goes to file descriptor 2 instead, unless the -s keyletter is used, in which case it disappears.

-s Suppresses all output normally written on the standard output. However, fatal error messages (which always go to file descriptor 2) remain unaffected.

-m Causes each text line retrieved from the SCCS file to be preceded by the SID of the delta that inserted the text line in the SCCS file. The format is: SID, followed by a horizontal tab, followed by the text line.

-n Causes each generated text line to be preceded with the %M% identification keyword value (see below). The format is: %M% value, followed by a horizontal tab, followed by the text line. When both the -m and -n keyletters are used, the format is: %M% value, followed by a horizontal tab, followed by the -m keyletter generated format.

-g Suppresses the actual retrieval of text from the SCCS file. It is primarily used to generate an l-file, or to verify the existence of a particular SID.

-t Used to access the most recently created delta in a given release (e.g., -r1), or release and level (e.g., -r1.2).
Substitute string for all occurrences of \%W\% when getting the file.

The delta sequence number of the SCCS file delta (version) to be retrieved [see sccsfile(5)]. This keyletter is used by the comb(1) command; it is not a generally useful keyletter. If both the -r and -a keyletters are specified, only the -a keyletter is used. Care should be taken when using the -a keyletter in conjunction with the -e keyletter, as the SID of the delta to be created may not be what one expects. The -r keyletter can be used with the -a and -e keyletters to control the naming of the SID of the delta to be created.

For each file processed, get responds (on the standard output) with the SID being accessed and with the number of lines retrieved from the SCCS file.

If the -e keyletter is used, the SID of the delta to be made appears after the SID accessed and before the number of lines generated. If there is more than one named file or if a directory or standard input is named, each file name is printed (preceded by a new-line) before it is processed. If the -i keyletter is used included deltas are listed following the notation "Included"; if the -x keyletter is used, excluded deltas are listed following the notation "Excluded".

### TABLE 1. Determination of SCCS Identification String

<table>
<thead>
<tr>
<th>SID* Specified</th>
<th>-b Keyletter Used†</th>
<th>Other Conditions</th>
<th>SID Retrieved</th>
<th>SID of Delta to be Created</th>
</tr>
</thead>
<tbody>
<tr>
<td>none‡</td>
<td>no</td>
<td>R defaults to mR</td>
<td>mR.mL</td>
<td>mR.(mL+1)</td>
</tr>
<tr>
<td>none‡</td>
<td>yes</td>
<td>R defaults to mR</td>
<td>mR.mL</td>
<td>mR.mL.(mB+1).1</td>
</tr>
<tr>
<td>R</td>
<td>no</td>
<td>R = mR</td>
<td>mR.mL</td>
<td>R.1***</td>
</tr>
<tr>
<td>R</td>
<td>yes</td>
<td>R &gt; mR</td>
<td>mR.mL</td>
<td>mR.mL.(mB+1).1</td>
</tr>
<tr>
<td>R</td>
<td>yes</td>
<td>R = mR</td>
<td>mR.mL</td>
<td>mR.mL.(mB+1).1</td>
</tr>
<tr>
<td>R</td>
<td>-</td>
<td>R &lt; mR and R does not exist</td>
<td>hR.mL**</td>
<td>hR.mL.(mB+1).1</td>
</tr>
<tr>
<td>R</td>
<td>-</td>
<td>Trunk succ. in release &gt; R and R exists</td>
<td>R.mL</td>
<td>R.mL.(mB+1).1</td>
</tr>
<tr>
<td>R.L</td>
<td>no</td>
<td>No trunk succ.</td>
<td>R.L</td>
<td>R.(L+1)</td>
</tr>
<tr>
<td>R.L</td>
<td>yes</td>
<td>No trunk succ.</td>
<td>R.L</td>
<td>R.L.(mB+1).1</td>
</tr>
<tr>
<td>R.L</td>
<td>-</td>
<td>Trunk succ. in release ≥ R</td>
<td>R.L</td>
<td>R.L.(mB+1).1</td>
</tr>
<tr>
<td>R.L.B</td>
<td>no</td>
<td>No branch succ.</td>
<td>R.L.B.mS</td>
<td>R.L.B.(mS+1)</td>
</tr>
<tr>
<td>R.L.B</td>
<td>yes</td>
<td>No branch succ.</td>
<td>R.L.B.mS</td>
<td>R.L.B.(mB+1).1</td>
</tr>
<tr>
<td>R.L.B.S</td>
<td>no</td>
<td>No branch succ.</td>
<td>R.L.B.S</td>
<td>R.L.B.S.(S+1)</td>
</tr>
<tr>
<td>R.L.B.S</td>
<td>yes</td>
<td>No branch succ.</td>
<td>R.L.B.S</td>
<td>R.L.B.S.(mB+1).1</td>
</tr>
<tr>
<td>R.L.B.S</td>
<td>-</td>
<td>Branch succ.</td>
<td>R.L.B.S</td>
<td>R.L.B.(S+1)</td>
</tr>
</tbody>
</table>

* "R", "L", "B", and "S" are the "release", "level", "branch", and "sequence" components of the SID, respectively; "m" means "maximum". Thus, for example, "R.mL" means "the maximum level number within release R"; "R.L.(mB+1).1" means "the first sequence number on the new branch (i.e., maximum branch number plus one) of level L within release R". Note that if the SID specified is of the form "R.L", "R.L.B", or "R.L.B.S", each of the specified components must exist.
** "hR" is the highest existing release that is lower than the specified, nonexistent, release R.

*** This is used to force creation of the first delta in a new release.

# Successor.

† The -b keyletter is effective only if the b flag [see admin (1)] is present in the file. An entry of - means "irrelevant".

‡ This case applies if the d (default SID) flag is not present in the file. If the d flag is present in the file, then the SID obtained from the d flag is interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies.

IDENTIFICATION KEYWORDS

Identifying information is inserted into the text retrieved from the SCCS file by replacing identification keywords with their value wherever they occur. The following keywords may be used in the text stored in an SCCS file:

Keyword | Value
--- | ---
%M% | Module name: either the value of the m flag in the file [see admin(1)], or if absent, the name of the SCCS file with the leading s. removed.
%I% | SCCS identification (SID) (%R%%L%%B%%S%) of the retrieved text.
%R% | Release.
%L% | Level.
%B% | Branch.
%S% | Sequence.
%D% | Current date (YY/MM/DD).
%H% | Current date (MM/DD/YY).
%T% | Current time (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: value of the t flag in the SCCS file [see admin(1)].
%F% | SCCS file name.
%P% | Fully qualified SCCS file name.
%Q% | The value of the q flag in the file [see admin(1)].
%C% | Current line number. This keyword is intended for identifying messages output by the program such as "this should not have happened" type errors. It is not intended to be used on every line to provide sequence numbers.
%Z% | The 4-character string @(1) recognizable by what(1).
%W% | A shorthand notation for constructing what(1) strings for UNIX system program files. %W% = %Z%%M%<horizontal-tab>%I%
%A% | Another shorthand notation for constructing what(1) strings for non-UNIX system program files. %A% = %Z%%Y%%M% %I%%Z%

Several auxiliary files may be created by get. These files are known generically as the g-file, l-file, p-file, and z-file. The letter before the hyphen is called the tag. An auxiliary file name is formed from the SCCS file name: the last component of all SCCS file names must be of the form s.module-name, the auxiliary files are named by replacing the leading s with the tag. The g-file is an exception to this scheme: the g-file is named by removing the s. prefix. For example, s.xyz.c, the auxiliary file names would be xyz.c, l.xyz.c, p.xyz.c, and z.xyz.c, respectively.

The g-file, which contains the generated text, is created in the current directory (unless the -p keyletter is used). A g-file is created in all cases, whether or not any lines of text were generated by the get. It is owned by the real user. If the -k
keyletter is used or implied its mode is 644; otherwise its mode is 444. Only the real user need have write permission in the current directory.

The l-file contains a table showing which deltas were applied in generating the retrieved text. The l-file is created in the current directory if the -l keyletter is used; its mode is 444 and it is owned by the real user. Only the real user need have write permission in the current directory.

Lines in the l-file have the following format:

a. A blank character if the delta was applied;
   * otherwise.

b. A blank character if the delta was applied or was not applied and ignored;
   * if the delta was not applied and was not ignored.

c. A code indicating a "special" reason why the delta was or was not applied: "I": Included.
   "X": Excluded.
   "C": Cut off (by a -c keyletter).

d. Blank.

e. SCCS identification (SID).

f. Tab character.

g. Date and time (in the form YY/MM/DD HH:MM:SS) of creation.

h. Blank.

i. Login name of person who created delta.

The comments and MR data follow on subsequent lines, indented one horizontal tab character. A blank line terminates each entry.

The p-file is used to pass information resulting from a get with an -e keyletter along to delta. Its contents are also used to prevent a subsequent execution of get with an -e keyletter for the same SID until delta is executed or the joint edit flag, j, [see admin(1)] is set in the SCCS file. The p-file is created in the directory containing the SCCS file and the effective user must have write permission in that directory. Its mode is 644 and it is owned by the effective user. The format of the p-file is: the gotten SID, followed by a blank, followed by the SID that the new delta will have when it is made, followed by a blank, followed by the login name of the real user, followed by a blank, followed by the date-time the get was executed, followed by a blank and the -i keyletter argument if it was present, followed by a blank and the -x keyletter argument if it was present, followed by a new-line. There can be an arbitrary number of lines in the p-file at any time; no two lines can have the same new delta SID.

The z-file serves as a lock-out mechanism against simultaneous updates. Its contents are the binary (2 bytes) process ID of the command (i.e., get) that created it. The z-file is created in the directory containing the SCCS file for the duration of get. The same protection restrictions as those for the p-file apply for the z-file. The z-file is created mode 444.

**FILES**

g-file

Existed before the execution of delta; removed after completion of delta.

p-file

Existed before the execution of delta; may exist after completion of delta.

q-file

Created during the execution of delta; removed after completion of delta.
x-file  Created during the execution of *delta*; renamed to SCCS file after completion of *delta*.

z-file  Created during the execution of *delta*; removed during the execution of *delta*.

d-file  Created during the execution of *delta*; removed after completion of *delta*.

/usr/bin/bdiff  Program to compute differences between the “gotten” file and the *g-file*.

**SEE ALSO**

admin(1), delta(1), help(1), prs(1), what(1).

**DIAGNOSTICS**

Use *help*(1) for explanations.

**BUGS**

If the effective user has write permission (either explicitly or implicitly) in the directory containing the SCCS files, but the real user does not, then only one file may be named when the −e keyletter is used.
NAME

g getopt – parse command options

SYNOPSIS

set -- `g getopt optstring `$

DESCRIPTION

WARNING: Start using the new command getopt(1) in place of getopt(1). getopt(1) will not be supported in the next major release. For more information, see the WARNINGS section, below.

g getopt is used to break up options in command lines for easy parsing by shell procedures and to check for legal options. opstring is a string of recognized option letters (see getopt(3C)); if a letter is followed by a colon, the option is expected to have an argument which may or may not be separated from it by white space. The special option -- is used to delimit the end of the options. If it is used explicitly, getopt will recognize it; otherwise, getopt will generate it; in either case, getopt will place it at the end of the options. The positional parameters ($1 $2 ...) of the shell are reset so that each option is preceded by a – and is in its own positional parameter; each option argument is also parsed into its own positional parameter.

EXAMPLE

The following code fragment shows how one might process the arguments for a command that can take the options a or b, as well as the option o, which requires an argument:

```
set -- `g getopt abo: `$
if [ $? != 0 ]
    then
echo $USAGE
    exit 2
fi
for i in $*
do
case $i in
    -a | -b) FLAG=$i; shift;;
    -o) OARG=$2; shift 2;;
    --) shift; break;;
esac
done
```

This code will accept any of the following as equivalent:

```
cmd -aoarg file file
cmd -a -o arg file file
cmd -oarg -a file file
cmd -a -oarg -- file file
```

SEE ALSO

getopts(1), sh(1), getopt(3C).

DIAGNOSTICS

g getopt prints an error message on the standard error when it encounters an option letter not included in opstring.

WARNINGS

g getopt(1) does not support the part of Rule 8 of the command syntax standard (see intro(1)) that permits groups of option-arguments following an option to be separated by white space and quoted. For example,
cmd -a -b -o "xxx z yy" file

is not handled correctly). To correct this deficiency, use the new command *getopts(1)*
in place of *getopt(1)*.

*getopt(1)* will not be supported in the next major release. For this release a conver-
sion tool has been provided, *getoptcut*. For more information about *getopts* and
*getoptcut*, see the *getopts(1)* manual page.

If an option that takes an option-argument is followed by a value that is the same as
one of the options listed in *optstring* (referring to the earlier EXAMPLE section, but
using the following command line:  cmd -o -a file), *getopt* will always treat -a
as an option-argument to -o; it will never recognize -a as an option. For this case,
the for loop in the example will shift past the file argument.
NAME
getopts, getoptcvt - parse command options

SYNOPSIS
getopts optstring name [arg ...]
/usr/lib/getoptcvt [-b] file

DESCRIPTION
getopts is used by shell procedures to parse positional parameters and to check for legal options. It supports all applicable rules of the command syntax standard (see Rules 3-10, intro(1)). It should be used in place of the getopt(1) command. (See the WARNING, below.)

optstring must contain the option letters the command using getopt will recognize; if a letter is followed by a colon, the option is expected to have an argument, or group of arguments, which must be separated from it by white space.

Each time it is invoked, getopt will place the next option in the shell variable name and the index of the next argument to be processed in the shell variable OPTIND. Whenever the shell or a shell procedure is invoked, OPTIND is initialized to 1.

When an option requires an option-argument, getopt places it in the shell variable OPTARG.

If an illegal option is encountered, ? will be placed in name.

When the end of options is encountered, getopt exits with a non-zero exit status. The special option "--" may be used to delimit the end of the options.

By default, getopt parses the positional parameters. If extra arguments (arg ...) are given on the getopt command line, getopt will parse them instead.

/usr/lib/getoptcvt reads the shell script in file, converts it to use getopt(1) instead of getopt(1), and writes the results on the standard output.

-b the results of running /usr/lib/getoptcvt will be portable to earlier releases of the UNIX system. /usr/lib/getoptcvt modifies the shell script in file so that when the resulting shell script is executed, it determines at run time whether to invoke getopt(1) or getopt(1).

So all new commands will adhere to the command syntax standard described in intro(1), they should use getopt(1) or getopt(3C) to parse positional parameters and check for options that are legal for that command (see WARNINGS, below).

EXAMPLE

The following fragment of a shell program shows how one might process the arguments for a command that can take the options a or b, as well as the option o, which requires an option-argument:

    while getopt abo: c
do
        case $c in
        a | b)
            FLAG=$c;
        o)
            OARG=$OFTARG;
        \
        )
            echo $USAGE
            exit 2;
        esac
    done
    shift \ expr $OPTIND = 1\n

Stardent 1500/3000  207
This code will accept any of the following as equivalent:

```
cmd -a -b -o "xxx z yy" file
```
```
cmd -a -b -o "xxx z yy" -- file
```
```
cmd -ab -o xxx,z,yy file
```
```
cmd -ab -o "xxx z yy" file
```
```
cmd -o xxx,z,yy -b -a file
```

**SEE ALSO**

intro(1), sh(1), getopts(3C).

**WARNING**

Although the following command syntax rule (see intro(1)) relaxations are permitted under the current implementation, they should not be used because they may not be supported in future releases of the system. As in the EXAMPLE section above, `a` and `b` are options, and the option `o` requires an option-argument:

```
cmd -aboxxx file  (Rule 5 violation: options with
  option-arguments must not be grouped with other options)
```
```
cmd -ab -oxxx file  (Rule 6 violation: there must be
  white space after an option that takes an option-argument)
```

Changing the value of the shell variable OPTIND or parsing different sets of arguments may lead to unexpected results.

**DIAGNOSTICS**

`getopts` prints an error message on the standard error when it encounters an option letter not included in `opistring`. 
**NAME**

getty – set terminal type, modes, speed, and line discipline

**SYNOPSIS**

```
/etc/getty [ -h ] [ -t timeout ] line [ speed [ type [ linedisc ] ] ]
/etc/getty -c file
```

**DESCRIPTION**

`getty` is a program that is invoked by `init(1M)`. It is the second process in the series, `(init-getty-login-shell)` that ultimately connects a user with the UNIX system. It can only be executed by the super-user; that is, a process with the user-ID of `root`. Initially `getty` prints the login message field for the entry it is using from `/etc/gettydefs`. `getty` reads the user's login name and invokes the `login(1)` command with the user's name as argument. While reading the name, `getty` attempts to adapt the system to the speed and type of terminal being used. It does this by using the options and arguments specified.

`Line` is the name of a tty line in `/dev` to which `getty` is to attach itself. `getty` uses this string as the name of a file in the `/dev` directory to open for reading and writing. Unless `getty` is invoked with the `-h` flag, `getty` will force a hangup on the line by setting the speed to zero before setting the speed to the default or specified speed. The `-t` flag plus `timeout` (in seconds), specifies that `getty` should exit if the open on the line succeeds and no one types anything in the specified number of seconds.

`Speed`, the optional second argument, is a label to a speed and tty definition in the file `/etc/gettydefs`. This definition tells `getty` what speed to initially run, what the login message should look like, what the initial tty settings are, and what speed to try next should the user indicate that the speed is inappropriate (by typing a `<break>` character). The default `speed` is 300 baud.

`Type`, the optional third argument, is a character string describing to `getty` what type of terminal is connected to the line in question. `getty` recognizes the following types:

- `none` default
- `ds40-1` Datasync40/1
- `tektronix,tek` Tektronix
- `vt61` DEC vt61
- `vt100` DEC vt100
- `hp45` Hewlett-Packard 45
- `c100` Concept 100

The default terminal is `none`; i.e., any crt or normal terminal unknown to the system. Also, for terminal type to have any meaning, the virtual terminal handlers must be compiled into the operating system. They are available, but not compiled in the default condition.

`Linedisc`, the optional fourth argument, is a character string describing which line discipline to use in communicating with the terminal. Again the hooks for line disciplines are available in the operating system but there is only one presently available, the default line discipline, LDISC0.

When given no optional arguments, `getty` sets the `speed` of the interface to 300 baud, specifies that raw mode is to be used (awaken on every character), that echo is to be suppressed, either parity allowed, new-line characters will be converted to carriage return-line feed, and tab expansion performed on the standard output. It types the login message before reading the user's name a character at a time. If a null character (or framing error) is received, it is assumed to be the result of the user pushing the "break" key. This will cause `getty` to attempt the next `speed` in the series. The series that `getty` tries is determined by what it finds in `/etc/gettydefs`. 
After the user's name has been typed in, it is terminated by a new-line or carriage-return character. The latter results in the system being set to treat carriage returns appropriately (see ioctl(2)).

The user's name is scanned to see if it contains any lower-case alphabetic characters; if not, and if the name is non-empty, the system is told to map any future upper-case characters into the corresponding lower-case characters.

Finally, login is exec'd with the user's name as an argument. Additional arguments may be typed after the login name. These are passed to login, which will place them in the environment (see login(1)).

A check option is provided. When getty is invoked with the -c option and file, it scans the file as if it were scanning /etc/gettydefs and prints out the results to the standard output. If there are any unrecognized modes or improperly constructed entries, it reports these. If the entries are correct, it prints out the values of the various flags. See ioctl(2) to interpret the values. Note that some values are added to the flags automatically.

FILES

/etc/gettydefs
/etc/issue

SEE ALSO

ct(1C), gettydefs(4), init(1M), inittab(4), ioctl(2), login(1), tty(7).

BUGS

While getty understands simple single character quoting conventions, it is not possible to quote certain special control characters used by getty. Thus, you cannot login via getty and type a #, @, /, \, backspace, ^U, ^D, or & as part of your login name or arguments. getty uses them to determine when the end of the line has been reached, which protocol is being used, and what the erase character is. They will always be interpreted as having their special meaning.
NAME
grep – search a file for a pattern

SYNOPSIS
grep [options] limited regular expression [file ...]

DESCRIPTION

grep searches files for a pattern and prints all lines that contain that pattern. grep uses limited regular expressions (expressions that have string values that use a subset of the possible alphanumeric and special characters) like those used with ed (1) to match the patterns. It uses a compact non-deterministic algorithm.

Be careful using the characters $, *, [^, |, (, ), and \ in the limited regular expression because they are also meaningful to the shell. It is safest to enclose the entire limited regular expression in single quotes ‘...’.

If no files are specified, grep assumes standard input. Normally, each line found is copied to standard output. The file name is printed before each line found if there is more than one input file.

Command line options are:

-`b` Precede each line by the block number on which it was found. This can be useful in locating block numbers by context (first block is 0).
-`c` Print only a count of the lines that contain the pattern.
-`i` Ignore upper/lower case distinction during comparisons.
-`l` Print the names of files with matching lines once, separated by new-lines. Does not repeat the names of files when the pattern is found more than once.
-`n` Precede each line by its line number in the file (first line is 1).
-`s` Suppress error messages about nonexistent or unreadable files
-`v` Print all lines except those that contain the pattern.

SEE ALSO
ed(1), egrep(1), fgrep(1), sed(1), sh(1).

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files (even if matches were found).

BUGS

Lines are limited to BUFSIZE characters; longer lines are truncated. BUFSIZE is defined in /usr/include/stdio.h.

If there is a line with embedded nulls, grep will only match up to the first null; if it matches, it will print the entire line.
**NAME**

hd – hexadecimal and ASCII dump

**SYNOPSIS**

hd [file] [[ + ] offset [.][ b ]]

**DESCRIPTION**

`hd` dumps `file` in both hexadecimal and ASCII format. The `file` argument specifies which file is to be dumped. If no `file` argument is specified, the standard input is used.

The offset argument specifies the offset in the file where dumping is to commence. This argument is normally interpreted as octal bytes. If . is appended, the offset is interpreted in decimal. If b is appended, the offset is interpreted in blocks of 512 bytes. If the file argument is omitted, the offset argument must be preceded by +.

Dumping continues until end-of-file.
NAME

head – give first few lines

SYNOPSIS

head [ -count ] [ file ... ]

DESCRIPTION

This filter gives the first count lines of each of the specified files, or of the standard input. If count is omitted it defaults to 10.

SEE ALSO

tail(1)
NAME

help - ask for help regarding SCCS

SYNOPSIS

help [args]

DESCRIPTION

help finds information to explain a message from a command or explain the use of a command. Zero or more arguments may be supplied. If no arguments are given, help prompt for one.

The arguments may be either message numbers (which normally appear in parentheses following messages) or command names, of one of the following types:

type 1  Begins with non-numerics, ends in numerics. The non-numeric prefix is usually an abbreviation for the program or set of routines which produced the message (for example, ge6, for message 6 from the get command).

type 2  Does not contain numerics (as a command, such as get)

type 3  Is all numeric (for example, 212)

The response of the program will be the explanatory information related to the argument, if there is any. When all else fails, try help stuck.

FILES

/usr/lib/help  directory containing files of message text.
NAME

hostid – set or print identifier of current host system

SYNOPSIS

hostid [ identifier ]

DESCRIPTION

The hostid command prints the identifier of the current host in hexadecimal. This numeric value is expected to be unique across all hosts and is commonly set to the host’s Internet address. The super-user can set the hostid by giving a hexadecimal argument or the hostname; this is usually done in the startup script /etc/rc.local.

SEE ALSO

gethostid(2), sethostid(2)
NAME

hostname – set or print name of current host system

SYNOPSIS

hostname [ nameofhost ]

DESCRIPTION

The hostname command prints the name of the current host, as given before the “login” prompt. The super-user can set the hostname by giving an argument.

SEE ALSO

gethostname(2), sethostname(2)
NAME
id – print user and group IDs and names

SYNOPSIS
id

DESCRIPTION
id outputs the user and group IDs and the corresponding names of the invoking process. If the effective and real IDs are different, both are printed.

SEE ALSO
getuid(2), logname(1).
NAME
indent – indent and format C program source

SYNOPSIS
[-v | -nv ]

DESCRIPTION
indent is a C program formatter. It reformats the C program in the input-file according
to the switches. The switches which can be specified are described below. They
may appear before or after the file names.

NOTE: If you only specify an input-file, the formatting is done ‘in-place’, that is, the
formatted file is written back into input-file and a backup copy of input-file is written
in the current directory. If input-file is named ‘/blah/blah/file’, the backup file is
named file.BAK.

If output-file is specified, indent checks to make sure it is different from input-file.

OPTIONS
The options listed below control the formatting style imposed by indent.
-bad,-nbad If -bad is specified, a blank line is forced after every block of
declarations. Default: -nbad.
-bap,-nbap If -bap is specified, a blank line is forced after every procedure
body. Default: -nbap.
-bbb,-nbbb If -bbb is specified, a blank line is forced before every block com-
ment. Default: -nbbb.
-bc,-nbc If -bc is specified, then a newline is forced after each comma in a
declaration. -nbc turns off this option. The default is -nbc.
-br,-bl Specifying -bl lines up compound statements like this:
  if (...) {
    code
  }
Specifying -br (the default) makes them look like this:
  if (...) {
    code
  }
-cn The column in which comments on code start. The default is 33.
-cdn The column in which comments on declarations start. The default is
for these comments to start in the same column as those on code.
-cdb,-ncdb Enables (disables) the placement of comment delimiters on blank
lines. With this option enabled, comments look like this:
/*
 * this is a comment
 */
Rather than like this:
/* this is a comment */
This only affects block comments, not comments to the right of
code. The default is -cdb.
-ce, -nce Enables (disables) forcing 'else's to cuddle up to the immediately preceding}'. The default is -ce.

-cin Sets the continuation indent to be n. Continuation lines will be indented that far from the beginning of the first line of the statement. Parenthesized expressions have extra indentation added to indicate the nesting, unless -lp is in effect. -ci defaults to the same value as -i.

-clin Causes case labels to be indented n tab stops to the right of the containing switch statement. -c10.5 causes case labels to be indented half a tab stop. The default is -c10. (This is the only option that takes a fractional argument.)

-drn Controls the placement of comments which are not to the right of code. Specifying -dr1 means that such comments are placed one indentation level to the left of code. The default -d0 lines up these comments with the code. See the section on comment indentation below.

-din Specifies the indentation, in character positions, from a declaration keyword to the following identifier. The default is -d16.

-dj, -ndj -dj left justifies declarations. -ndj indents declarations the same as code. The default is -ndj.

-ei, -nei Enables (disables) special else-if processing. If enabled, if's following else's will have the same indentation as the preceding if statement. The default is -ei.

-fc1, -nfc1 Enables (disables) the formatting of comments that start in column 1. Often, comments whose leading '/' is in column 1 have been carefully hand formatted by the programmer. In such cases, -nfc1 should be used. The default is -fc1.

-in The number of spaces for one indentation level. The default is 8.

-ip, -nip Enables (disables) the indentation of parameter declarations from the left margin. The default is -ip.

-ln Maximum length of an output line. The default is 78.

-lp, -nlp Lines up code surrounded by parenthesis in continuation lines. If a line has a left paren which is not closed on that line, then continuation lines will be lined up to start at the character position just after the left paren. For example, here is how a piece of continued code looks with -nlp in effect:

```
p1 = first_procedure(second_procedure(p2, p3),
                     third_procedure(p4, p5));
```

With -lp in effect (the default) the code looks somewhat clearer:

```
p1 = first_procedure(second_procedure(p2, p3),
                     third_procedure(p4, p5));
```

Inserting two more newlines we get:

```
p1 = first_procedure(second_procedure(p2, p3),
                     third_procedure(p4, p5));
```

-npro Causes the profile files, ‘./indent.pro’ and ‘–./indent.pro’, to be ignored.
-pcs,-npcs  If true (-pcs) all procedure calls will have a space inserted between
the name and the ‘.’. The default is -npcs.

-ps,-nps  If true (-ps) the pointer following operator ‘->’ will be surrounded
by spaces on either side. The default is -nps.

-psl,-npsl  If true (-psl) the names of procedures being defined are placed in
column 1 – their types, if any, will be left on the previous lines. The
default is -psl.

-sc,-nsc  Enables (disables) the placement of asterisks (*’s) at the left edge of
all comments. The default is -sc.

-sob,-nsob  If -sob is specified, indent will swallow optional blank lines. You
can use this to get rid of blank lines after declarations. Default:
-nsob.

-st  Causes indent to take its input from stdin, and put its output to
stdout.

-Ttypename  Adds typename to the list of type keywords. Names accumulate: -T
can be specified more than once. You need to specify all the
typenames that appear in your program that are defined by
typedefs – nothing will be harmed if you miss a few, but the pro-
gram won’t be formatted as nicely as it should. This sounds like a
painful thing to have to do, but it’s really a symptom of a problem
in C: typedef causes a syntactic change in the language and indent
can’t find all typedefs.

-troff  Causes indent to format the program for processing by troff. It will
produce a fancy listing in much the same spirit as vgrind. If the
output file is not specified, the default is standard output, rather
than formatting in place.

-v,-nv  -v turns on ‘verbose’ mode; -nv turns it off. When in verbose
mode, indent reports when it splits one line of input into two or
more lines of output, and gives some size statistics at completion.
The default is -nv.

FURTHER DESCRIPTION

You may set up your own ‘profile’ of defaults to indent by creating a file called
.indent.pro in either your login directory and/or the current directory and including
whatever switches you like. Switches in ‘.indent.pro’ in the current directory over-
ride those in your login directory (with the exception of -T type definitions, which
just accumulate). If indent is run and a profile file exists, then it is read to set up the
program’s defaults. The switches should be separated by spaces, tabs or newlines.
Switches on the command line, however, override profile switches.

Comments

‘Box’ comments. Indent assumes that any comment with a dash or star immediately
after the start of comment (that is, ‘/-’ or ‘/**’ ) is a comment surrounded by a box
of stars. Each line of such a comment is left unchanged, except that its indentation
may be adjusted to account for the change in indentation of the first line of the com-
ment.

Straight text. All other comments are treated as straight text. Indent fits as many
words (separated by blanks, tabs, or newlines) on a line as possible. Blank lines
break paragraphs.
Comment indentation

If a comment is on a line with code it is started in the ‘comment column’, which is set by the \-cn command line parameter. Otherwise, the comment is started at n indentation levels less than where code is currently being placed, where n is specified by the \-dn command line parameter. If the code on a line extends past the comment column, the comment starts further to the right, and the right margin may be automatically extended in extreme cases.

Preprocessor lines

In general, indent leaves preprocessor lines alone. The only reformatting that it will do is to straighten up trailing comments. It leaves embedded comments alone. Conditional compilation (#ifdef...#endif) is recognized and indent attempts to correctly compensate for the syntactic peculiarities introduced.

C syntax

Indent understands a substantial amount about the syntax of C, but it has a ‘forgiving’ parser. It attempts to cope with the usual sorts of incomplete and misformed syntax. In particular, the use of macros like:

\#define forever for(;;)

is handled properly.

FILES

./indent.pro  profile file
~/.indent.pro  profile file

BUGS

Indent has even more switches than ls.

A common mistake that often causes grief is typing:

indent *.c
to the shell in an attempt to indent all the C programs in a directory. This is probably a bug, not a feature.
NAME
infocmp – compare or print out terminfo descriptions

SYNOPSIS
[-A directory] [-B directory] [termname ...

DESCRIPTION
infocmp can be used to compare a binary terminfo(4) entry with other terminfo entries,
rewrite a terminfo(4) description to take advantage of the use= terminfo field, or print
out a terminfo(4) description from the binary file (term(4)) in a variety of formats. In
all cases, the boolean fields will be printed first, followed by the numeric fields, fol-
lowed by the string fields.

Default Options
If no options are specified and zero or one termnames are specified, the –I option will
be assumed. If more than one termname is specified, the –d option will be assumed.

Comparison Options [–d] [–c] [–n]
infocmp compares the terminfo(4) description of the first terminal termname with each
of the descriptions given by the entries for the other terminal’s termnames. If a capa-
bility is defined for only one of the terminals, the value returned will depend on the
type of the capability: F for boolean variables, –I for integer variables, and NULL for
string variables.

-d produce a list of each capability that is different. In this manner, if one has
two entries for the same terminal or similar terminals, using infocmp will show what is
different between the two entries. This is sometimes neces-
sary when more than one person produces an entry for the same terminal
and one wants to see what is different between the two.

-c produce a list of each capability that is common between the two entries.
Capabilities that are not set are ignored. This option can be used as a quick
check to see if the –u option is worth using.

-n produce a list of each capability that is in neither entry. If no termnames are
given, the environment variable TERM will be used for both of the term-
names. This can be used as a quick check to see if anything was left out of
the description.

Source Listing Options [–I] [–L] [–C] [–r]
The –I, –L, and –C options will produce a source listing for each terminal named.

-I use the terminfo(4) names

-L use the long C variable name listed in <term.h>

-C use the termcap names

-r when using –C, put out all capabilities in termcap form

If no termnames are given, the environment variable TERM will be used for the termi-
nal name.

The source produced by the –C option may be used directly as a termcap entry, but
not all of the parameterized strings may be changed to the termcap format. infocmp
will attempt to convert most of the parameterized information, but that which it
doesn’t will be plainly marked in the output and commented out. These should be
edited by hand.

All padding information for strings will be collected together and placed at the
beginning of the string where termcap expects it. Mandatory padding (padding infor-
mation with a trailing ‘/’) will become optional.
All *termcap* variables no longer supported by *terminfo*(4), but which are derivable from other *terminfo*(4) variables, will be output. Not all *terminfo*(4) capabilities will be translated; only those variables which were part of *termcap* will normally be output. Specifying the `-r` option will take off this restriction, allowing all capabilities to be output in *termcap* form.

Note that because padding is collected to the beginning of the capability, not all capabilities are output, mandatory padding is not supported, and *termcap* strings were not as flexible, it is not always possible to convert a *terminfo*(4) string capability into an equivalent *termcap* format. Not all of these strings will be able to be converted. A subsequent conversion of the *termcap* file back into *terminfo*(4) format will not necessarily reproduce the original *terminfo*(4) source.

Some common *terminfo* parameter sequences, their *termcap* equivalents, and some terminal types which commonly have such sequences, are:

<table>
<thead>
<tr>
<th>Terminfo</th>
<th>Termcap</th>
<th>Representative Terminals</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>%p1%c</code></td>
<td><code>%</code></td>
<td>adm</td>
</tr>
<tr>
<td><code>%p1%d</code></td>
<td><code>%d</code></td>
<td>hp, ANSI standard, vt100</td>
</tr>
<tr>
<td><code>%p1%*c</code></td>
<td><code>%+x</code></td>
<td>concept</td>
</tr>
<tr>
<td><code>%i</code></td>
<td><code>%i</code></td>
<td>ANSI standard, vt100</td>
</tr>
<tr>
<td><code>%p1%?%*c&gt;%t%p1%y%+%;</code></td>
<td><code>%&gt;xy</code></td>
<td>concept</td>
</tr>
<tr>
<td><code>%p2 is printed before %p1</code></td>
<td><code>%r</code></td>
<td>hp</td>
</tr>
</tbody>
</table>

**Use** Option `[-u]`

- `u` produce a *terminfo*(4) source description of the first terminal *termname* which is relative to the sum of the descriptions given by the entries for the other terminals *termnames*. It does this by analyzing the differences between the first *termname* and the other *termnames* and producing a description with *use=* fields for the other terminals. In this manner, it is possible to retrofit generic *terminfo* entries into a terminal’s description. Or, if two similar terminals exist, but were coded at different times or by different people so that each description is a full description, using *infocmp* will show what can be done to change one description to be relative to the other.

A capability will get printed with an at-sign (@) if it no longer exists in the first *termname*, but one of the other *termname* entries contains a value for it. A capability’s value gets printed if the value in the first *termname* is not found in any of the other *termname* entries, or if the first of the other *termname* entries that has this capability gives a different value for the capability than that in the first *termname*.

The order of the other *termname* entries is significant. Since the *terminfo* compiler *tlc*(1M) does a left-to-right scan of the capabilities, specifying two *use=* entries that contain differing entries for the same capabilities will produce different results depending on the order that the entries are given in. *infocmp* will flag any such inconsistencies between the other *termname* entries as they are found.

Alternatively, specifying a capability after a *use=* entry that contains that capability will cause the second specification to be ignored. Using *infocmp* to recreate a description can be a useful check to make sure that everything was specified correctly in the original source description.

Another error that does not cause incorrect compiled files, but will slow down the compilation time, is specifying extra *use=* fields that are superfluous. *infocmp* will flag any other *termname* *use=* fields that were not needed.
Terminal Information Utilities

INFOCMP (1M)

Other Options [-s d|i|l|c] [-V] [-I] [-w width]

- sort the fields within each type according to the argument below:
  -s leave fields in the order that they are stored in the terminfo database.
  -d sort by terminfo name.
  -i sort by the long C variable name.
  -c sort by the termcap name.

If no -s option is given, the fields printed out will be sorted alphabetically
by the terminfo name within each type, except in the case of the -C or the -L
options, which cause the sorting to be done by the termcap name or the long
C variable name, respectively.

- v print out tracing information on standard error as the program runs.
- V print out the version of the program in use on standard error and exit.
- I cause the fields to printed out one to a line. Otherwise, the fields will be
  printed several to a line to a maximum width of 60 characters.

- w change the output to width characters.

Changing Databases [-A directory] [-B directory]
The location of the compiled terminfo(4) database is taken from the environment variable TERMINFO. If the variable is not defined, or the terminal is not found in that location, the system terminfo(4) database, usually in /usr/lib/terminfo, will be used.
The options -A and -B may be used to override this location. The -A option will set
TERMINFO for the first termname and the -B option will set TERMINFO for the other
termmnames. With this, it is possible to compare descriptions for a terminal with the
same name located in two different databases. This is useful for comparing descriptions
for the same terminal created by different people. Otherwise the terminals
would have to be named differently in the terminfo(4) database for a comparison to be
made.

FILES

/usr/lib/terminfo/??/* compiled terminal description database

DIAGNOSTICS

malloc is out of space!

There was not enough memory available to process all the terminal
descriptions requested. Run infocomp several times, each time includ-
ing a subset of the desired termmnames.

use= order dependency found:
A value specified in one relative terminal specification was different
from that in another relative terminal specification.

'use=term' did not add anything to the description.
A relative terminal name did not contribute anything to the final
description.

must have at least two terminal names for a comparison to be done.

The -u, -d and -c options require at least two terminal names.

SEE ALSO

captnfo(1M), curses(3X), term(4), terminfo(4), tic(1M).

NOTE

The termcap database (from earlier releases of UNIX System V) may not be supplied in
future releases.
NAME
init, telinit – process control initialization

SYNOPSIS
/etc/init [ 0123456sQq ]
/etc/telinit [ 0123456sSQQabc ]

DESCRIPTION
Init
init is a general process spawner. Its primary role is to create processes from information stored in the file /etc/inittab (see inittab(4)). This file usually has init spawn getty’s on each line that a user may log in on. It also controls autonomous processes required by any particular system.

init considers the system to be in a run-level at any given time. A run-level can be viewed as a software configuration of the system where each configuration allows only a selected group of processes to exist. The processes spawned by init for each of these run-levels is defined in the inittab file. init can be in one of eight run-levels, 0–6 and S or s. The run-level is changed by having a privileged user run /etc/init. This user-spawned init sends appropriate signals to the original init spawned by the operating system when the system was rebooted, telling it which run-level to change to.

init is invoked inside the UNIX system as the last step in the boot procedure. First init looks in /etc/inittab for the initdefault entry (see inittab(4)). If there is one, init uses the run-level specified in that entry as the initial run-level to enter. If this entry is not in /etc/inittab, init requests that the user enter a run-level from the virtual system console, /dev/console. If an S or an s is entered, init goes into the SINGLE USER state. This is the only run-level that doesn’t require the existence of a properly formatted /etc/inittab file. If it doesn’t exist, then by default the only legal run-level that init can enter is the SINGLE USER state. In the SINGLE USER state the virtual console terminal /dev/console is opened for reading and writing and the command /bin/su is invoked immediately. To exit from the SINGLE USER state, use either init or telinit, to signal init to change the run-level of the system. Note that if the shell is terminated (via an end-of-file), init will only re-initialize to the SINGLE USER state.

If init fails to prompt for a new run-level when attempting to boot the system, it may be the result of the device /dev/console being linked to a device other than the physical system console (/dev/contty). If this occurs, init can be forced to relink /dev/console if you type a delete on the physical system console.

When init prompts for the new run-level, the operator may enter only one of the digits 0 through 6 or the letters S or s. If S or s is entered, init operates as previously described in the SINGLE USER state with the additional result that /dev/console is linked to the user’s terminal line, thus making it the virtual system console. A message is generated on the physical console, /dev/contty, saying where the virtual terminal has been relocated.

When init comes up initially and whenever it switches out of SINGLE USER state to normal run states, it sets the ioctl(2) states of the virtual console, /dev/console, to those modes saved in the file /etc/ioctl.syscon. This file is written by init whenever the SINGLE USER state is entered.

If a 0 through 6 is entered init enters the corresponding run-level. Any other input will be rejected and the user will be re-prompted. Note that, on the 3B2 Computer, the run-levels 0, 1, 5, and 6 are reserved states for shutting the system down; the run-levels 2, 3, and 4 are available as normal operating states.
If this is the first time init has entered a run-level other than SINGLE USER, init first scans inittab for special entries of the type boot and bootwait. These entries are performed, providing the run-level entered matches that of the entry before any normal processing of inittab takes place. In this way any special initialization of the operating system, such as mounting file systems, can take place before users are allowed onto the system. The inittab file is scanned to find all entries that are to be processed for that run-level.

Run-level 2 is defined to contain all of the terminal processes and daemons that are spawned in the multi-user environment. Hence, it is commonly referred to as the MULTI-USER state. Run-level 3 is defined to start up remote file sharing processes and daemons as well as mount and advertise remote resources. So, run-level 3 extends multi-user mode and is known as the Remote File Sharing state. Run-level 4 is available to be defined as an alternative multi-user environment configuration, however, it is not necessary for system operation and is usually unused.

In a MULTI-USER environment, the inittab file is set up so that init will create a process for each terminal on the system that the administrator sets up to respawn.

For terminal processes, ultimately the shell will terminate because of an end-of-file either typed explicitly or generated as the result of hanging up. When init receives a signal telling it that a process it spawned has died, it records the fact and the reason it died in /etc/wtmp and /etc/wtmp if it exists (see who(1)). A history of the processes spawned is kept in /etc/wtmp.

To spawn each process in the inittab file, init reads each entry and for each entry that should be respawned, it forks a child process. After it has spawned all of the processes specified by the inittab file, init waits for one of its descendant processes to die, a powerfail signal, or until init is signaled by init or telinit to change the system’s run-level. When one of these conditions occurs, init re-examines the inittab file. New entries can be added to the inittab file at any time; however, init still waits for one of the above three conditions to occur. To get around this, init Q or init q command wakes init to re-examine the inittab file immediately.

If init receives a powerfail signal (SIGPWR) it scans inittab for special entries of the type powerfail and powerwait. These entries are invoked (if the run-levels permit) before any further processing takes place. In this way init can perform various cleanup and recording functions during the powerdown of the operating system. Note that in the SINGLE-USER state only powerfail and powerwait entries are executed.

When init is requested to change run-levels (via telinit), init sends the warning signal (SIGTERM) to all processes that are undefined in the target run-level. init waits 5 seconds before forcibly terminating these processes via the kill signal (SIGKILL).

Telinit

Telinit, which is linked to /etc/init, is used to direct the actions of init. It takes a one-character argument and signals init via the kill system call to perform the appropriate action. The following arguments serve as directives to init.

- 0–6 tells init to place the system in one of the run-levels 0–6.
- a,b,c tells init to process only those /etc/inittab file entries having the a, b or c run-level set. These are pseudo-states, which may be defined to run certain commands, but which do not cause the current run-level to change.
- Q,q tells init to re-examine the /etc/inittab file.
- S,S tells init to enter the single user environment. When this level change is effected, the virtual system teletype, /dev/console, is changed to the terminal from which the command was executed.
FILES

/etc/inittab
/etc/utmp
/etc/wtmp
/etc/iocall.syscon
/dev/console
/dev/contty

SEE ALSO

getty(1M), inittab(4), kill(2), login(1), sh(1), termio(7), utmp(4), who(1).

DIAGNOSTICS

If init finds that it is respawning an entry from /etc/inittab more than 10 times in 2 minutes, it will assume that there is an error in the command string in the entry, and generate an error message on the system console. It will then refuse to respawn this entry until either 5 minutes has elapsed or it receives a signal from a user-spawned init (telinit). This prevents init from eating up system resources when someone makes a typographical error in the inittab file or a program is removed that is referenced in the inittab.

WARNINGS

Telinit can be run only by someone who is super-user or a member of group sys.

BUGS

Attempting to relink /dev/console with /dev/contty by typing a delete on the system console does not work.
NAME
install – install binaries

SYNOPSIS
install [-c ] [-m mode ] [-o owner ] [-g group ] [-s ] binary destination

DESCRIPTION
Binary is moved (or copied if -c is specified) to destination. If destination already
exists, it is removed before binary is moved. If the destination is a directory then
binary is moved into the destination directory with its original file-name.
The mode for Destination is set to 755; the -m mode option may be used to specify a
different mode.
Destination is changed to owner root; the -o owner option may be used to specify a
different owner.
Destination is changed to group staff; the -g group option may be used to specify a
different group.
If the -s option is specified the binary is stripped after being installed.
Install refuses to move a file onto itself.

SEE ALSO
chgrp(1), chmod(1), cp(1), mv(1), strip(1), chown(1)
NAME

install – install commands

SYNOPSIS


DESCRIPTION

The install command is most commonly used in "makefiles" [See make(1)] to install a file (updated target file) in a specific place within a file system. Each file is installed by copying it into the appropriate directory, thereby retaining the mode and owner of the original command. The program prints messages telling the user exactly what files it is replacing or creating and where they are going.

If no options or directories (dirx ...) are given, install will search 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. When the first occurrence is found, install issues a message saying that it is overwriting that file with file, and proceeds to do so. If the file is not found, the program states this and exits without further action.

If one or more directories (dirx ...) are specified after file, those directories will be searched before the directories specified in the default list.

The meanings of the options are:

-c dira Installs a new command (file) in the directory specified by dira, only if it is not found. If it is found, install issues a message saying that the file already exists, and exits without overwriting it. May be used alone or with the -s option.

-f dirb Forces file to be installed in given directory, whether or not one already exists. If the file being installed does not already exist, the mode and owner of the new file will be set to 755 and bin, respectively. If the file already exists, the mode and owner will be that of the already existing file. May be used alone or with the -o or -s options.

-i Ignores default directory list, searching only through the given directories (dirx ...). May be used alone or with any other options except -c and -f.

-n dirc If file is not found in any of the searched directories, it is put in the directory specified in dirc. The mode and owner of the new file will be set to 755 and bin, respectively. May be used alone or with any other options except -c and -f.

-m mode The mode of the new file is set to mode. Superuser only.

-u user The owner of the new file is set to user. Superuser only.

-g group The group id of the new file is set to group. Superuser only.

-o If file is found, this option saves the "found" file by copying it to OLDfile in the directory in which it was found. This option is useful when installing a frequently used file such as /bin/sh or /etc/getty, where the existing file cannot be removed. May be used alone or with any other options except -c.

-s Suppresses printing of messages other than error messages. May be used alone or with any other options.

SEE ALSO

make(1).
NAME

installpkg – install all software distribution tapes

SYNOPSIS

/etc/installpkg

DESCRIPTION

installpkg spools forward to the second file on the distribution tape, copies this file from tape to /tmp/do_install, executes /tmp/do_install, and finally rewinds the tape.

If the environment variable $DEBUG_INSTALLPKG is set to debug, an editing session is opened on the specific installation script on the tape being read prior to execution.
NAME

ipcrm – remove a message queue, semaphore set or shared memory id

SYNOPSIS

ipcrm [ options ]

DESCRIPTION

ipcrm will remove one or more specified messages, semaphore or shared memory identifiers. The identifiers are specified by the following options:

-\texttt{-q msqid} removes the message queue identifier \texttt{msqid} from the system and destroys the message queue and data structure associated with it.

-\texttt{-m shmidx} removes the shared memory identifier \texttt{shmidx} from the system. The shared memory segment and data structure associated with it are destroyed after the last detach.

-\texttt{-s semidx} removes the semaphore identifier \texttt{semidx} from the system and destroys the set of semaphores and data structure associated with it.

-\texttt{-Q msgkey} removes the message queue identifier, created with key \texttt{msgkey}, from the system and destroys the message queue and data structure associated with it.

-\texttt{-M shmkey} removes the shared memory identifier, created with key \texttt{shmkey}, from the system. The shared memory segment and data structure associated with it are destroyed after the last detach.

-\texttt{-S semkey} removes the semaphore identifier, created with key \texttt{semkey}, from the system and destroys the set of semaphores and data structure associated with it.

The details of the removes are described in msgctl(2), shmctl(2), and semctl(2). The identifiers and keys may be found by using \texttt{ipcs}(1).

SEE ALSO

ipcs(1),
msgctl(2), msgget(2), msgop(2), semctl(2), semget(2), semop(2), shmct1(2), shmget(2), shmpop(2).
NAME

ipcs - report inter-process communication facilities status

SYNOPSIS

ipcs [ options ]

DESCRIPTION

ipcs prints certain information about active inter-process communication facilities. Without options, information is printed in short format for message queues, shared memory, and semaphores that are currently active in the system. Otherwise, the information that is displayed is controlled by the following options:

-q  Print information about active message queues.
-m  Print information about active shared memory segments.
-s  Print information about active semaphores.

If any of the options -q, -m, or -s are specified, information about only those indicated will be printed. If none of these three are specified, information about all three will be printed subject to these options:

-b  Print biggest allowable size information. (Maximum number of bytes in messages on queue for message queues, size of segments for shared memory, and number of semaphores in each set for semaphores.) See below for meaning of columns in a listing.
-c  Print creator's login name and group name. See below.
-o  Print information on outstanding usage. (Number of messages on queue and total number of bytes in messages on queue for message queues and number of processes attached to shared memory segments.)
-p  Print process number information. (Process ID of last process to send a message and process ID of last process to receive a message on message queues and process ID of creating process and process ID of last process to attach or detach on shared memory segments) See below.
-t  Print time information. (Time of the last control operation that changed the access permissions for all facilities. Time of last msgsnd and last msgrcv on message queues, last shmat and last shmdt on shared memory, last semop(2) on semaphores.) See below.

-a  Use all print options. (This is a shorthand notation for -b, -c, -o, -p, and -t.)
-C corefile
   Use the file corefile in place of /dev/kmem.
-N namelist
   The argument will be taken as the name of an alternate namelist (unix is the default).

The column headings and the meaning of the columns in an ipcs listing are given below; the letters in parentheses indicate the options that cause the corresponding heading to appear; all means that the heading always appears. Note that these options only determine what information is provided for each facility; they do not
determine which facilities will be listed.

T (all)  Type of the facility:
         q  message queue;
         m  shared memory segment;
         s  semaphore.

ID (all)  The identifier for the facility entry.

KEY (all)  The key used as an argument to msgget, semget, or shmat to create
            the facility entry. (Note: The key of a shared memory segment is
            changed to IPC_PRIVATE when the segment has been removed
            until all processes attached to the segment detach it.)

MODE (all)  The facility access modes and flags: The mode consists of 11 char-
            acters that are interpreted as follows:
            The first two characters are:
            R  if a process is waiting on a msgrecv;
            S  if a process is waiting on a msgsnd;
            D  if the associated shared memory segment has been
                removed. It will disappear 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 executed;
                –  if the corresponding special flag is not set.
            The next 9 characters are interpreted as three sets of three bits each.
            The first set refers to the owner's permissions; the next to permis-
            sions 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 group name of the owner of 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.

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 process ID of the last process to send a message to the associ-
              ated queue.

LRPID (a,p)  The process ID of the last process to receive a message from the
            associated queue.

STIME (a,t)  The time the last message was sent to the associated queue.

RTIME (a,t)  The time the last message was received from the associated queue.

CTIME (a,t)  The time when the associated entry was created or changed.

NATTCCH (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 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 sema-
            phor entry.
OTIME (a,t)  The time the last semaphore operation was completed on the set
            associated with the semaphore entry.

FILES
/unix  system namelist
/dev/kmem  memory
/etc/passwd  user names
/etc/group  group names

SEE ALSO
msgop(2), semop(2), shmop(2).

BUGS
Things can change while ipcs is running; the picture it gives is only a close approxi-
    mation to reality.
JOIN(1)

NAME

join – relational database operator

SYNOPSIS

join [ options ] file1 file2

DESCRIPTION

join forms, on the standard output, a join of the two relations specified by the lines of file1 and file2. If file1 is –, the standard input is used.

File1 and file2 must be sorted in increasing ASCII collating sequence on the fields on which they are to be joined, normally the first in each line [see sort(1)]. There is one line in the output for each pair of lines in file1 and file2 that have identical join fields. The output line normally consists of the common field, then the rest of the line from file1, then the rest of the line from file2.

The default input field separators are blank, tab, or new-line. In this case, multiple separators count as one field separator, and leading separators are ignored. The default output field separator is a blank.

Some of the below options use the argument n. This argument should be a 1 or a 2 referring to either file1 or file2, respectively. The following options are recognized:

- an In addition to the normal output, produce a line for each unpairable line in file n, where n is 1 or 2.

- es Replace empty output fields by string s.

- jn m Join on the mth field of file n. If n is missing, use the mth field in each file. Fields are numbered starting with 1.

- o list Each output line comprises the fields specified in list, each element of which has the form n,m, where n is a file number and m is a field number. The common field is not printed unless specifically requested.

- tc Use character c as a separator (tab character). Every appearance of c in a line is significant. The character c is used as the field separator for both input and output.

EXAMPLE

The following command line will join the password file and the group file, matching on the numeric group ID, and outputting the login name, the group name and the login directory. It is assumed that the files have been sorted in ASCII collating sequence on the group ID fields.

    join -j1 4 -j2 3 -o 1.1 2.1 1.6 -t: /etc/passwd /etc/group

SEE ALSO

awk(1), comm(1), sort(1), uniq(1).

BUGS

With default field separation, the collating sequence is that of sort –b; with –t, the sequence is that of a plain sort.

The conventions of join, sort, comm, uniq and awk(1) are wildly incongruous.

Filenames that are numeric may cause conflict when the -o option is used right before listing filenames.
NAME

kill – terminate a process

SYNOPSIS

kill [-signo] PID ...

DESCRIPTION

kill sends signal 15 (terminate) to the specified processes. This will normally kill processes that do not catch or ignore the signal. The process number of each asynchronous process started with & is reported by the shell (unless more than one process is started in a pipeline, in which case the number of the last process in the pipeline is reported). Process numbers can also be found by using ps(1).

The details of the kill are described in kill(2). For example, if process number 0 is specified, all processes in the process group are signaled.

The killed process must belong to the current user unless he is the super-user.

If a signal number preceded by – is given as first argument, that signal is sent instead of terminate (see signal(2)). In particular “kill -9 …” is a sure kill.

SEE ALSO

ps(1), sh(1), kill(2), signal(2).
NAME
killall – kill all active processes

SYNOPSIS
/etc/killall [ signal ]

DESCRIPTION
killall is used by /etc/shutdown to kill all active processes not directly related to the shutdown procedure.

killall terminates all processes with open files so that the mounted file systems will be unbusied and can be unmounted.

killall sends signal (see kill(1)) to all processes not belonging to the above group of exclusions. If no signal is specified, a default of 9 is used.

FILES
/etc/shutdown

SEE ALSO
fuser(1M), kill(1), ps(1), shutdown(1M), signal(2).

WARNINGS
The killall command can only be run by the super-user.
NAME

labelit – provide labels for file systems

SYNOPSIS

/etc/labelit special [ fsname volume [ –n ] ]

DESCRIPTION

labelit can be used to provide labels for unmounted disk file systems or file systems being copied to tape. The –n option provides for initial labeling only (this destroys previous contents).

With the optional arguments omitted, labelit prints current label values.

The special name should be the physical disk section (e.g., /dev/dsk/c0d0s6), or the cartridge tape (e.g., /dev/SA/ctape1). The device may not be on a remote machine.

The fsname argument represents the mounted name (e.g., root, u1, etc.) of the file system.

Volume may be used to equate an internal name to a volume name applied externally to the disk pack, diskette or tape.

For file systems on disk, fsname and volume are recorded in the superblock.

SEE ALSO

fs(4), sh(1).
NAME

last – indicate last logins of users and teletypes

SYNOPSIS

last [−N][ name ... ][ tty ... ]

DESCRIPTION

last will look back in the wtmp file which records all logins and logouts for information about a user, a teletype or any group of users and teletypes. Arguments specify names of users or teletypes of interest. Names of teletypes may be given fully or abbreviated. For example ‘last 0’ is the same as ‘last tty0’. If multiple arguments are given, the information which applies to any of the arguments is printed. For example ‘last root console’ would list all of “root’s” sessions as well as all sessions on the console terminal. last will print the sessions of the specified users and teletypes, most recent first, indicating the times at which the session began, the duration of the session, and the teletype which the session took place on. If the session is still continuing or was cut short by a reboot, last so indicates.

The pseudo-user reboot logs in at reboots of the system, thus

    last reboot

will give an indication of mean time between reboot.

last with no arguments prints a record of all logins and logouts, in reverse order. The −N option limits the report to N lines.

If last is interrupted, it indicates how far the search has progressed in wtmp. If interrupted with a quit signal (generated by a control-\) last indicates how far the search has progressed so far, and the search continues.

FILES

/etc/wtmp       login data base

SEE ALSO

wtmp(5), ac(8), lastcomm(1)

AUTHOR

Howard Katseff
NAME
lastcomm – show last commands executed in reverse order

SYNOPSIS
lastcomm [ command name ] ... [user name] ... [terminal name] ...

DESCRIPTION
lastcomm gives information on previously executed commands. With no arguments, lastcomm prints information about all the commands recorded during the current accounting file's lifetime. If called with arguments, only accounting entries with a matching command name, user name, or terminal name are printed. So, for example, lastcomm a.out root ttyd0 would produce a listing of all the executions of commands named a.out by user root on the terminal ttyd0.

For each process entry, the following are printed.
   The name of the user who ran the process.
   Flags, as accumulated by the accounting facilities in the system.
   The command name under which the process was called.
   The amount of cpu time used by the process (in seconds).
   The time the process exited.

The flags are encoded as follows: "S" indicates the command was executed by the super-user, "F" indicates the command ran after a fork, but without a following exec, "D" indicates the command terminated with the generation of a core file, and "X" indicates the command was terminated with a signal.

FILES
/usr/adm/pacct

SEE ALSO
last(1), sigvec(2), acct(8)
NAME

Id – link editor for common object files

SYNOPSIS

Id [options] filename

DESCRIPTION

The ld command combines several object files into one, performs relocation, resolves external symbols, and supports symbol table information for debugging. In the simplest case, the names of several object programs are given, and ld combines the objects, producing an object module that can be executed. The output of ld is left in a.out. By default this file is executable if no errors occurred during the load. If any input file, filename, is not an object file, ld assumes it is an archive library.

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. The library (archive) symbol table [see ar(4)] is searched sequentially with as many passes as are necessary to resolve external references which can be satisfied by library members. Thus, the ordering of library members is functionally unimportant, unless there exist multiple library members defining the same external symbol.

The following options are recognized by ld:

-B hhhhhhh
    Generate an a.out file with the bss address at hhhhhhh.

-D hhhhhhh
    Generate an a.out file with the data address at hhhhhhh.

-esym
    Set the default entry point address for the output file to be that of sym.

-L
    Do not search for -l libraries in /lib or /usr/lib.

-Ldir
    Search for -l libraries in dir.

-m
    Generate a simple load map on standarad output.

-n
    Generate NMAGIC file type.

-o filename
    Place the output into filename.

-opct
    Insert code into the program that produces a count of all the FPU ops executed by the program, dynamically.

-p
    Generate code to profile the source file during execution.

-r
    Produce a relocatable output file.

-s
    Strip line numbers and symbol table information from the output object file.

-T hhhhhhh
    Generate an a.out file with the text address at hhhhhhh.

-t
    Turn off warnings about multiply defined symbols that are not the same size.

-uname
    Enter name as an undefined symbol in the symbol table.

-V
    Print version information.
-yname
    Trace name and print out all uses and definitions.

FILES
LIBDIR/libx.a        libraries
LLIBDIR/libx.a       libraries
a.out                output file
LIBDIR               usually /lib
LLIBDIR              usually /usr/lib

SEE ALSO
as(1), cc(1), fc(1), exit(2), end(3C), a.out(4), ar(4).
Programmer's Guide.
NAME
leave – remind you when you have to leave

SYNOPSIS
leave [ [+hhmm]

DESCRIPTION
leave waits until the specified time, then reminds you that you have to leave. You are reminded 5 minutes and 1 minute before the actual time, at the time, and every minute thereafter. When you log off, leave exits just before it would have printed the next message.

The time of day is in the form hhmm where hh is a time in hours (on a 12 or 24 hour clock). All times are converted to a 12 hour clock, and assumed to be in the next 12 hours.

If the time is preceded by ‘+’, the alarm will go off in hours and minutes from the current time.

If no argument is given, leave prompts with "When do you have to leave?". A reply of newline causes leave to exit, otherwise the reply is assumed to be a time. This form is suitable for inclusion in a .login or .profile.

leave ignores interrupts, quits, and terminates. To get rid of it you should either log off or use “kill –9” giving its process id.

SEE ALSO
calendar(1)
**NAME**

lex – generate programs for simple lexical tasks

**SYNOPSIS**

```plaintext
lex [-rctvn] [file]...
```

**DESCRIPTION**

The `lex` command generates programs to be used in simple lexical analysis of text.

The input files (standard input default) contain strings and expressions to be searched for, and C text to be executed when strings are found.

A file `lex.yy.c` is generated which, when loaded with the library, copies the input to the output except when a string specified in the file is found; then the corresponding program text is executed. The actual string matched is left in `yytext`, an external character array. Matching is done in order of the strings in the file. The strings may contain square brackets to indicate character classes, as in `[abx-z]` to indicate `a`, `b`, `x`, `y`, and `z`; and the operators `*`, `+`, and `?` mean respectively any non-negative number of, any positive number of, and either zero or one occurrence of, the previous character or character class. The character `.` is the class of all ASCII characters except new-line.

Parentheses for grouping and vertical bar for alternation are also supported. The notation `r(d,e)` in a rule indicates between `d` and `e` instances of regular expression `r`. It has higher precedence than `,` but lower than `*`, `?`, `+`, and concatenation. Thus `[a-zA-Z]+` matches a string of letters. The character `^` at the beginning of an expression permits a successful match only immediately after a new-line, and the character `$` at the end of an expression requires a trailing new-line. The character `/` in an expression indicates trailing context; only the part of the expression up to the slash is returned in `yytext`, but the remainder of the expression must follow in the input stream. An operator character may be used as an ordinary symbol if it is within " symbols or preceded by \\.

Three subroutines defined as macros are expected: `input()` to read a character; `unput()` to replace a character read; and `output()` to place an output character. They are defined in terms of the standard streams, but you can override them. The program generated is named `yylex()`, and the library contains a `main()` which calls it. The action `REJECT` on the right side of the rule causes this match to be rejected and the next suitable match executed; the function `yymore()` accumulates additional characters into the same `yytext`; and the function `yyless()` pushes back the portion of the string matched beginning at `p`, which should be between `yytext` and `yytext+yylen`. The macros `input` and `output` use files `yyn` and `yyout` to read from and write to, respectively.

Any line beginning with a blank is assumed to contain only C text and is copied; if it precedes `%%` it is copied into the external definition area of the `lex.yy.c` file. All rules should follow a `%%`, as in YACC. Lines preceding `%%` which begin with a non-blank character define the string on the left to be the remainder of the line; it can be called out later by surrounding it with `{}`. Note that curly brackets do not imply parentheses; only string substitution is done.

**EXAMPLE**

```
D       [0-9]
%%
if      printf("IF statement\n");
[a-z]+  printf("tag, value %s\n",yytext);
0[D]+   printf("octal number %s\n",yytext);
[D]+    printf("decimal number %s\n",yytext);
"++"    printf("unary op\n");
"+"     printf("binary op\n");
```
"/\*"   skipcommnts();
%
skipcommnts()
{
   for (; ;)
   {
      while (input() != '*')
         ;
      if (input() != '/')
         unput(yytext[yylen-1]);
      else
         return;
   }
}

The external names generated by lex all begin with the prefix yy or YY.
The flags must appear before any files. The flag -r indicates RATFOR actions, -c indicates C actions and is the default, -t causes the lex.yy.c program to be written instead to standard output, -v provides a one-line summary of statistics, -n will not print out the -v summary. Multiple files are treated as a single file. If no files are specified, standard input is used.

Certain table sizes for the resulting finite state machine can be set in the definitions section:

%p n   number of positions is n (default 2500)
%n n   number of states is n (500)
%e n   number of parse tree nodes is n (1000)
%a n   number of transitions is n (2000)
%k n   number of packed character classes is n (1000)
%o n   size of output array is n (3000)

The use of one or more of the above automatically implies the -v option, unless the -n option is used.

SEE ALSO

yacc(1).

BUGS

The -r option is not yet fully operational.
NAME
line – read one line

SYNOPSIS
line

DESCRIPTION
line copies one line (up to a new-line) from the standard input and writes it on the standard output. It returns an exit code of 1 on EOF and always prints at least a new-line. It is often used within shell files to read from the user’s terminal.

SEE ALSO
sh(1), read(2).
NAME

link, unlink – link and unlink files and directories

SYNOPSIS

/etc/link file1 file2
/etc/unlink file

DESCRIPTION

The `link` command is used to create a file name that points to another file. Linked files and directories can be removed by the `unlink` command; however, it is strongly recommended that the `rm(1)` and `rmdir(1)` commands be used instead of the `unlink` command.

The only difference between `ln(1)` and `link/unlink` is that the latter do exactly what they are told to do, abandoning all error checking. This is because they directly invoke the `link(2)` and `unlink(2)` system calls.

SEE ALSO

`link(2)`, `unlink(2)`, `rm(1)`. 

WARNINGS

These commands can only be run by the super-user.
NAME
localize – make names local to an object file

SYNOPSIS
localize filename [ globalnames... ]

DESCRIPTION
The localize command makes all but a specified list of names local to the object file. filename is both read and written. globalnames are external entry points in the .o file that are retained. This command is particularly useful when an object file is created with the ld -r.

SEE ALSO
ld(1)
NAME
logger – make entries in the system log

SYNOPSIS
logger [ -t tag ] [ -p pri ] [ -i ] [ -f file ] [ message ... ]

ARGUMENTS
- t tag Mark every line in the log with the specified tag.
- p pri Enter the message with the specified priority. The priority may be
  specified numerically or as a “facility.level” pair. For example, “-p
  local3.info” logs the message(s) as informational level in the local3 faci-
  lity. The default is “user.notice.”
- i Log the process id of the logger process with each line.
- f file Log the specified file.
message The message to log; if not specified, the -f file or standard input is
logged.

DESCRIPTION
Logger provides a program interface to the syslog(3) system log module.
A message can be given on the command line, which is logged immediately, or a file
is read and each line is logged.

EXAMPLES
logger System rebooted
logger -p local0.notice -t HOSTIDM -f /dev/idmc

SEE ALSO
syslog(3), syslogd(8)
NAME

clogin – sign on

SYNOPSIS

login [ name [ env-var … ]]

DESCRIPTION

The login command is used at the beginning of each terminal session and allows you to identify yourself to the system. It may be invoked as a command or by the system when a connection is first established. Also, it is invoked by the system when a previous user has terminated the initial shell by typing a \texttt{ctrld} to indicate an “end-of-file.” (See \textit{How to Get Started} at the beginning of this volume for instructions on how to dial up initially.)

If login is invoked as a command it must replace the initial command interpreter. This is accomplished by typing: \texttt{exec login} from the initial shell. login asks for your user name (if not supplied as an argument), and, if appropriate, your password. Echoing is turned off (where possible) during the typing of your password, so it will not appear on the written record of the session.

At some installations, an option may be invoked that will require you to enter a second “dialup” password. This will occur only for dial-up connections, and will be prompted by the message “dialup password?”. Both passwords are required for a successful login.

If you do not complete the login successfully within a certain period of time (e.g., one minute), you are likely to be silently disconnected.

After a successful login, accounting files are updated, the procedure \texttt{/etc/profile} is performed, the message-of-the-day, if any, is printed, the user-ID, the group-ID, the working directory, and the command interpreter (usually \texttt{sh(1)}) is initialized, and the file \texttt{.profile} in the working directory is executed, if it exists. These specifications are found in the \texttt{/etc/passwd} file entry for the user. The name of the command interpreter is – followed by the last component of the interpreter’s path name (i.e., –\texttt{sh}). If this field in the password file is empty, then the default command interpreter, \texttt{/bin/sh} is used. If this field is “\texttt{*}”, then the named directory becomes the root directory, the starting point for path searches for path names beginning with a \texttt{/}. At that point login is re-executed at the new level which must have its own root structure, including \texttt{/etc/login} and \texttt{/etc/passwd}.

The basic \texttt{environment} is initialized to:

\begin{verbatim}
HOME=your-login-directory
PATH=/bin:/usr/bin
SHELL=last-field-of-passwd-entry
MAIL=/usr/mail/your-login-name
TZ=timezone-specification
\end{verbatim}

The environment may be expanded or modified by supplying additional arguments to login, either at execution time or when login requests your login name. The arguments may take either the form \texttt{xxx} or \texttt{xxx=yyy}. Arguments without an equal sign are placed in the environment as \texttt{Ln=xxx} where \texttt{n} is a number starting at \texttt{0} and is incremented each time a new variable name is required. Variables containing an \texttt{=} are placed into the environment without modification. If they already appear in the environment, then they replace the older value. There are two exceptions. The variables PATH and SHELL cannot be changed. This prevents people, logging into restricted shell environments, from spawning secondary shells which are not restricted. Both login and getty understand simple single-character quoting conventions. Typing...
a backslash in front of a character quotes it and allows the inclusion of such things as spaces and tabs.

FILES

/etc/utmp           accounting
/etc/wtmp           accounting
/usr/mail/your-name mailbox for user your-name
/etc/motd           message-of-the-day
/etc/passwd         password file
/etc/profile        system profile
.profile              user's login profile
/usr/adm/lastlog    records last login time

SEE ALSO

mail(1), newgrp(1), sh(1), su(1M), passwd(4), profile(4), environ(5).

DIAGNOSTICS

login incorrect if the user name or the password cannot be matched.
No shell, cannot open password file, or no directory: consult a UNIX system program-
ing counselor.
No utmp entry. You must exec "login" from the lowest level "sh" if you attempted to exe-
cute login as a command without using the shell's exec internal command or from
other than the initial shell.

NOTE

After a successful login, accounting files are updated and the user is informed of the exis-
tence of mail. The message of the day is printed, as is the time of the last login.
Both are suppressed if the user's home directory contains a "hushlogin" file; this is mostly used to make life easier for non-human users, such as uucp(1O).
NAME
    logname – get login name
SYNOPSIS
    logname
DESCRIPTION
    logname returns the contents of the environment variable $LOGNAME, which is set when a user logs into the system.
FILES
    /etc/profile
SEE ALSO
    env(1), login(1). logname(3X), environ(5).
User Commands

LOOK(1)

NAME

look – find lines in a sorted list

SYNOPSIS

look [ -df ] string [ file ]

DESCRIPTION

Look consults a sorted file and prints all lines that begin with string. It uses binary search.

The options d and f affect comparisons as in sort(1):

d  ‘Dictionary’ order: only letters, digits, tabs and blanks participate in comparisons.

f  Fold. Upper case letters compare equal to lower case.

If no file is specified, /usr/dict/words is assumed with collating sequence –df.

FILES

/usr/dict/words

SEE ALSO

sort(1), grep(1)
NAME

lp, cancel – send/cancel requests to an LP line printer

SYNOPSIS

cancel [ids] [printers]

DESCRIPTION

lp arranges for the named files and associated information (collectively called a request) to be printed by a line printer. If no file names are mentioned, the standard input is assumed. The file name – stands for the standard input and may be supplied on the command line in conjunction with named files. The order in which files appear is the same order in which they will be printed.

lp associates a unique id with each request and prints it on the standard output. This id can be used later to cancel (see cancel) or find the status (see lpstat(1)) of the request.

The following options to lp may appear in any order and may be intermixed with file names:

-\(c\)

Make copies of the files to be printed immediately when lp is invoked. Normally, files will not be copied, but will be linked whenever possible. If the \(-c\) option is not given, then the user should be careful not to remove any of the files before the request has been printed in its entirety. It should also be noted that in the absence of the \(-c\) option, any changes made to the named files after the request is made but before it is printed will be reflected in the printed output.

-\(ddest\)

Choose dest as the printer or class of printers that is to do the printing. If dest is a printer, then the request will be printed only on that specific printer. If dest is a class of printers, then the request will be printed on the first available printer that is a member of the class. Under certain conditions (printer unavailability, file space limitation, etc.), requests for specific destinations may not be accepted (see accept(1M) and lpstat(1)). By default, dest is taken from the environment variable LPDEST (if it is set). Otherwise, a default destination (if one exists) for the computer system is used. Destination names vary between systems (see lpstat(1)).

-\(m\)

Send mail (see mail(1)) after the files have been printed. By default, no mail is sent upon normal completion of the print request.

-\(nnumber\)

Print number copies (default of 1) of the output.

-\(o\) \(option\)

Specify printer-dependent or class-dependent options. Several such options may be collected by specifying the \(-o\) keyletter more than once. For more information about what is valid for options, see Models in ladmin(1M).

-\(s\)

Suppress messages from lp(1) such as "request id is ...".

-\(ttitle\)

Print title on the banner page of the output.

-\(w\)

Write a message on the user's terminal after the files have been printed. If the user is not logged in, then mail will be sent instead.

Cancel cancels line printer requests that were made by the lp(1) command. The command line arguments may be either request ids (as returned by lp(1)) or printer names (for a complete list, use lpstat(1)). Specifying a request id cancels the associated request even if it is currently printing. Specifying a printer cancels the request which is currently printing on that printer. In either case, the cancellation of a request that is currently printing frees the printer to print its next available request.
FILES

/usr/spool/lp/*

SEE ALSO

accept(1M), enable(1), lpadmin(1M), lpsched(1M), lpstat(1), mail(1).
NAME
lpadmin – configure the LP spooling system

SYNOPSIS
/usr/lib/lpadmin -p printer [ options ]
/usr/lib/lpadmin -x dest
/usr/lib/lpadmin -d[dest]

DESCRIPTION
lpadmin configures line printer (LP) spooling systems to describe printers, classes and
devices. It is used to add and remove destinations, change membership in classes,
change devices for printers, change printer interface programs and to change the sys-
tem default destination. lpadmin may not be used when the LP scheduler,
lpsched(1M), is running, except where noted below.

Exactly one of the -p, -d or -x options must be present for every legal invocation of
lpadmin.

-pprinter names a printer to which all of the options below refer. If printer does not
exist then it will be created.

-xdest removes destination dest from the LP system. If dest is a printer and is
the only member of a class, then the class will be deleted, too. No other
options are allowed with -x.

-d[dest] makes dest, an existing destination, the new system default destination.
If dest is not supplied, then there is no system default destination. This
option may be used when lpsched(1M) is running. No other options are
allowed with -d.

The following options are only useful with -p and may appear in any order. For ease
of discussion, the printer will be referred to as P below.

-cclass inserts printer P into the specified class. Class will be created if it does
not already exist.

-eprinter copies an existing printer’s interface program to be the new interface
program for P.

-h indicates that the device associated with P is hardwired. This option is
assumed when adding a new printer unless the -I option is supplied.

-iinterface establishes a new interface program for P. Interface is the path name of
the new program.

-I indicates that the device associated with P is a login terminal. The LP
scheduler, lpsched, disables all login terminals automatically each time it
is started. Before re-enabling P, its current device should be established
using lpadmin.

-mmodel selects a model interface program for P. Model is one of the model inter-
face names supplied with the LP Spooling Utilities (see Models below).

-xclass removes printer P from the specified class. If P is the last member of the
class, then the class will be removed.

-vedevice associates a new device with printer P. Device is the pathname of a file
that is writable by lp. Note that the same device can be associated with
more than one printer. If only the -p and -v options are supplied, then
lpadmin may be used while the scheduler is running.

Restrictions.

When creating a new printer, the -v option and one of the -e, -I or -m options must
be supplied. Only one of the -e, -I or -m options may be supplied. The -h and -l
keyletters are mutually exclusive. Printer and class names may be no longer than 14 characters and must consist entirely of the characters A-Z, a-z, 0-9 and _ (underscore).

Models.

Model printer interface programs are supplied with the LP Spooling Utilities. They are shell procedures which interface between lpsched and devices. All models reside in the directory /usr/spool/lp/model and may be used as is with lpadmin -m. Copies of model interface programs may also be modified and then associated with printers using lpadmin -i. The following describes two models which may be given on the lp command line using the -o keyletter:

PS A 300-dpi PostScript® laser printer, such as the Apple LaserWriter.
tek4693 The Tektronix 4693D 300-dpi color thermal transfer printer.

EXAMPLES

1. Use the following command to connect a PostScript printer named PS1:
   
   /usr/lib/lpadmin -pPS -v/dev/tty0 -mPS.interface

2. Use the following command to install a Tektronix 4693D color printer:
   
   /usr/lib/lpadmin -pTEK -v/dev/tek -mtek4693

FILES

/usr/spool/lp/*

SEE ALSO

accept(1M), enable(1), lp(1), lpsched(1M), lpstat(1).
NAME
lpsched, lpshut, lpmove – start/stop the LP scheduler and move requests

SYNOPSIS
/usr/lib/lpsched
/usr/lib/lpshut
/usr/lib/lpmove requests dest
/usr/lib/lpmove dest1 dest2

DESCRIPTION
lpsched schedules requests taken by lp(1) for printing on line printers (LP's).

Lpshut shuts down the line printer scheduler. All printers that are printing at the
time lpshut is invoked will stop printing. Requests that were printing at the time a
printer was shut down will be reprinted in their entirety after lpsched is started again.

Lpmove moves requests that were queued by lp(1) between LP destinations. This
command may be used only when lpsched is not running.

The first form of the command moves the named requests to the LP destination, dest.
Requests are request ids as returned by lp(1). The second form moves all requests for
destination dest1 to destination dest2. As a side effect, lp(1) will reject requests for
dest1.

Note that lpmove never checks the acceptance status (see accept(1M)) for the new des-
tination when moving requests.

FILES
/usr/spool/lp/*

SEE ALSO
accept(1M), enable(1), lp(1), ladmin(1M), lpstat(1).
NAME
lpstat – print LP status information

SYNOPSIS
lpstat [ options ]

DESCRIPTION
lpstat prints information about the current status of the LP spooling system.
If no options are given, then lpstat prints the status of all requests made to lp(1) by the
user. Any arguments that are not options are assumed to be request ids (as returned
by lp). lpstat prints the status of such requests. Options may appear in any order and
may be repeated and intermixed with other arguments. Some of the keyletters below
may be followed by an optional list that can be in one of two forms: a list of items
separated from one another by a comma, or a list of items enclosed in double quotes
and separated from one another by a comma and/or one or more spaces. For exam-
ple:

   -u "user1, user2, user3"

The omission of a list following such keyletters causes all information relevant to the
keyletter to be printed, for example:

   lpstat -o

prints the status of all output requests.

-a [ list ]  Print acceptance status (with respect to lp) of destinations for requests. List
            is a list of intermixed printer names and class names.

-c [ list ]  Print class names and their members. List is a list of class names.

-d          Print the system default destination for lp.

-o [ list ]  Print the status of output requests. List is a list of intermixed printer
            names, class names, and request ids.

-p [ list ]  Print the status of printers. List is a list of printer names.

-r          Print the status of the LP request scheduler

-s          Print a status summary, including the system default destination, a list of
            class names and their members, and a list of printers and their associated
            devices.

-t          Print all status information.

-u [ list ]  Print status of output requests for users. List is a list of login names.

-v [ list ]  Print the names of printers and the path names of the devices associated
            with them. List is a list of printer names.

FILES
/usr/spool/lp/*

SEE ALSO
enable(1), lp(1).
NAME
ls – list contents of directory

SYNOPSIS
ls [-RdCxmlnogrtucpFbqisf] [names]

DESCRIPTION
For each directory argument, ls lists the contents of the directory; for each file argument, ls repeats its name and any other information requested. The output is sorted alphabetically by default. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments appear before directories and their contents.

There are three major listing formats. The default format is to list one entry per line, the -C and -x options enable multi-column formats, and the -m option enables stream output format. In order to determine output formats for the -C, -x, and -m options, ls uses an environment variable, COLUMNS, to determine the number of character positions available on one output line. If this variable is not set, the terminfo(4) database is used to determine the number of columns, based on the environment variable TERM. If this information cannot be obtained, 80 columns are assumed.

The ls command has the following options:
-R Recursively list subdirectories encountered.
-a List all entries, including those that begin with a dot (.), which are normally not listed.
-d If an argument is a directory, list only its name (not its contents); often used with -l to get the status of a directory.
-C Multi-column output with entries sorted down the columns.
-x Multi-column output with entries sorted across rather than down the page.
-m Stream output format; files are listed across the page, separated by commas.
-l List in long format, giving mode, number of links, owner, group, size in bytes, and time of last modification for each file (see below). If the file is a special file, the size field will instead contain the major and minor device numbers rather than a size.
-n The same as -l, except that the owner’s UID and group’s GID numbers are printed, rather than the associated character strings.
-o The same as -l, except that the group is not printed.
-g The same as -l, except that the owner is not printed.
-r Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.
-t Sort by time stamp (latest first) instead of by name. The default is the last modification time. (See -n and -c.)
-u Use time of last access instead of last modification for sorting (with the -t option) or printing (with the -l option).
-c Use time of last modification of the i-node (file created, mode changed, etc.) for sorting (-t) or printing (-l).
-p Put a slash (/) after each filename if that file is a directory.
-F Put a slash (/) after each filename if that file is a directory and put an asterisk (*) after each filename if that file is executable.
-b Force printing of non-graphic characters to be in the octal \ddd notation.
-q Force printing of non-graphic characters in file names as the character (?).
-i For each file, print the i-number in the first column of the report.
-s Give size in blocks, including indirect blocks, for each entry.
-f Force each argument to be interpreted as a directory and list the name found in each slot. This option turns off -l, -t, -s, and -r, and turns on -a; the order is the order in which entries appear in the directory.

The mode printed under the -l option consists of ten characters. The first character may be one of the following:

  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 fifo (a.k.a. "named pipe") special file;
  - the entry is an ordinary file.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner’s permissions; the next to permissions of others in the user-group of the file; and the last to all others. Within each set, the three characters indicate permission to read, to write, and to execute the file as a program, respectively. For a directory, "execute" permission is interpreted to mean permission to search the directory for a specified file.

ls -l (the long list) prints its output as follows:

    -rwxrwxrwx 1 smith dev 10876 May 16 9:42 part2
    lrwxrwxrwx 1 paul hum 23568 May 9 17:31 symfoo -> foo

This horizontal configuration provides a good deal of information. Reading from right to left in the first example, you see that the current directory holds one file, named "part2." Next, the last time that file’s contents were modified was 9:42 A.M. on May 16. The file is moderately sized, containing 10,876 characters, or bytes. The owner of the file, or the user, belongs to the group "dev" (perhaps indicating "development"), and his or her login name is "smith." The number, in this case "1," indicates the number of links to file "part2." Finally, the row of dash and letters tell you that user, group, and others have permissions to read, write, execute "part2." The second example shows how a symbolic link is displayed. The file symfoo is a symbolic link to the file foo.

The execute (x) symbol here occupies the third position of the three-character sequence. A – in the third position would have indicated a denial of execution permissions.

The permissions are indicated as follows:

  r the file is readable
  w the file is writable
  x the file is executable
  - the indicated permission is not granted
  l mandatory locking will occur during access (the set-group-ID bit is on and the group execution bit is off)
  s the set-user-ID or set-group-ID bit is on, and the corresponding user or group execution bit is also on
  S undefined bit-state (the set-user-ID bit is on and the user execution bit if off)
the 1000 (octal) bit, or sticky bit, is on (see chmod(1)), and execution is on
the 1000 bit is turned on, and execution is off (undefined bit-state)

For user and group permissions, the third position is sometimes occupied by a character other than x or -. s also may occupy this position, referring to the state of the set-ID bit, whether it be the user's or the group's. The ability to assume the same ID as the user during execution is, for example, used during login when you begin as root but need to assume the identity of the user stated at "login."

In the case of the sequence of group permissions, I may occupy the third position. I refers to mandatory file and record locking. This permission describes a file's ability to allow other files to lock its reading or writing permissions during access.

For others permissions, the third position may be occupied by t or T. These refer to the state of the sticky bit and execution permissions.

EXAMPLES

The first set of examples refers to permissions:

```
-rwxr--r--
```

This describes a file that is readable, writable, and executable by the user and readable by the group and others.

```
-rwsr-xr-x
```

The second example describes a file that is readable, writable, and executable by the user, readable and executable by the group and others, and allows its user-ID to be assumed, during execution, by the user presently executing it.

```
-rw-rw-l--
```

This example describes a file that is readable and writable only by the user and the group and can be locked during access.

```
ls -a
```

This command will print the names of all files in the current directory, including those that begin with a dot (.), which normally do not print.

```
ls -al
```

This command will provide you with quite a bit of information including all files, including non-printing ones (a), the i-number—the memory address of the i-node associated with the file—printed in the left-hand column (i); the size (in blocks) of the files, printed in the column to the right of the i-numbers (s); finally, the report is displayed in the numeric version of the long list, printing the UID (instead of user name) and GID (instead of group name) numbers associated with the files.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks, is printed.

FILES

```
/etc/passwd
/etc/group
/usr/lib/terminfo/*/user IDs for ls -l and ls -o
group IDs for ls -l and ls -g
terminal information database
```

SEE ALSO

chmod(1), find(1).

BUGS

Unprintable characters in file names may confuse the columnar output options.
NAME

m4 - macro processor

SYNOPSIS

m4 [ options ] [ files ]

DESCRIPTION

The m4 command is a macro processor intended as a front end for Ratfor, C, and other languages. Each of the argument files is processed in order; if there are no files, or if a file name is -, the standard input is read. The processed text is written on the standard output.

The options and their effects are as follows:

-e Operate interactively. Interrupts are ignored and the output is unbuffered.
-s Enable line sync output for the C preprocessor (#line ...)
-Bint Change the size of the push-back and argument collection buffers from the default of 4,096.
-Hint Change the size of the symbol table hash array from the default of 199. The size should be prime.
-Sint Change the size of the call stack from the default of 100 slots. Macros take three slots, and non-macro arguments take one.
-Tint Change the size of the token buffer from the default of 512 bytes.

To be effective, these flags must appear before any file names and before any -D or -U flags:

-Dname[=val]
Defines name to val or to null in val's absence.
-Uname undefines name.

Macro calls have the form:

name(arg1,arg2, ..., argn)

The ( must immediately follow the name of the macro. If the name of a defined macro is not followed by a (, it is deemed to be a call of that macro with no arguments. Potential macro names consist of alphabetic letters, digits, and underscore _, where the first character is not a digit.

Leading unquoted blanks, tabs, and new-lines are ignored while collecting arguments. Left and right single quotes are used to quote strings. The value of a quoted string is the string stripped of the quotes.

When a macro name is recognized, its arguments are collected by searching for a matching right parenthesis. If fewer arguments are supplied than are in the macro definition, the trailing arguments are taken to be null. Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses which happen to turn up within the value of a nested call are as effective as those in the original input text. After argument collection, the value of the macro is pushed back onto the input stream and rescanned.

m4 makes available the following built-in macros. They may be redefined, but once this is done the original meaning is lost. Their values are null unless otherwise stated.

define the second argument is installed as the value of the macro whose name is the first argument. Each occurrence of $n in the replacement text, where $n is a digit, is replaced by the $n-th argument. Argument 0
is the name of the macro; missing arguments are replaced by the null string; $# is replaced by the number of arguments; $* is replaced by a list of all the arguments separated by commas; $@ is like $*, but each argument is quoted (with the current quotes).

**undef**

removes the definition of the macro named in its argument.

**defn**

returns the quoted definition of its argument(s). It is useful for renaming macros, especially built-ins.

**pushdef**

like **define**, but saves any previous definition.

**popdef**

removes current definition of its argument(s), exposing the previous one, if any.

**ifdef**

if the first argument is defined, the value is the second argument, otherwise the third. If there is no third argument, the value is null. The word **unix** is predefined on UNIX system versions of **m4**.

**shift**

returns all but its first argument. The other arguments are quoted and pushed back with commas in between. The quoting nullifies the effect of the extra scan that will subsequently be performed.

**changequote**

change quote symbols to the first and second arguments. The symbols may be up to five characters long. **Changequote** without arguments restores the original values (i.e., ` `).  

**changecom**

change left and right comment markers from the default # and newline. With no arguments, the comment mechanism is effectively disabled. With one argument, the left marker becomes the argument and the right marker becomes newline. With two arguments, both markers are affected. Comment markers may be up to five characters long.

**divert**

**m4** maintains 10 output streams, numbered 0-9. The final output is the concatenation of the streams in numerical order; initially stream 0 is the current stream. The **divert** macro changes the current output stream to its (digit-string) argument. Output diverted to a stream other than 0 through 9 is discarded.

**undivert**

causes immediate output of text from diversions named as arguments, or all diversions if no argument. Text may be undiverted into another diversion. Undiverting discards the diverted text.

**divnum**

returns the value of the current output stream.

**dnl**

reads and discards characters up to and including the next newline.

**ifelse**

has three or more arguments. If the first argument is the same string as the second, then the value is the third argument. If not, and if there are more than four arguments, the process is repeated with arguments 4, 5, 6 and 7. Otherwise, the value is either the fourth string, or, if it is not present, null.

**incr**

returns the value of its argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.

**decr**

returns the value of its argument decremented by 1.

**eval**

evaluates its argument as an arithmetic expression, using 32-bit arithmetic. Operators include +, −, *, /, %, ^ (exponentiation), bitwise &|, ^, and ∼; relational; parentheses. Octal and hex numbers may be specified as in C. The second argument specifies the radix for the result; the default is 10. The third argument may be used to specify
the minimum number of digits in the result.

len
returns the number of characters in its argument.

index
returns the position in its first argument where the second argument begins (zero origin), or -1 if the second argument does not occur.

substr
returns a substring of its first argument. The second argument is a zero origin number selecting the first character; the third argument indicates the length of the substring. A missing third argument is taken to be large enough to extend to the end of the first string.

translit
transliterates the characters in its first argument from the set given by the second argument to the set given by the third. No abbreviations are permitted.

include
returns the contents of the file named in the argument.

sinclude
is identical to include, except that it says nothing if the file is inaccessible.

syscmd
executes the UNIX system command given in the first argument. No value is returned.

sysval
is the return code from the last call to syscmd.

maketemp
fills in a string of XXXXX in its argument with the current process ID.

m4exit
causes immediate exit from m4. Argument 1, if given, is the exit code; the default is 0.

m4wrap
argument 1 will be pushed back at final EOF; example: m4wrap("cleanup()")

errprint
prints its argument on the diagnostic output file.

dumpdef
prints current names and definitions, for the named items, or for all if no arguments are given.

traceon
with no arguments, turns on tracing for all macros (including built-ins). Otherwise, turns on tracing for named macros.

traceoff
turns off trace globally and for any macros specified. Macros specifically traced by traceon can be untraced only by specific calls to traceoff.

SEE ALSO
cc(1), cpp(1).

The m4 Macro Processor in the Support Tools Guide.
NAME
machid: titan, mips, m68000, pdp11, u3b, u3b2, u3b5, vax – get processor type truth value

SYNOPSIS
titan
mips
m68000
pdp11
u3b
u3b2
u3b5
vax

DESCRIPTION
The following commands return a true value (exit code of 0) if you are using a processor that the command name indicates.

titan  True for the Stardent Computer Inc. Stardent 1500/3000 graphics supercomputer.
mips   True for a MIPS M/ series computer.
m68000 True for a Motorola 680x0 series processor.
pdp11  True for a PDP-11/45 or PDP-11/70.
u3b    True for a 3B20 computer.
u3b2   True for a 3B2 computer.
u3b5   True for a 3B5 computer.
vax    True for a VAX-11/750 or VAX-11/780.

The commands that do not apply return a false (non-zero) value. These commands are often used within makefiles (see make(1)) and shell procedures (see sh(1)) to increase portability.

SEE ALSO
sh(1), make(1), test(1), true(1).
NAME

mail, rmail – send mail to users or read mail

SYNOPSIS

Sending mail:

mail [-oswt] persons

rmail [-oswt] persons

Reading mail:

mail [-ehpqr] [-f] [-F persons]

DESCRIPTION

Sending mail:

The command-line arguments that follow affect SENDING mail:

-o suppresses the address optimization facility.

-s suppresses the addition of a <new-line> at the top of the letter being sent. See WARNINGS below.

-w causes a letter to be sent to a remote user without waiting for the completion of the remote transfer program.

-t causes a To: line to be added to the letter, showing the intended recipients.

A person is usually a user name recognized by login(1). When persons are named, mail assumes a message is being sent (except in the case of the /-F option). It reads from the standard input up to an end-of-file (control-d), or until it reads a line consisting of just a period. When either of those signals is received, mail adds the letter to the mailfile for each person. A letter is a message preceded by a postmark. The message is preceded by the sender’s name and a postmark. A postmark consists of one or more ‘From’ lines followed by a blank line (unless the -s argument was used).

If a letter is found to be undeliverable, it is returned to the sender with diagnostics that indicate the location and nature of the failure. If mail is interrupted during input, the file dead.letter is saved to allow editing and resending. dead.letter is recreated every time it is needed, erasing any previous contents.

rmail only permits the sending of mail; uucp(1C) uses rmail as a security precaution.

If the local system has the Basic Networking Utilities installed, mail may be sent to a recipient on a remote system. Prefix person by the system name and exclamation point. A series of system names separated by exclamation points can be used to direct a letter through an extended network.

Reading Mail:

The command-line arguments that follow affect READING mail:

-e causes mail not to be printed. An exit value of 0 is returned if the user has mail; otherwise, an exit value of 1 is returned.

-h causes a window of headers to be displayed rather than the latest message. The display is followed by the ‘?’ prompt.

-p causes all messages to be printed without prompting for disposition.

-q causes mail to terminate after interrupts. Normally an interrupt causes only the termination of the message being printed.

-r causes messages to be printed in first-in, first-out order.
--file causes mail to use file (e.g., mbox) instead of the default mailfile.

--persons
    entered into an empty mailbox, causes all incoming mail to be forwarded to persons.

mail, unless otherwise influenced by command-line arguments, prints a user's mail
messages in last-in, first-out order. For each message, the user is prompted with a ?,
and a line is read from the standard input. The following commands are available to
determine the disposition of the message:

  <new-line>, +, or n  Go on to next message.
  d, or dp            Delete message and go on to next message.
  d #                Delete message number #. Do not go on to next message.
  dq                 Delete message and quit mail.
  h                  Display a window of headers around current message.
  h #                Display header of message number #.
  h a                Display headers of ALL messages in the user's mailfile.
  h d                Display headers of messages scheduled for deletion.
  p                  Print current message again.
  -                  Print previous message.
  a                  Print message that arrived during the mail session.
  #                  Print message number #.
  r [ users ]        Reply to the sender, and other user(s), then delete the message.
  s [ files ]        Save message in the named files (mbox is default).
  y                  Same as save.
  u [ # ]            Undelete message number # (default is last read).
  w [ files ]        Save message, without its top-most header, in the named
                     files (mbox is default).
  m [ persons ]      Mail the message to the named persons.
  q, or ctl-d        Put undeleted mail back in the mailfile and quit mail.
  x                  Put all mail back in the mailfile unchanged and exit mail.
  !command           Escape to the shell to do command.
  ?                  Print a command summary.

When a user logs in, the presence of mail, if any, is indicated. Also, notification is
made if new mail arrives while using mail.

The mailfile may be manipulated in two ways to alter the function of mail. The other
permissions of the file may be read-write, read-only, or neither read nor write to
allow different levels of privacy. If changed to other than the default, the file will be
preserved even when empty to perpetuate the desired permissions. The file may also
contain the first line:

Forward to person

which will cause all mail sent to the owner of the mailfile to be forwarded to person. A "Forwarded by..." message is added to the header. This is especially useful in a
multi-machine environment to forward all of a person's mail to a single machine, and
to keep the recipient informed if the mail has been forwarded. Installation and remo-
val of forwarding is done with the –F option.

To forward all of one’s mail to someone enter:

    mail -F system!user

To forward to more than one user enter:

    mail -F "user1, system!user2, system!systemb!user3"

Note that when more than one user is specified, the whole list should be enclosed in
double quotes so that it may all be interpreted as the operand of the –F option. The
list can be up to 1024 bytes; either commas or white space can be used to separate
users.

To remove forwarding enter:

    mail –F ""

The pair of double quotes is mandatory to set a NULL argument for the –F option.

In order for forwarding to work properly the mailfile should have "mail" as group ID,
and the group permission should be read-write.

FILES

/bin/mail
/etc/passwd  to identify sender and locate persons
/usr/mail/user  incoming mail for user; i.e., the mailfile
$HOME/mbox  saved mail
$MAIL  variable containing path name of mailfile
/tmp/ma*  temporary file
/usr/mail/*.lock  lock for mail directory
dead.letter  unmaillable text

SEE ALSO

login(1), mailx(1), write(1).
User’s Guide.
System Administrator’s Guide.

WARNING

The "Forward to person" feature may result in a loop, if sys1!userb forwards to
sys2!userb and sys2!userb forwards to sys1!userb. The symptom is a message saying
"unbounded...saved mail in dead.letter."

The –s option should be used with caution. It allows the text of a message to be
interpreted as part of the postmark of the letter, possibly causing confusion to other
mail programs. To allow compatibility with mailx(1), if the first line of the message is
"Subject:...", the addition of a <newline> is suppressed whether or not the –s option is
used.

BUGS

Conditions sometimes result in a failure to remove a lock file.
After an interrupt, the next message may not be printed; printing may be forced by
typing a p.
NAME

mailx – interactive message processing system

SYNOPSIS

mailx [options] [name...]

DESCRIPTION

The command mailx provides a comfortable, flexible environment for sending and receiving messages electronically. When reading mail, mailx provides commands to facilitate saving, deleting, and responding to messages. When sending mail, mailx allows editing, reviewing and other modification of the message as it is entered.

Many of the remote features of mailx will only work if the Basic Networking Utilities are installed on your system.

Incoming mail is stored in a standard file for each user, called the mailbox for that user. When mailx is called to read messages, the mailbox is the default place to find them. As messages are read, they are marked to be moved to a secondary file for storage, unless specific action is taken, so that the messages need not be seen again. This secondary file is called the mbox and is normally located in the user’s HOME directory (see "MBOX" (ENVIRONMENT VARIABLES) for a description of this file). Messages can be saved in other secondary files named by the user. Messages remain in a secondary file until forcibly removed.

The user can access a secondary file by using the -f option of the mailx command. Messages in the secondary file can then be read or otherwise processed using the same COMMANDS as in the primary mailbox. This gives rise within these pages to the notion of a current mailbox.

On the command line, options start with a dash (-) and any other arguments are taken to be destinations (recipients). If no recipients are specified, mailx will attempt to read messages from the mailbox. Command line options are:

-e Test for presence of mail. mailx prints nothing and exits with a successful return code if there is mail to read.
-f [filename] Read messages from filename instead of mailbox. If no filename is specified, the mbox is used.
-f Read the message in a file named after the first recipient. Overrides the "record" variable, if set (see ENVIRONMENT VARIABLES).
-h number The number of network "hope" made so far. This is provided for network software to avoid infinite delivery loops. (See addsopt under ENVIRONMENT VARIABLES)
-H Print header summary only.
-I Ignore interrupts. See also "ignore" (ENVIRONMENT VARIABLES).
-n Do not initialize from the system default mailx.rc file.
-N Do not print initial header summary.
-r address Pass address to network delivery software. All tilde commands are disabled. (See addsopt under ENVIRONMENT VARIABLES)
-s subject Set the Subject header field to subject.
-u user Read user’s mailbox. This is only effective if user’s mailbox is not read protected.
-U Convert uiucp style addresses to internet standards. Overrides the "conv" environment variable. (See addsopt under ENVIRONMENT VARIABLES)
When reading mail, *mailx* is in *command mode*. A header summary of the first several messages is displayed, followed by a prompt indicating *mailx* can accept regular commands (see COMMANDS below). When sending mail, *mailx* is in *input mode*. If no subject is specified on the command line, a prompt for the subject is printed. (A "subject" longer than 1024 characters will cause *mailx* to dump core) As the message is typed, *mailx* will read the message and store it in a temporary file. Commands may be entered by beginning a line with the tilde (~) escape character followed by a single command letter and optional arguments. See TILDE ESCAPES for a summary of these commands.

At any time, the behavior of *mailx* is governed by a set of environment variables. These are flags and valued parameters which are set and cleared via the *set* and *unset* commands. See ENVIRONMENT VARIABLES below for a summary of these parameters.

Recipients listed on the command line may be of three types: login names, shell commands, or alias groups. Login names may be any network address, including mixed network addressing. If mail is found to be undeliverable, an attempt is made to return it to the sender's *mailbox*. If the recipient name begins with a pipe symbol (|), the rest of the name is taken to be a shell command to pipe the message through. This provides an automatic interface with any program that reads the standard input, such as *lp*(1) for recording outgoing mail on paper. Alias groups are set by the alias command (see COMMANDS below) and are lists of recipients of any type.

Regular commands are of the form

```
[ command ] [ msglist ] [ arguments ]
```

If no command is specified in *command mode*, print is assumed. In *input mode*, commands are recognized by the escape character, and lines not treated as commands are taken as input for the message.

Each message is assigned a sequential number, and there is at any time the notion of a current message, marked by a right angle bracket (>) in the header summary. Many commands take an optional list of messages (*msglist*) to operate on. The default for *msglist* is the current message. A *msglist* is a list of message identifiers separated by spaces, which may include:

- **n** Message number n.
- **.** The current message.
- **^** The first undeleted message.
- **$** The last message.
- ***** All messages.
- **n-m** An inclusive range of message numbers.
- **user** All messages from user.
- **/string** All messages with string in the subject line (case ignored).
- **:c** All messages of type c, where c is one of:
  - **d** deleted messages
  - **n** new messages
  - **o** old messages
  - **r** read messages
  - **u** unread messages Note that the context of the command determines whether this type of message specification makes sense.

Other arguments are usually arbitrary strings whose usage depends on the command involved. File names, where expected, are expanded via the normal shell conventions (see *sh*(1)). Special characters are recognized by certain commands and are documented with the commands below.
At start-up time, \texttt{mailx} tries to execute commands from the optional system-wide file 
(\texttt{/usr/lib/mailx/mailx.rc}) to initialize certain parameters, then from a private start-up file 
(\texttt{$HOME/.mailrc}) for personalized variables. With the exceptions noted below, regular commands are legal inside start-up files. The most common uses of a start-up file is to set up initial display options and alias lists. The following commands are not legal in the start-up file: \texttt{!}, \texttt{Copy}, \texttt{edit}, \texttt{followup}, \texttt{Followup}, \texttt{hold}, \texttt{mail}, \texttt{preserve}, \texttt{reply}, \texttt{Reply}, \texttt{shell}, and \texttt{visual}. An error in the start-up file causes the remaining lines in the file to be ignored. The \texttt{.mailrc} file is optional, and must be constructed locally.

\textbf{COMMANDS}

The following is a complete list of \texttt{mailx} commands:

\texttt{!shell-command}

\texttt{Escape to the shell. See "SHELL" (ENVIRONMENT VARIABLES).}

\texttt{# comment}

\texttt{Null command (comment). This may be useful in .mailrc files.}

\texttt{=}

\texttt{Print the current message number.}

\texttt{?}

\texttt{Prints a summary of commands.}

\texttt{alias alias name ...}
\texttt{group alias name ...}

\texttt{Declare an alias for the given names. The names will be substituted when alias is used as a recipient. Useful in the .mailrc file.}

\texttt{alternates name ...}

\texttt{Declares a list of alternate names for your login. When responding to a message, these names are removed from the list of recipients for the response. With no arguments, alternates prints the current list of alternate names. See also "allnet" (ENVIRONMENT VARIABLES).}

\texttt{cd [directory]}
\texttt{chdir [directory]}

\texttt{Change directory. If \texttt{directory} is not specified, SHOME is used.}

\texttt{copy [filename]}
\texttt{copy [msglist] filename}

\texttt{Copy messages to the file without marking the messages as saved. Otherwise equivalent to the save command.}

\texttt{Copy [msglist]}

\texttt{Save the specified messages in a file whose name is derived from the author of the message to be saved, without marking the messages as saved. Otherwise equivalent to the Save command.}

\texttt{delete [msglist]}

\texttt{Delete messages from the \texttt{mailbox}. If "autoprint" is set, the next message after the last one deleted is printed (see ENVIRONMENT VARIABLES).}
discard [header-field ...]
ignore [header-field ...]
Suppresses printing of the specified header fields when displaying messages on
the screen. Examples of header fields to ignore are "status" and "cc." The fields
are included when the message is saved. The Print and Type commands over-
ride this command.

dp [msglist]
dt [msglist]
Delete the specified messages from the mailbox and print the next message after
the last one deleted. Roughly equivalent to a delete command followed by a
print command.

echo string ...
Echo the given strings (like echo(1)).

edit [msglist]
Edit the given messages. The messages are placed in a temporary file and the
"EDITOR" variable is used to get the name of the editor (see ENVIRONMENT
VARIABLES). Default editor is ed(1).

exit
xit
Exit from mailx, without changing the mailbox. No messages are saved in the
mbox (see also quit).

file [filename]
folder [filename]
Quit from the current file of messages and read in the specified file. Several
special characters are recognized when used as file names, with the following
substitutions:
  %    the current mailbox.
  %user the mailbox for user.
  #    the previous file.
  &    the current mbox. Default file is the current mailbox.

folders
Print the names of the files in the directory set by the "folder" variable (see
ENVIRONMENT VARIABLES).

followup [message]
Respond to a message, recording the response in a file whose name is derived
from the author of the message. Overrides the "record" variable, if set. See also
the Followup, Save, and Copy commands and "outfolder" (ENVIRONMENT
VARIABLES).

Followup [msglist]
Respond to the first message in the msglist, sending the message to the author
of each message in the msglist. The subject line is taken from the first message
and the response is recorded in a file whose name is derived from the author of
the first message. See also the followup, Save, and Copy commands and "out-
folder" (ENVIRONMENT VARIABLES).
from [msglist]
  Prints the header summary for the specified messages.

group alias name ...
alias alias name ...
  Declare an alias for the given names. The names will be substituted when alias
  is used as a recipient. Useful in the .mailrc file.

headers [message]
  Prints the page of headers which includes the message specified. The "screen"
  variable sets the number of headers per page (see ENVIRONMENT VARIABLES).
  See also the z command.

help
  Prints a summary of commands.

hold [msglist]
preserve [msglist]
  Holds the specified messages in the mailbox.

if s | r
mail-commands
else
mail-commands
endif
  Conditional execution, where s will execute following mail-commands, up to an
  else or endif, if the program is in send mode, and r causes the mail-commands to
  be executed only in receive mode. Useful in the .mailrc file.

ignore header-field ...
discard header-field ...
  Suppresses printing of the specified header fields when displaying messages on
  the screen. Examples of header fields to ignore are "status" and "cc." All fields
  are included when the message is saved. The Print and Type commands over-
  ride this command.

list
  Prints all commands available. No explanation is given.

mail name ...
  Mail a message to the specified users.

Mail name
  Mail a message to the specified user and record a copy of it in a file named after
  that user.

mbox [msglist]
  Arrange for the given messages to end up in the standard mbox save file when
  mailx terminates normally. See "MBOX" (ENVIRONMENT VARIABLES) for a
  description of this file. See also the exit and quit commands.
next [message]
Go to next message matching message. A msglist may be specified, but in this case the first valid message in the list is the only one used. This is useful for jumping to the next message from a specific user, since the name would be taken as a command in the absence of a real command. See the discussion of msglists above for a description of possible message specifications.

pipe [msglist] [shell-command]
Pipe the message through the given shell-command. The message is treated as if it were read. If no arguments are given, the current message is piped through the command specified by the value of the "cmd" variable. If the "page" variable is set, a form feed character is inserted after each message (see ENVIRONMENT VARIABLES).

preserve [msglist]
hold [msglist]
Preserve the specified messages in the mailbox.

Print [msglist]
Type [msglist]
Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the ignore command.

print [msglist]
type [msglist]
Print the specified messages. If "crt" is set, the messages longer than the number of lines specified by the "crt" variable are paged through the command specified by the "PAGER" variable. The default command is pg(1) (see ENVIRONMENT VARIABLES).

quit
Exit from mailx, storing messages that were read in mbox and unread messages in the mailbox. Messages that have been explicitly saved in a file are deleted.

Reply [msglist]
Respond [msglist]
Send a response to the author of each message in the msglist. The subject line is taken from the first message. If "record" is set to a file name, the response is saved at the end of that file (see ENVIRONMENT VARIABLES).

reply [message]
respond [message]
Reply to the specified message, including all other recipients of the message. If "record" is set to a file name, the response is saved at the end of that file (see ENVIRONMENT VARIABLES).

Save [msglist]
Save the specified messages in a file whose name is derived from the author of the first message. The name of the file is taken to be the author's name with all network addressing stripped off. See also the copy, followup, and Followup commands and "outfolder" (ENVIRONMENT VARIABLES).
save [filename]
save [msglist] filename

Save the specified messages in the given file. The file is created if it does not exist. The message is deleted from the mailbox when mailx terminates unless "keepsave" is set (see also ENVIRONMENT VARIABLES and the exit and quit commands).

set
set name
set name=string
set name=number

Define a variable called name. The variable may be given a null, string, or numeric value. Set by itself prints all defined variables and their values. See ENVIRONMENT VARIABLES for detailed descriptions of the mailx variables.

shell

Invoke an interactive shell (see also "SHELL" (ENVIRONMENT VARIABLES)).

size [msglist]
Print the size in characters of the specified messages.

source filename

Read commands from the given file and return to command mode.

top [msglist]
Print the top few lines of the specified messages. If the "toplines" variable is set, it is taken as the number of lines to print (see ENVIRONMENT VARIABLES). The default is 5.

touch [msglist]
Touch the specified messages. If any message in msglist is not specifically saved in a file, it will be placed in the mbox upon normal termination. See exit and quit.

Type [msglist]
Print [msglist]

Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the ignore command.

type [msglist]
ping [msglist]

Print the specified messages. If "crt" is set, the messages longer than the number of lines specified by the "crt" variable are paged through the command specified by the "PAGER" variable. The default command is pg(1) (see ENVIRONMENT VARIABLES).

undelete [msglist]

Restore the specified deleted messages. Will only restore messages deleted in the current mail session. If "autoprint" is set, the last message of those restored is printed (see ENVIRONMENT VARIABLES).
unset name ...
Causes the specified variables to be erased. If the variable was imported from
the execution environment (i.e., a shell variable) then it cannot be erased.

version
Prints the current version and release date.

visual [msglist]
Edit the given messages with a screen editor. The messages are placed in a
temporary file and the "VISUAL" variable is used to get the name of the editor
(see ENVIRONMENT VARIABLES).

write [msglist] filename
Write the given messages on the specified file, minus the header and trailing
blank line. Otherwise equivalent to the save command.

xit
exit
Exit from mailx, without changing the mailbox. No messages are saved in the
mbox (see also quit).

z[+ | –]
Scroll the header display forward or backward one screen-full. The number of
headers displayed is set by the "screen" variable (see ENVIRONMENT VARI-
ABLES).

TILDE ESCAPES
The following commands may be entered only from input mode, by beginning a line
with the tilde escape character (~). See "escape" (ENVIRONMENT VARIABLES) for
changing this special character.

~! shell-command
Escape to the shell.

~.
Simulate end of file (terminate message input).

~: mail-command
~_ mail-command
Perform the command-level request. Valid only when sending a message while
reading mail.

~?
Print a summary of tilde escapes.

~A
Insert the autograph string "Sign" into the message (see ENVIRONMENT VARI-
ABLES).

~a
Insert the autograph string "sign" into the message (see ENVIRONMENT VARI-
ABLES).
~b name ...
     Add the names to the blind carbon copy (Bcc) list.

~c name ...
     Add the names to the carbon copy (Cc) list.

~d
     Read in the dead.letter file. See "DEAD" (ENVIRONMENT VARIABLES) for a
description of this file.

~e
     Invoke the editor on the partial message. See also "EDITOR" (ENVIRONMENT
VARIABLES).

~f [msglist]
     Forward the specified messages. The messages are inserted into the message,
without alteration.

~h
     Prompt for Subject line and To, Cc, and Bcc lists. If the field is displayed with
an initial value, it may be edited as if you had just typed it.

~i string
     Insert the value of the named variable into the text of the message. For exam-
ple, ~A is equivalent to ~i Sign.'

~m [msglist]
     Insert the specified messages into the letter, shifting the new text to the right
one tab stop. Valid only when sending a message while reading mail.

~p
     Print the message being entered.

~q
     Quit from input mode by simulating an interrupt. If the body of the message is
not null, the partial message is saved in dead.letter. See "DEAD" (ENVIRON-
MENT VARIABLES) for a description of this file.

~r filename
~r< filename
~r< !shell-command
     Read in the specified file. If the argument begins with an exclamation point (!),
the rest of the string is taken as an arbitrary shell command and is executed,
with the standard output inserted into the message.

~s string ...
     Set the subject line to string.

~t name ...
     Add the given names to the To list.
~v
Invoke a preferred screen editor on the partial message. See also "VISUAL" (ENVIRONMENT VARIABLES).

~w filename
Write the partial message onto the given file, without the header.

~x
Exit as with ~q except the message is not saved in dead.letter.

~l shell-command
Pipe the body of the message through the given shell-command. If the shell-command returns a successful exit status, the output of the command replaces the message.

ENVIRONMENT VARIABLES
The following are environment variables taken from the execution environment and are not alterable within mailx.

HOME=directory
The user's base of operations.

MAILRC=filename
The name of the start-up file. Default is $HOME/.mailrc.

The following variables are internal mailx variables. They may be imported from the execution environment or set via the set command at any time. The unset command may be used to erase variables.

addsoft
Enabled by default. If /bin/mail is not being used as the deliverer, noaddsoft should be specified. (See WARNINGS below)

allnet
All network names whose last component (login name) match are treated as identical. This causes the msglist message specifications to behave similarly. Default is noallnet. See also the alternates command and the "metoo" variable.

append
Upon termination, append messages to the end of the mbox file instead of prepending them. Default is noappend.

askcc
Prompt for the Cc list after message is entered. Default is noaskcc.

asksub
Prompt for subject if it is not specified on the command line with the -s option. Enabled by default.

autoprint
Enable automatic printing of messages after delete and undelete commands. Default is noautoprint.
bang
   Enable the special-casing of exclamation points (!) in shell escape command lines as in vi(1). Default is nobang.

cmd=shell-command
   Set the default command for the pipe command. No default value.

conv=conversion
   Convert uucp addresses to the specified address style. The only valid conversion now is internet, which requires a mail delivery program conforming to the RFC822 standard for electronic mail addressing. Conversion is disabled by default. See also 'sendmail' and the -U command line option.

crt=number
   Pipe messages having more than number lines through the command specified by the value of the "PAGER" variable (pg(1) by default). Disabled by default.

DEAD=filename
   The name of the file in which to save partial letters in case of untimely interrupt. Default is $HOME/dead.letter.

debag
   Enable verbose diagnostics for debugging. Messages are not delivered. Default is nodbug.

dot
   Take a period on a line by itself during input from a terminal as end-of-file. Default is nodot.

EDITOR=shell-command
   The command to run when the edit or -e command is used. Default is ed(1).

escape=c
   Substitute c for the ~ escape character. Takes effect with next message sent.

folder=directory
   The directory for saving standard mail files. User-specified file names beginning with a plus (+) are expanded by preceding the file name with this directory name to obtain the real file name. If directory does not start with a slash (/), $HOME is prepended to it. In order to use the plus (+) construct on a mailx command line, "folder" must be an exported sh environment variable. There is no default for the "folder" variable. See also "outfolder" below.

header
   Enable printing of the header summary when entering mailx. Enabled by default.

hold
   Preserve all messages that are read in the mailbox instead of putting them in the standard mbox save file. Default is nohold.

ignore
   Ignore interrupts while entering messages. Handy for noisy dial-up lines. Default is noignore.
ignoreeof
Ignore end-of-file during message input. Input must be terminated by a period (. ) on a line by itself or by the -. command. Default is noignoreeof. See also "dot" above.

keep
When the mailbox is empty, truncate it to zero length instead of removing it. Disabled by default.

keepsave
Keep messages that have been saved in other files in the mailbox instead of deleting them. Default is nokeepsave.

MBOX=filename
The name of the file to save messages which have been read. The xit command overrides this function, as does saving the message explicitly in another file. Default is $HOME/mbox.

metoo
If your login appears as a recipient, do not delete it from the list. Default is nometoo.

LISTER=shell-command
The command (and options) to use when listing the contents of the "folder" directory. The default is ls(1).

onehop
When responding to a message that was originally sent to several recipients, the other recipient addresses are normally forced to be relative to the originating author's machine for the response. This flag disables alteration of the recipients' addresses, improving efficiency in a network where all machines can send directly to all other machines (i.e., one hop away).

outfolder
Causes the files used to record outgoing messages to be located in the directory specified by the "folder" variable unless the path name is absolute. Default is nooutfolder. See "folder" above and the Save, Copy, followup, and Followup commands.

page
Used with the pipe command to insert a form feed after each message sent through the pipe. Default is nopage.

PAGER=shell-command
The command to use as a filter for paginating output. This can also be used to specify the options to be used. Default is pg(1).

prompt=string
Set the command mode prompt to string. Default is ".?".

quiet
Refrain from printing the opening message and version when entering mailx. Default is noquiet.
record=filename
Record all outgoing mail in filename. Disabled by default. See also "outfolder" above.

save
Enable saving of messages in dead.letter on interrupt or delivery error. See "DEAD" for a description of this file. Enabled by default.

screen=number
Sets the number of lines in a screen–full of headers for the headers command.

sendmail=shell-command
Alternate command for delivering messages. Default is mail(1).

sendwait
Wait for background mailer to finish before returning. Default is nosendwait.

SHELL=shell-command
The name of a preferred command interpreter. Default is sh(1).

showto
When displaying the header summary and the message is from you, print the recipient's name instead of the author's name.

sign=string
The variable inserted into the text of a message when the -a (autograph) command is given. No default (see also -i (TILDE ESCAPES)).

Sign=string
The variable inserted into the text of a message when the -A command is given. No default (see also -i (TILDE ESCAPES)).

toplines=number
The number of lines of header to print with the top command. Default is 5.

VISUAL=shell-command
The name of a preferred screen editor. Default is vi(1).

FILES
$HOME/.mailrc  personal start-up file
$HOME/mbox secondary storage file
/usr/mail/*  post office directory
/usr/lib/mailx/mailx.help* help message files
/usr/lib/mailx/mailx.rc optional global start-up file
/tmp/R[emq]sx* temporary files

SEE ALSO
Is(1), mail(1), pg(1).

WARNINGS
The -h, -r and -U options can be used only if mailx is built with a delivery program other than /bin/mail.

BUGS
Where shell-command is shown as valid, arguments are not always allowed. Experimentation is recommended.
Internal variables imported from the execution environment cannot be unset.
The full internet addressing is not fully supported by mailx. The new standards need some time to settle down.
Attempts to send a message having a line consisting only of a "." are treated as the end of the message by mail(1) (the standard mail delivery program).
NAME
make - maintain, update, and regenerate groups of programs

SYNOPSIS

DESCRIPTION
The make command allows the programmer to maintain, update, and regenerate groups of computer programs. The following is a brief description of all options and some special names:

-f makefile  Description file name. makefile is assumed to be the name of a description file.
-p  Print out the complete set of macro definitions and target descriptions.
-i  Ignore error codes returned by invoked commands. This mode is entered if the fake target name .IGNORE appears in the description file.
-k  Abandon work on the current entry if it fails, but continue on other branches that do not depend on that entry.
-s  Silent mode. Do not print command lines before executing. This mode is also entered if the fake target name .SILENT appears in the description file.
-r  Do not use the built-in rules.
-n  No execute mode. Print commands, but do not execute them. Even lines beginning with an @ are printed.
-b  Compatibility mode for old makefiles.
-e  Environment variables override assignments within makefiles.
-u  Force an unconditional update.
-t  Touch the target files (causing them to be up-to-date) rather than issue the usual commands.
-q  Question. The make command returns a zero or non-zero status code depending on whether the target file is or is not up-to-date.

.DEFAULT  If a file must be made but there are no explicit commands or relevant built-in rules, the commands associated with the name .DEFAULT are used if it exists.

.PRECIOUS  Dependents of this target will not be removed when quit or interrupt are hit.

.SILENT  Same effect as the -s option.

.IGNORE  Same effect as the -i option.

make executes commands in makefile to update one or more target names. Name is typically a program. If no -f option is present, makefile, Makefile, and the Source Code Control System(SCCS) files s.makefile, and s.Makefile are tried in order. If makefile is -, the standard input is taken. More than one - makefile argument pair may appear.

make updates a target only if its dependents are newer than the target (unless the -u option is used to force an unconditional update). All prerequisite files of a target are added recursively to the list of targets. Missing files are deemed to be out-of-date.

makefile contains a sequence of entries that specify dependencies. The first line of an entry is a blank-separated, non-null list of targets, then a ;, then a (possibly null) list of prerequisite files or dependencies. Text following a ; and all following lines that
begin with a tab are shell commands to be executed to update the target. The first non-empty line that does not begin with a tab or \# begins a new dependency or macro definition. Shell commands may be continued across lines with the <backslash><new-line> sequence. Everything printed by make (except the initial tab) is passed directly to the shell as is. Thus,

```
  echo a\n
  b
```

will produce

```
  ab
```

exactly the same as the shell would.

Sharp (\#) and new-line surround comments.

The following `makefile` says that `pgm` depends on two files `a.o` and `b.o`, and that they in turn depend on their corresponding source files (`a.c` and `b.c`) and a common file `incl.h`:

```
pgm: a.o  b.o
    cc a.o  b.o  -o  pgm

  a.o:  incl.h  a.c
        cc  -c  a.c

  b.o:  incl.h  b.c
        cc  -c  b.c
```

Command lines are executed one at a time, each by its own shell. The SHELL environment variable can be used to specify which shell make should use to execute commands. The default is `/bin/sh`. The first one or two characters in a command can be the following: \-, \@, \@-, or \@-. If \@ is present, printing of the command is suppressed. If \- is present, make ignores an error. A line is printed when it is executed unless the \-s option is present, or the entry .SILENT: is in makefile, or unless the initial character sequence contains a \@. The \-n option specifies printing without execution; however, if the command line has the string `$(MAKE)` in it, the line is always executed (see discussion of the MAKEFLAGS macro under Environment). The \-t (touch) option updates the modified date of a file without executing any commands.

Commands returning non-zero status normally terminate make. If the \-l option is present, or the entry .IGNORE: appears in makefile, or the initial character sequence of the command contains \-, the error is ignored. If the \-k option is present, work is abandoned on the current entry, but continues on other branches that do not depend on that entry.

The \-b option allows old makefiles (those written for the old version of make) to run without errors.

Interrupt and quit cause the target to be deleted unless the target is a dependent of the special name .PRECIOUS.

Environment

The environment is read by make. All variables are assumed to be macro definitions and processed as such. The environment variables are processed before any makefile and after the internal rules; thus, macro assignments in a makefile override environment variables. The \-e option causes the environment to override the macro assignments in a makefile. Suffixes and their associated rules in the makefile will override any identical suffixes in the built-in rules.

The MAKEFLAGS environment variable is processed by make as containing any legal input option (except \-f and \-p) defined for the command line. Further, upon invocation, make "invents" the variable if it is not in the environment, puts the current
options into it, and passes it on to invocations of commands. Thus, MAKEFLAGS always contains the current input options. This proves very useful for "super-makes". In fact, as noted above, when the -n option is used, the command $(MAKE) is executed anyway; hence, one can perform a make -n recursively on a whole software system to see what would have been executed. This is because the -n is put in MAKEFLAGS and passed to further invocations of $(MAKE). This is one way of debugging all of the makefiles for a software project without actually doing anything.

Include Files

If the string include appears as the first seven letters of a line in a makefile, and is followed by a blank or a tab, the rest of the line is assumed to be a file name and will be read by the current invocation, after substituting for any macros.

Macros

Entries of the form string1 = string2 are macro definitions. String2 is defined as all characters up to a comment character or an unescaped new-line. Subsequent appearances of $(string1[subst1=[subst2]]) are replaced by string2. The parentheses are optional if a single character macro name is used and there is no substitute sequence. The optional :subst1=subst2 is a substitute sequence. If it is specified, all non-overlapping occurrences of subst1 in the named macro are replaced by subst2. Strings (for the purposes of this type of substitution) are delimited by blanks, tabs, new-line characters, and beginnings of lines. An example of the use of the substitute sequence is shown under Libraries.

Internal Macros

There are five internally maintained macros which are useful for writing rules for building targets.

$* The macro $* stands for the file name part of the current dependent with the suffix deleted. It is evaluated only for inference rules.

$@ The $@ macro stands for the full target name of the current target. It is evaluated only for explicitly named dependencies.

$< The $< macro is only evaluated for inference rules or the .DEFAULT rule. It is the module which is out-of-date with respect to the target (i.e., the "manufactured" dependent file name). Thus, in the .c.o rule, the $< macro would evaluate to the .c file. An example for making optimized .o files from .c files is:

    .c.o:
     cc -c -O $*.c

or:

    .c.o:
     cc -c -O $<

$? The $? macro is evaluated when explicit rules from the makefile are evaluated. It is the list of prerequisites that are out-of-date with respect to the target; essentially, those modules which must be rebuilt.

$% The $% macro is only evaluated when the target is an archive library member of the form lib(file.o). In this case, $@ evaluates to lib and $% evaluates to the library member, file.o.

Four of the five macros can have alternative forms. When an upper case D or F is appended to any of the four macros, the meaning is changed to "directory part" for D and "file part" for F. Thus, $(@D) refers to the directory part of the string $@. If there is no directory part, . is generated. The only macro excluded from this alternative form is $?.
Suffices

Certain names (for instance, those ending with .o) have inferable prerequisites such as .c, .s, etc. If no update commands for such a file appear in makefile, and if an inferable prerequisite exists, that prerequisite is compiled to make the target. In this case, make has inference rules which allow building files from other files by examining the suffices and determining an appropriate inference rule to use. The current default inference rules are:

.c   .c~   .f   .f~   .sh  .sh~
.c.o  .c.a  .c~.o  .c~.c  .c~.a
.f.o  .f.a  .f~.o  .f~.f  .f~.a
.h~h  .s.o  .s~.o  .s~.s  .s~.a  .sh~sh
.l.o  .l.c  .l~.o  .l~.l  .l~.c
.y.o  .y.c  .y~.o  .y~.y  .y~.c

The internal rules for make are contained in the source file rules.c for the make program. These rules can be locally modified. To print out the rules compiled into the make on any machine in a form suitable for recomilation, the following command is used:

    make -fp - 2>/dev/null < /dev/null

A tilde in the above rules refers to an SCCS file [see sccsfile(4)]. Thus, the rule .c~.o would transform an SCCS C source file into an object file (.o). Because the s. of the SCCS files is a prefix, it is incompatible with make's suffix point of view. Hence, the tilde is a way of changing any file reference into an SCCS file reference.

A rule with only one suffix (i.e., .c) is the definition of how to build $x$ from $x.c$. In effect, the other suffix is null. This is useful for building targets from only one source file (e.g., shell procedures, simple C programs).

Additional suffixes are given as the dependency list for .SUFFIXES. Order is significant; the first possible name for which both a file and a rule exist is inferred as a prerequisite. The default list is:

.SUFFIXES: .o .c .c~ .y .y~ .l .l~ .s .s~ .sh .sh~ .h .h~ .f .f~

Here again, the above command for printing the internal rules will display the list of suffixes implemented on the current machine. Multiple suffix lists accumulate; .SUFFIXES; with no dependencies clears the list of suffixes.

Inference Rules

The first example can be done more briefly.

    pgm: a.o b.o
    cc a.o b.o -o pgm
    a.o b.o: incl.h

This is because make has a set of internal rules for building files. The user may add rules to this list by simply putting them in the makefile.

Certain macros are used by the default inference rules to permit the inclusion of optional matter in any resulting commands. For example, CFLAGS, LFLAGS, and YFLAGS are used for compiler options to cc(1), lex(1), and yacc(1), respectively. Again, the previous method for examining the current rules is recommended.

The inference of prerequisites can be controlled. The rule to create a file with suffix .o from a file with suffix .c is specified as an entry with .c.o: as the target and no dependents. Shell commands associated with the target define the rule for making a .o file from a .c file. Any target that has no slashes in it and starts with a dot is identified as a rule and not a true target.
Libraries

If a target or dependency name contains parentheses, it is assumed to be an archive library, the string within parentheses referring to a member within the library. Thus `lib(file.o)` and `$(LIB)(file.o)` both refer to an archive library which contains `file.o`. (This assumes the `LIB` macro has been previously defined.) The expression `$(LIB)(file1.o file2.o)` is not legal. Rules pertaining to archive libraries have the form `.XX.a` where the `XX` is the suffix from which the archive member is to be made. An unfortunate byproduct of the current implementation requires the `XX` to be different from the suffix of the archive member. Thus, one cannot have `lib(file.o)` depend upon `file.o` explicitly. The most common use of the archive interface follows. Here, we assume the source files are all C type source:

```
lib: lib(file1.o) lib(file2.o) lib(file3.o)
    @echo lib is now up-to-date
    .c.a:
        $(CC) -c $(CFLAGS) $<
        $(AR) $(ARFLAGS) $@ $*.o
        rm -f $*.o
```

In fact, the `.c.a` rule listed above is built into `make` and is unnecessary in this example. A more interesting, but more limited example of an archive library maintenance construction follows:

```
lib: lib(file1.o) lib(file2.o) lib(file3.o)
    $(CC) -c $(CFLAGS) $(?=.o=.c)
    $(AR) $(ARFLAGS) lib $?
    rm $? @echo lib is now up-to-date
    .c.a;
```

Here the substitution mode of the macro expansions is used. The `$?` list is defined to be the set of object file names (inside `lib`) whose C source files are out-of-date. The substitution mode translates the `.o` to `.c`. (Unfortunately, one cannot as yet transform to `.c`; however, this may become possible in the future.) Note also, the disabling of the `.c.a` rule, which would have created each object file, one by one. This particular construct speeds up archive library maintenance considerably. This type of construct becomes very cumbersome if the archive library contains a mix of assembly programs and C programs.

FILES

[Mm]akefile and s.[Mm]akefile
/bin/sh

SEE ALSO

cc(1), cd(1), lex(1), sh(1), yacc(1), printf(3S), sccsfile(4).

NOTES

Some commands return non-zero status inappropriately; use `–i` to overcome the difficulty.

BUGS

File names with the characters = : @ will not work. Commands that are directly executed by the shell, notably `cd(1)`, are ineffectual across new-lines in `make`. The syntax `lib(file1.o file2.o file3.o)` is illegal. You cannot build `lib(file.o)` from `file.o`. The macro `$(a.o=.c)` does not work. Named pipes are not handled well.
NAME
makekey – generate encryption key

SYNOPSIS
/usr/lib/makekey

DESCRIPTION
makekey improves the usefulness of encryption schemes depending on a key by increasing the amount of time required to search the key space. It reads 10 bytes from its standard input, and writes 13 bytes on its standard output. The output depends on the input in a way intended to be difficult to compute (i.e., to require a substantial fraction of a second).

The first eight input bytes (the input key) can be arbitrary ASCII characters. The last two (the salt) are best chosen from the set of digits, ,, /, and upper- and lower-case letters. The salt characters are repeated as the first two characters of the output. The remaining 11 output characters are chosen from the same set as the salt and constitute the output key.

The transformation performed is essentially the following: the salt is used to select one of 4,096 cryptographic machines all based on the National Bureau of Standards DES algorithm, but broken in 4,096 different ways. Using the input key as key, a constant string is fed into the machine and recirculated a number of times. The 64 bits that come out are distributed into the 66 output key bits in the result.

makekey is intended for programs that perform encryption. Usually, its input and output will be pipes.

SEE ALSO
ed(1), crypt(1), vt(1), passwd(4).

WARNING
This command is provided with the Security Administration Utilities, which is only available in the United States.
NAME

man – find manual information by keywords; print out the manual

SYNOPSIS

man [ - ] [ -M path ] [ bsd ] [ section ] title ...
man –k keyword ...
man –f file ...

DESCRIPTION

man is a program which gives information from the programmers manual. It can be asked for one line descriptions of commands specified by name, or for all commands whose description contains any of a set of keywords. It can also provide on-line access to the sections of the printed manual.

When given the option –k and a set of keywords, man prints out a one-line synopsis of each manual sections whose listing in the table of contents contains one of those keywords.

When given the option –f and a list of file names, man attempts to locate manual sections related to those files, printing out the table of contents lines for those sections.

When neither –k nor –f is specified, man formats a specified set of manual pages. If a section specifier is given man looks in the that section of the manual for the given titles. Section is either an Arabic section number (3 for instance), or one of the words "new," "local," "old," or "public." A section number may be followed by a single letter classifier (for instance, 1m, indicating an administration program in section 1). If section is omitted, man searches all sections of the manual, giving preference to commands over subroutines in system libraries, and printing the first section it finds, if any.

When given the option bsd, man looks only for man pages related to commands and subroutines from the 4.3 BSD distribution.

If the standard output is a teletype, or if the flag – is given, man pipes its output through more(1) with the option –s to crush out useless blank lines and to stop after each page on the screen. Hit a space to continue, a control-D to scroll 11 more lines when the output stops.

Normally man checks in a standard location for manual information (/usr/man). This can be changed by supplying a search path (a la the shell) with the –M flag. The search path is a colon (:) separated list of directories in which manual subdirectories may be found; e.g. "/usr/local:/usr/man".

MANPATH

If the environment variable ‘MANPATH’ is set, its value is used for the default path. If a search path is supplied with the –k or –f options, it must be specified first.

FILES

/usr/man standard manual area
/usr/man/cat?/* directories containing preformatted pages
/usr/man/what is keyword database
/usr/man/cat?/Index index of entries and filenames
/usr/man/bsd/cat?/* standard manual area for bsd pages

SEE ALSO

apropos(1), more(1), man(5), catman(8)
NAME

msg – permit or deny messages

SYNOPSIS

msg [-n] [-y]

DESCRIPTION

msg with argument n forbids messages via write(1) by revoking non-user write permission on the user's terminal. msg with argument y reinstates permission. All by itself, msg reports the current state without changing it.

FILES

/dev/tty*

SEE ALSO

write(1).

DIAGNOSTICS

Exit status is 0 if messages are receivable, 1 if not, 2 on error.
NAME
mkdir – make directories

SYNOPSIS
mkdir [ -m mode ] [ -p ] dirame ...

DESCRIPTION
mkdir creates the named directories in mode 777 (possibly altered by umask(1)).
Standard entries in a directory (e.g., the files ., for the directory itself, and .., for its
parent) are made automatically. mkdir cannot create these entries by name. Creation
of a directory requires write permission in the parent directory.
The owner ID and group ID of the new directories are set to the process’s real user ID
and group ID, respectively.
Two options apply to mkdir:
-m This option allows users to specify the mode to be used for new directories.
Choices for modes can be found in chmod(1).
-p With this option, mkdir creates dirame by creating all the non-existing parent
directories first.

EXAMPLE
To create the subdirectory structure ltr/ld/jan, type:

    mkdir -p ltr/ld/jan

SEE ALSO
sh(1), rm(1), umask(1), intro(2), mkdir(2).

DIAGNOSTICS
mkdir returns exit code 0 if all directories given in the command line were made suc-
cessfully. Otherwise, it prints a diagnostic and returns non-zero. An error code is
stored in errno.
NAME
mkfarm – configure a striped file system

SYNOPSIS
/etc/mkfarm [-m [-y]] farm_number special special ...

DESCRIPTION
Mkfarm configures a striped file system by associating all the specials into 1 logical file
system known as /dev/farm/farm(farm_number).

The -m option allows an mkfs to be done to the logical disk farm at the same time.
The -m option always makes an Ardent Fast File System. If the -m option is used, the
user is prompted for a yes/no answer before mkfarm proceeds to do the mkfs on the
striped file system. The -y option suppresses the prompt and silently proceeds with
the mkfs.

DIAGNOSTICS
mkfarm requires that all the disk partitions named have the same parameters (bytes
per sector, sectors per track, tracks per cylinder), and that they are the same size.
Mkfarm will use the entire partitions named on the command line.

SEE ALSO
dconf(1M), dflist(1M), dfreset(1M), mkfs(1M)
MKFS(1M)

NAME

mkfs – construct a file system

SYNOPSIS

/etc/mkfs special blocks[i-nodes] [gap blocks/cyl]
/etc/mkfs special proto [gap blocks/cyl]
/etc/mkfs [-t type] file sys proto [ gap blocks/cyl ]
/etc/mkfs [-t type] file sys blocks[i-nodes] [ gap blocks/cyl ]

DESCRIPTION

mkfs constructs a file system by writing on the special file using the values found in
the remaining arguments of the command line. Invoking mkfs with only the -t
option lists the types of file systems mkfs knows how to create. Names listed on the
same line are synonyms; the first type listed is the default type. Invoking mkfs with
-t and a file system type gives the command syntax for the file system type.

4K FsTYPE4 Dana5V Sys5-4K
AFFS affs FFS ffs DANA Dana

If the second argument is a string of digits, the size of the file system is the value of
blocks interpreted as a decimal number. This is the number of 512-byte sectors the file
system will occupy. If the number of i-nodes is not given, the default is the number of
logical (512, 1024, or 4096 byte) blocks divided by 4. mkfs builds a file system with
a single empty directory on it. The boot program block (block zero) is left uninitial-
ized.

If the second argument is the name of a file that can be opened, mkfs assumes it to be
a prototype file proto, and will take its directions from that file. The prototype file
contains tokens separated by spaces or new-lines. A sample prototype specification
follows (line numbers have been added to aid in the explanation):

1. /stand/diskboot
2. 4872 110
3. d—777 3 1
4.  d—777 3 1
5. usr sh —755 3 1 /bin/sh
6. ken d—755 6 1
7. $
8. b0 b—644 3 1 0 0
9. c0 c—644 3 1 0 0
10. $
11. $

Line 1 in the example is the name of a file to be copied onto block zero as the
bootstrap program.

Line 2 specifies the number of 512-byte sectors the file system is to occupy and the
number of i-nodes in the file system.

Lines 3-9 tell mkfs about files and directories to be included in this file system.

Line 3 specifies the root directory.

lines 4-6 and 8-9 specifies other directories and files.

The $ on line 7 tells mkfs to end the branch of the file system it is on, and continue
from the next higher directory. The $ on lines 10 and 11 end the process, since no
additional specifications follow.

File specifications give the mode, the user ID, the group ID, and the initial contents of
the file. Valid syntax for the contents field depends on the first character of the
mode.
The mode for a file is specified by a 6-character string. The first character specifies the type of the file. The character range is -bcd to specify regular, block special, character special and directory files respectively. The second character of the mode is either u or - to specify set-user-id mode or not. The third is g or - for the set-group-id mode. The rest of the mode is a 3 digit octal number giving the owner, group, and other read, write, execute permissions (see chmod(1)).

Two decimal number tokens come after the mode; they specify the user and group IDs of the owner of the file.

If the file is a regular file, the next token of the specification may be a path name to which the contents and size are copied. If the file is a block or character special file, two decimal numbers follow which give the major and minor device numbers. If the file is a directory, mkfs makes the entries . and .. and then reads a list of names and (recursively) file specifications for the entries in the directory. As noted above, the scan is terminated with the token $.

SEE ALSO

chmod(1), dir(4), dvhtool(1M), fs(4)

Installation/Administration Manual

BUGS

There is no way to specify links. The maximum number of i-nodes configurable is 65500.
NAME

mklist -- PostScript label generator

SYNOPSIS

mklist [-pos ] file(s)

DESCRIPTION

mklist writes multiple labels on the default PostScript device using LW(IPS). It is generally used to produce labels for bulk mailings. (Note: mklist requests manual feed and will print as many sheets of labels as necessary to list all the names in file(s).)

The addresses in file(s) should have the following format:

    name 1
    addr 1-1
    addr 1-2
    ...
    %EON%
    name 2
    addr 2-1
    addr 2-2
    ...
    %EON%
    ...

Addresses shouldn't be longer than five lines, as small (one inch high) labels can't fit much more.

The following option is understood:

- pos     pos (an integer number) determines the first label that is written (say the first sheet is only partially used). See mkenv(IPS) for information on the label numbering scheme.

EXAMPLE

Print labels for the addresses in mail.list starting on label #10:

    $ mklist -10 mail.list

FILES

/usr/lib/laser/maillist     PostScript prolog for labels

SEE ALSO

LW(IPS), mkenv(IPS)
Devps User's Guide

WARNING

mklist will print as many address lines on the envelope or label as you give it. Specifying too many lines, particularly on small labels, will cause the address to "spill over" onto the next label.
NAME
mknod – build special file

SYNOPSIS
/etc/mknod name b | c major minor
/etc/mknod name p

DESCRIPTION
mknod makes a directory entry and corresponding i-node for a special file.
The first argument is the name of the entry. The UNIX convention is to keep such files
in the /dev directory.
In the first case, the second argument is b if the special file is block-type (disks, tape)
or c if it is character-type (other devices). The last two arguments are numbers specifying the major device type and the minor device (e.g., unit, drive, or line number).
They may be either decimal or octal. The assignment of major device numbers is
specific to each system. The information is contained in the system source file
master.c. You must be the super-user to use this form of the command.
The second case is the form of the mknod that is used to create FIFO’s (a.k.a named
pipes).

SEE ALSO
mknod(2)
NAME

mkstr - create an error message file by massaging C source

SYNOPSIS

mkstr [-] messagefile prefix file ...

DESCRIPTION

mkstr is used to create files of error messages. Its use can make programs with large numbers of error diagnostics much smaller, and reduce system overhead in running the program as the error messages do not have to be constantly swapped in and out.

mkstr will process each of the specified files, placing a massaged version of the input file in a file whose name consists of the specified prefix and the original name. A typical usage of mkstr would be

    mkstr pstrings xx *.c

This command would cause all the error messages from the C source files in the current directory to be placed in the file pstrings and processed copies of the source for these files to be placed in files whose names are prefixed with xx.

To process the error messages in the source to the message file mkstr keys on the string "error" in the input stream. Each time it occurs, the C string starting at the "" is placed in the message file followed by a null character and a new-line character; the null character terminates the message so it can be easily used when retrieved, the new-line character makes it possible to sensibly cat the error message file to see its contents. The massaged copy of the input file then contains a fseek pointer into the file which can be used to retrieve the message, i.e.: 

```c
char  efilname[] = "/usr/lib/pi_strings";
int   efil = -1;

error(a1, a2, a3, a4)
{
    char buf[256];

    if (efil < 0) {
        efil = open(efilname, 0);
        if (efil < 0) {
            oops:
            perror(efilname);
            exit(1);
        }
    }
    if (fseek(efil, (long) a1, 0) || read(efil, buf, 256) <= 0)
        goto oops;
    printf(buf, a2, a3, a4);
}
```

The optional - causes the error messages to be placed at the end of the specified message file for recompiling part of a large mkstr ed program.

SEE ALSO

lseek(2), xstr(1)
NAME
more, page – file perusal filter for crt viewing

SYNOPSIS
more [-cdfs] [-n] [+linenumber] [+pattern] [name ...]
page more options

DESCRIPTION
more is a filter which allows examination of a continuous text one screenful at a time
on a soft-copy terminal. It normally pauses after each screenful, printing --More-- at
the bottom of the screen. If the user then types a carriage return, one more line is
displayed. If the user hits a space, another screenful is displayed. Other possibilities
are enumerated later.

The command line options are:

- \(-n\) An integer which is the size (in lines) of the window which more will use
  instead of the default.

- \(-c\) more will draw each page by beginning at the top of the screen and erasing each
  line just before it draws on it. This avoids scrolling the screen, making it easier
  to read while more is writing. This option will be ignored if the terminal does
  not have the ability to clear to the end of a line.

- \(-d\) more will prompt the user with the message "Press space to continue, 'q' to
  quit." at the end of each screenful, and will respond to subsequent illegal user
  input by printing "Press 'h' for instructions." instead of ringing the bell. This is
  useful if more is being used as a filter in some setting, such as a class, where
  many users may be unsophisticated.

- \(-f\) This causes more to count logical, rather than screen lines. That is, long lines are
  not folded. This option is recommended if nroff output is being piped through
  ul, since the latter may generate escape sequences. These escape sequences con-
tain characters which would ordinarily occupy screen positions, but which do
  not print when they are sent to the terminal as part of an escape sequence.
  Thus more may think that lines are longer than they actually are, and fold lines
  erroneously.

- \(-l\) Do not treat ^L (form feed) specially. If this option is not given, more will pause
  after any line that contains a ^L, as if the end of a screenful had been reached.
  Also, if a file begins with a form feed, the screen will be cleared before the file is
  printed.

- \(-s\) Squeeze multiple blank lines from the output, producing only one blank line.
  Especially helpful when viewing nroff output, this option maximizes the useful
  information present on the screen.

- \(-u\) Normally, more will handle underlining such as produced by nroff in a manner
  appropriate to the particular terminal: if the terminal can perform underlining
  or has a stand-out mode, more will output appropriate escape sequences to
  enable underlining or stand-out mode for underlined information in the source
  file. The \(-u\) option suppresses this processing.

+linenumber
Start up at linenumber.

+pattern
Start up two lines before the line containing the regular expression pattern.
If the program is invoked as page, then the screen is cleared before each screenful is printed (but only if a full screenful is being printed), and \( k - 1 \) rather than \( k - 2 \) lines are printed in each screenful, where \( k \) is the number of lines the terminal can display.

`more` looks in the file `/etc/termcap` to determine terminal characteristics, and to determine the default window size. On a terminal capable of displaying 24 lines, the default window size is 22 lines.

`more` looks in the environment variable `MORE` to pre-set any flags desired. For example, if you prefer to view files using the `--c` mode of operation, the `csh` command `setenv MORE --c` or the `sh` command sequence `MORE='--c' ; export MORE` would cause all invocations of `more`, including invocations by programs such as `man` and `msgs`, to use this mode. Normally, the user will place the command sequence which sets up the `MORE` environment variable in the `.cshrc` or `.profile` file.

If `more` is reading from a file, rather than a pipe, then a percentage is displayed along with the `More` prompt. This gives the fraction of the file (in characters, not lines) that has been read so far.

Other sequences which may be typed when `more` pauses, and their effects, are as follows (i is an optional integer argument, defaulting to 1):

- `i<space>`: display i more lines, (or another screenful if no argument is given)
- `^D`: display 11 more lines (a "scroll"). If i is given, then the scroll size is set to i.
- `d`: same as `^D` (control-D)
- `iz`: same as typing a space except that i, if present, becomes the new window size.
- `is`: skip i lines and print a screenful of lines
- `if`: skip i screenfuls and print a screenful of lines
- `ib`: skip back i screenfuls and print a screenful of lines
- `i^B`: same as `b`
- `q` or `Q`: Exit from `more`.
- `=`: Display the current line number.
- `v`: Start up the editor `vi` at the current line.
- `h`: Help command; give a description of all the `more` commands.
- `/expr`: search for the i-th occurrence of the regular expression `expr`. If there are less than i occurrences of `expr`, and the input is a file (rather than a pipe), then the position in the file remains unchanged. Otherwise, a screenful is displayed, starting two lines before the place where the expression was found. The user's erase and kill characters may be used to edit the regular expression. Erasing back past the first column cancels the search command.
- `in`: search for the i-th occurrence of the last regular expression entered.
- `''`: (single quote) Go to the point from which the last search started. If no search has been performed in the current file, this command goes back to the beginning of the file.
- `!command`: invoke a shell with `command`. The characters `'` and `!` in "command" are replaced with the current file name and the previous shell command respectively. If there is no current file name, `'` is not expanded. The sequences
"\%" and "\!" are replaced by "%" and "!" respectively.

i:n skip to the i-th next file given in the command line (skips to last file if n doesn't make sense)

i:p skip to the i-th previous file given in the command line. If this command is
given in the middle of printing out a file, then more goes back to the beginning
of the file. If i doesn't make sense, more skips back to the first file. If more is not
reading from a file, the bell is rung and nothing else happens.

:f display the current file name and line number.

q or :Q
    exit from more (same as q or Q).

. (dot) repeat the previous command.

The commands take effect immediately, i.e., it is not necessary to type a carriage
return. Up to the time when the command character itself is given, the user may hit
the line kill character to cancel the numerical argument being formed. In addition,
the user may hit the erase character to redisplay the --More--(xx%) message.

At any time when output is being sent to the terminal, the user can hit the quit key
(normally control-\). more will stop sending output, and will display the usual
More-- prompt. The user may then enter one of the above commands in the normal
manner. Unfortunately, some output is lost when this is done, due to the fact that
any characters waiting in the terminal's output queue are flushed when the quit sig-
nal occurs.

The terminal is set to naecho mode by this program so that the output can be continu-
ous. What you type will thus not show on your terminal, except for the / and ! com-
mands.

If the standard output is not a teletype, then more acts just like cat, except that a
header is printed before each file (if there is more than one).

A sample usage of more in previewing nroff output would be

    nroff -ms +2 doc.n | more -s

FILES

/etc/termcap       Terminal data base
/usr/lib/more.help Help file

SEE ALSO

csh(1), man(1), msgs(1), script(1), sh(1), environ(7)

BUGS

Skipping backwards is too slow on large files.
NAME

mount, umount – mount and umount file systems and remote resources

SYNOPSIS

/etc/mount [-r] [-f fstyp] special directory
/etc/mount [-r] [-f NFS [ option ]] resource directory
/etc/umount special

DESCRIPTION

File systems other than root (/) are considered removable in the sense that they can be either available to users or unavailable. mount announces to the system that special, a block special device or resource, a remote resource, is available to users from the mount point directory. directory must exist already; it becomes the name of the root of the newly mounted special.

mount, when entered with arguments, adds an entry to the table of mounted devices, /etc/mnttab. umount removes the entry. If invoked with no arguments, mount prints the entire mount table. If invoked with an incomplete argument list, mount searches /etc/fstab for the missing arguments.

The following options are available:

-r indicates that special or resource is to be mounted read-only. If special or resource is write-protected, this flag must be used.

-f NFS indicates that an NFS file system is to be mounted. NFS options may be added after NFS separated by commas. The available NFS options are:

soft return error if the server doesn’t respond.

rsize=n set the read buffer size to n bytes.

wsize=n set the write buffer size to n bytes

timeo=n set the initial NFS timeout to n tenths of a second

retrans=n set the number of NFS retransmissions to n

port=n set the server IP port number to n

special indicates the block special device that is to be mounted on directory. If the file system type is NFS, then special should be of the form hostname/pathname.

resource indicates the remote resource name that is to be mounted on a directory. If the file system type is NFS, then mount attempts to translate resource into a special of the form hostname/pathname, and then proceeds as if a special were originally specified.

directory indicates the directory mount point for special or resource. (The directory must already exist.)

umount announces to the system that the file system previously mounted special or resource is to be made unavailable. If invoked with an incomplete argument list, umount searches /etc/fstab for the missing arguments.

mount can be used by any user to list mounted file systems and resources. Only a super-user can mount and umount file systems.
FILES

/etc/mnttab    mount table
/etc/fstab     file system table

SEE ALSO

fstab(4), fuser(1M), mnttab(4), mount(2), setmnt(1M), umount(2)

DIAGNOSTICS

If the mount(2) system call fails, mount prints an appropriate diagnostic. mount issues a warning if the file system to be mounted is currently mounted under another name. A remote resource mount will fail if the resource is not available.

umount fails if special or resource is not mounted or if it is busy. special or resource is busy if it contains an open file or some user's working directory. In such a case, you can use fuser(1M) to list and kill processes that are using special or resource.
NAME

mountall, umountall - mount, umount multiple file systems

SYNOPSIS

/etc/mountall [-] [file-system-table] ...
/etc/umountall [-k]

DESCRIPTION

These commands may be executed only by the super-user.

mountall is used to mount file systems according to a file-system-table. (/etc/fstab is the default file system table.) The special file name "-" reads from the standard input.

Before each file system is mounted, it is checked using fsstat(1M) to see if it appears mountable. If the file system does not appear mountable, it is checked, using fsck(1M), before the mount is attempted.

umountall causes all mounted file systems except root to be unmounted. The -k option sends a SIGKILL signal, via fuser(1M), to processes that have files open.

FILES

File-system-table format:

    column 1      block special file name of file system
    column 2      mount-point directory
    column 3      "-r" if to be mounted read-only
    column 4      (optional) file system type string
    column 5+     ignored

White-space separates columns. Lines beginning with "#" are comments. Empty lines are ignored.

A typical file-system-table might read:

    /dev/dsk/c1d0s2/usr -r S51K

SEE ALSO

fsck(1M), fsstat(1M), fstable(4), fuser(1M), mount(1M), signal(2), sysadm(1).

DIAGNOSTICS

No messages are printed if the file systems are mountable and clean.
Error and warning messages come from fsck(1M), fsstat(1M), and mount(1M).
NAME

mt – magnetic tape manipulating program

SYNOPSIS

mt [-f tapename] command [count]

DESCRIPTION

Mt is used to give commands to a magnetic tape drive. If a tape name is not
specified, the environment variable TAPE is used; if TAPE does not exist, mt uses the
device /dev/rmt12. Note that tapename must reference a raw (not block) tape device.
By default mt performs the requested operation once. Operations may be performed
multiple times by specifying count.

The available commands are listed below. Only as many characters as are required
to uniquely identify a command need be specified.

eof, weof
   Write count end-of-file marks at the current position on the tape.

fsf   Forward space count files.

fsr   Forward space count records.

bsf   Back space count files.

bsr   Back space count records.

rewind
   Rewind the tape (Count is ignored).

offline, rewoffl
   Rewind the tape and place the tape unit off-line (Count is ignored).

status
   Print status information about the tape unit.

Mt returns a 0 exit status when the operation(s) were successful, 1 if the command
was unrecognized, and 2 if an operation failed.

FILES

/dev/rmt* Raw magnetic tape interface

SEE ALSO

mtio(4), dd(1), ioctl(2), environ(7)
NAME

mvdir – move a directory

SYNOPSIS

/etc/mvdir dirname name

DESCRIPTION

mvdir moves directories within a file system. Dirname must be a directory. If name does not exist, it will be created as a directory. If name does exist, dirname will be created as name/dirname. Dirname and name may not be on the same path; that is, one may not be subordinate to the other. For example:

mvdir x/y x/z

is legal, but

mvdir x/y x/y/z

is not.

SEE ALSO

mkdir(1), mv(1).

WARNINGS

Only the super-user can use mvdir.
NAME
ncheck – generate path names from i-numbers

SYNOPSIS
/etc/ncheck [-i i-numbers] [-a] [-s] [file-system]

DESCRIPTION
ncheck with no arguments generates a path-name vs. i-number list of all files on a set of default file systems (see /etc/checklist). Names of directory files are followed by ".

The options are as follows:
- **i** limits the report to only those files whose i-numbers follow.
- **a** allows printing of the names . and .., which are ordinarily suppressed.
- **s** limits the report to special files and files with set-user-ID mode. This option may be used to detect violations of security policy.

File system must be specified by the file system’s special file.

The report should be sorted so that it is more useful.

SEE ALSO
fsck(1M), sort(1).

DIAGNOSTICS
If the file system structure is not consistent, ?? denotes the “parent” of a parentless file and a path-name beginning with ... denotes a loop.
NAME

nda - network daemon

SYNOPSIS

nda [-d] [-f confile]

DESCRIPTION

nda is the master network daemon and is usually started by rc2(1M). Currently, nda will open and link the following devices: ethernet, arp, ip, tcp, udp, and raw ip.

The driver information is specified in the default file /etc/nda.conf. The default can be changed with the -f option. Each line in the configuration file indicates the presence of a "stream." For example, the following line indicates that there exists a stream between the ip (mux) driver and the et driver.

    et   ip

The -d option provides debugging information.

This command is restricted to the super-user.

FILES

/etc/nda.conf    driver configuration

SEE ALSO

rc2(1M), uname(2)
NAME

netstat - show network status

SYNOPSIS

netstat [-Aan] [-f address_family] [system] [core]
netstat [-himnrs] [-f address_family] [system] [core]
netstat [-n] [-I interface] interval [system] [core]

DESCRIPTION

The netstat command symbolically displays the contents of various network-related data structures. There are a number of output formats, depending on the options for the information presented. The first form of the command displays a list of active sockets for each protocol. The second form presents the contents of one of the other network data structures according to the option selected. Using the third form, with an interval specified, netstat will continuously display the information regarding packet traffic on the configured network interfaces.

The options have the following meaning:

-A With the default display, show the address of any protocol control blocks associated with sockets; used for debugging.
-a With the default display, show the state of all sockets; normally sockets used by server processes are not shown.
-h Show the state of the IMP host table.
-i Show the state of interfaces which have been auto-configured (interfaces statically configured into a system, but not located at boot time are not shown).
-I interface
  Show information only about this interface; used with an interval as described below.
-m Show statistics recorded by the memory management routines (the network manages a private pool of memory buffers).
-n Show network addresses as numbers (normally netstat interprets addresses and attempts to display them symbolically). This option may be used with any of the display formats.
-s Show per-protocol statistics.
-r Show the routing tables. When -s is also present, show routing statistics instead.

-f address_family
  Limit statistics or address control block reports to those of the specified address family. The following address families are recognized: inet, for AF_INET, ns, for AF_NS, and unix, for AF_UNIX.

The arguments, system and core allow substitutes for the defaults "/unix" and "/dev/kmem".

The default display, for active sockets, shows the local and remote addresses, send and receive queue sizes (in bytes), protocol, and the internal state of the protocol. Address formats are of the form "host:port" or "network:port" if a socket's address specifies a network but no specific host address. When known the host and network addresses are displayed symbolically according to the data bases /etc/hosts and /etc/networks, respectively. If a symbolic name for an address is unknown, or if the -n option is specified, the address is printed numerically, according to the address family. For more information regarding the Internet "dot format," refer to inet(3N). Unspecified, or "wildcard", addresses and ports appear as "**".
The interface display provides a table of cumulative statistics regarding packets transferred, errors, and collisions. The network addresses of the interface and the maximum transmission unit ("mtu") are also displayed.

The routing table display indicates the available routes and their status. Each route consists of a destination host or network and a gateway to use in forwarding packets. The flags field shows the state of the route ("U" if "up"), whether the route is to a gateway ("G"), and whether the route was created dynamically by a redirect ("D"). Direct routes are created for each interface attached to the local host; the gateway field for such entries shows the address of the outgoing interface. The refcnt field gives the current number of active uses of the route. Connection oriented protocols normally hold on to a single route for the duration of a connection while connectionless protocols obtain a route while sending to the same destination. The use field provides a count of the number of packets sent using that route. The interface entry indicates the network interface utilized for the route.

When netstat is invoked with an interval argument, it displays a running count of statistics related to network interfaces. This display consists of a column for the primary interface (the first interface found during autoconfiguration) and a column summarizing information for all interfaces. The primary interface may be replaced with another interface with the -I option. The first line of each screen of information contains a summary since the system was last rebooted. Subsequent lines of output show values accumulated over the preceding interval.

**SEE ALSO**

hosts(5), networks(5), protocols(5), services(5)

**BUGS**

The notion of errors is ill-defined. Collisions mean something else for the IMP.
NAME
newform – change the format of a text file

SYNOPSIS

DESCRIPTION
newform reads lines from the named files, or the standard input if no input file is named, and reproduces the lines on the standard output. Lines are reformatted in accordance with command line options in effect.

Except for -s, command line options may appear in any order, may be repeated, and may be intermingled with the optional files. Command line options are processed in the order specified. This means that option sequences like "-e15 -l60" will yield results different from "-l60 -e15". Options are applied to all files on the command line.

-s Shears off leading characters on each line up to the first tab and places up to 8 of the sheared characters at the end of the line. If more than 8 characters (not counting the first tab) are sheared, the eighth character is replaced by a * and any characters to the right of it are discarded. The first tab is always discarded.

An error message and program exit will occur if this option is used on a file without a tab on each line. The characters sheared off are saved internally until all other options specified are applied to that line. The characters are then added at the end of the processed line.

For example, to convert a file with leading digits, one or more tabs, and text on each line, to a file beginning with the text, all tabs after the first expanded to spaces, padded with spaces out to column 72 (or truncated to column 72), and the leading digits placed starting at column 73, the command would be:

newform -s -i -l -a -e file-name

-itabspec Input tab specification: expands tabs to spaces, according to the tab specifications given. Ttabspec recognizes all tab specification forms described in tabs(1). In addition, tabspec may be --, in which newform assumes that the tab specification is to be found in the first line read from the standard input (see fspec(4)). If no tabspec is given, tabspec defaults to -8. A tabspec of -0 expects no tabs; if any are found, they are treated as -1.

-otabspec Output tab specification: replaces spaces by tabs, according to the tab specifications given. The tab specifications are the same as for -itabspec. If no tabspec is given, tabspec defaults to -8. A tabspec of -0 means that no spaces will be converted to tabs on output.

-bn Truncate n characters from the beginning of the line when the line length is greater than the effective line length (see -ln). Default is to truncate the number of characters necessary to obtain the effective line length. The default value is used when -b with no n is used. This option can be used to delete the sequence numbers from a COBOL program as follows:
newform -l1 -b7 file-name

-en Same as -bn except that characters are truncated from the end of the line.

-pn Prefix n characters (see -ck) to the beginning of a line when the line length is less than the effective line length. Default is to prefix the number of characters necessary to obtain the effective line length.
-an Same as -pn except characters are appended to the end of a line.
-ť Write the tab specification format line on the standard output before any other lines are output. The tab specification format line which is printed will correspond to the format specified in the last -o option. If no -o option is specified, the line which is printed will contain the default specification of -8.
-ck Change the prefix/append character to k. Default character for k is a space.
-ň Set the effective line length to n characters. If n is not entered, -ň defaults to 72. The default line length without the -ň option is 80 characters. Note that tabs and backspaces are considered to be one character (use -ň to expand tabs to spaces).

The -ňň must be used to set the effective line length shorter than any existing line in the file so that the -b option is activated.

**DIAGNOSTICS**

All diagnostics are fatal.

**usage:** ... newform was called with a bad option.
**not -s format** There was no tab on one line.
**can't open file** Self-explanatory.
**internal line too long** A line exceeds 512 characters after being expanded in the internal work buffer.
**tapspec in error** A tab specification is incorrectly formatted, or specified tab stops are not ascending.
**tapspec indirection illegal** A tapspec read from a file (or standard input) may not contain a tapspec referencing another file (or standard input).

0 – normal execution
1 – for any error

**SEE ALSO**

csplit(1), tabs(1), fspec(4).

**BUGS**

newform normally only keeps track of physical characters; however, for the -ň and -o options, newform will keep track of backspaces in order to line up tabs in the appropriate logical columns.

newform will not prompt the user if a tapspec is to be read from the standard input (by use of -ňň or -o--).

If the -ť option is used, and the last -o option specified was -o--", and was preceded by either a -o-- or a -ň--", the tab specification format line will be incorrect.
NAME
newgrp – log in to a new group

SYNOPSIS
newgrp [-] [ group ]

DESCRIPTION
newgrp changes a user’s group identification. The user remains logged in and the
current directory is unchanged, but calculations of access permissions to files are per-
formed with respect to the new real and effective group IDs. The user is always given
a new shell, replacing the current shell, by newgrp, regardless of whether it ter-
minated successfully or due to an error condition (i.e., unknown group).

Exported variables retain their values after invoking newgrp; however, all unex-
ported variables are either reset to their default value or set to null. System variables
(such as PS1, PS2, PATH, MAIL, and HOME), unless exported by the system or
explicitly exported by the user, are reset to default values. For example, a user has a
primary prompt string (PS1) other than $ (default) and has not exported PS1. After
an invocation of newgrp, successful or not, their PS1 will now be set to the default
prompt string $. Note that the shell command export (see sh(1)) is the method to
export variables so that they retain their assigned value when invoking new shells.

With no arguments, newgrp changes the group identification back to the group
specified in the user’s password file entry. This is a way to exit the effect of an earlier
newgrp command.

If the first argument to newgrp is a –, the environment is changed to what would be
expected if the user actually logged in again as a member of the new group.

A password is demanded if the group has a password and the user does not, or if the
group has a password and the user is not listed in /etc/group as being a member of
that group.

FILES
/etc/group system’s group file
/etc/passwd system’s password file

SEE ALSO
environ(5), group(4), login(1), passwd(4), sh(1).

BUGS
There is no convenient way to enter a password into /etc/group. Use of group pass-
words is not encouraged, because, by their very nature, they encourage poor security
practices. Group passwords may disappear in the future.
NAME

news – print news items

SYNOPSIS

news [-a] [-n] [-s] [items]

DESCRIPTION

news is used to keep the user informed of current events. By convention, these events are described by files in the directory /usr/news.

When invoked without arguments, news prints the contents of all current files in /usr/news, most recent first, with each preceded by an appropriate header. news stores the “currency” time as the modification date of a file named .news_time in the user’s home directory (the identity of this directory is determined by the environment variable $HOME); only files more recent than this currency time are considered “current.”

-a option causes news to print all items, regardless of currency. In this case, the stored time is not changed.

-n option causes news to report the names of the current items without printing their contents, and without changing the stored time.

-s option causes news to report how many current items exist, without printing their names or contents, and without changing the stored time. It is useful to include such an invocation of news in one’s .profile file, or in the system’s /etc/profile.

All other arguments are assumed to be specific news items that are to be printed.

If a delete is typed during the printing of a news item, printing stops and the next item is started. Another delete within one second of the first causes the program to terminate.

FILES

/etc/profile
/usr/news/*
$HOME/.news_time

SEE ALSO

profile(4), environ(5).
NAME

nice – run a command at low priority

SYNOPSIS

nice [ –increment ] command [ arguments ]

DESCRIPTION

nice executes command with a lower CPU scheduling priority. If the increment argument (in the range 1-19) is given, it is used; if not, an increment of 10 is assumed.

The super-user may run commands with priority higher than normal by using a negative increment, e.g., –10.

SEE ALSO

nohup(1), nice(2).

DIAGNOSTICS

nice returns the exit status of the subject command.

BUGS

An increment larger than 19 is equivalent to 19.
NAME
nl – line numbering filter

SYNOPSIS

[-nformat] [-ddelim] file

DESCRIPTION

nl reads lines from the named file or the standard input if no file is named and reproduces
the lines on the standard output. Lines are numbered on the left in accordance with the
command options in effect.

nl views the text it reads in terms of logical pages. Line numbering is reset at the
start of each logical page. A logical page consists of a header, a body, and a footer
section. Empty sections are valid. Different line numbering options are independ-
ently available for header, body, and footer (e.g., no numbering of header and
footer lines while numbering blank lines only in the body).

The start of logical page sections are signaled by input lines containing nothing but
the following delimiter character(s):

<table>
<thead>
<tr>
<th>Line contents</th>
<th>Start of</th>
</tr>
</thead>
<tbody>
<tr>
<td>::::\:</td>
<td>header</td>
</tr>
<tr>
<td>::::</td>
<td>body</td>
</tr>
<tr>
<td>::</td>
<td>footer</td>
</tr>
</tbody>
</table>

Unless optioned otherwise, nl assumes the text being read is in a single logical page
body.

Command options may appear in any order and may be intermingled with an
optional file name. Only one file may be named. The options are:

- btype Specifies which logical page body lines are to be numbered. Recognized
types and their meaning are:

- htype Same as btype except for header. Default type for logical page header is
  n (no lines numbered).

  a number all lines
t number lines with printable text only
n no line numbering
pstring number only lines that contain the regular expression
  specified in string.

Default type for logical page body is t (text lines numbered).

- ftype Same as -btype except for footer. Default for logical page footer is n (no
  lines numbered).

- vstart# Start# is the initial value used to number logical page lines. Default is 1.

- lincre Incr is the increment value used to number logical page lines. Default is
  1.

- p Do not restart numbering at logical page delimiters.

- lnum Num is the number of blank lines to be considered as one. For example,
  -12 results in only the second adjacent blank being numbered (if the
  appropriate -ha, -ba, and/or -fa option is set). Default is 1.
-ss sep
  Sep is the character(s) used in separating the line number and the corresponding text line. Default sep is a tab.

-w wwidth
  Width is the number of characters to be used for the line number. Default width is 6.

-n nformat
  Format is the line numbering format. Recognized values are: ln, left justified, leading zeroes suppressed; rn, right justified, leading zeroes suppressed; rz, right justified, leading zeroes kept. Default format is rn (right justified).

-d xx
  The delimiter characters specifying the start of a logical page section may be changed from the default characters (\:) to two user-specified characters. If only one character is entered, the second character remains the default character (:). No space should appear between the –d and the delimiter characters. To enter a backslash, use two backslashes.

**EXAMPLE**

The command:

```
nl -v10 -i10 -d!+ file1
```

will number file1 starting at line number 10 with an increment of ten. The logical page delimiters are !+.

**SEE ALSO**

pr(1).
**NAME**

nm – print name list of common object file

**SYNOPSIS**

nm [-adehnuxVT] filename ...

**DESCRIPTION**

The *nm* command displays the symbol table of each common object file, *filename*. *Filename* may be a relocatable or absolute common object file; or it may be an archive of relocatable or absolute common object files. For each symbol, the following information will be printed:

Name    The name of the symbol.
Value   Its value expressed as an offset or an address depending on its storage class.
Class   Its storage class.
Type    Its type and derived type. If the symbol is an instance of a structure or of a union then the structure or union tag will be given following the type (e.g., struct-tag). If the symbol is an array, then the array dimensions will be given following the type (e.g., char[n][m]). Note that the object file must have been compiled with the –g option of the *cc*(1) command for this information to appear.
Size    Its size in bytes, if available. Note that the object file must have been compiled with the –g option of the *cc*(1) command for this information to appear.
Section For storage classes static and external, the object file section containing the symbol (e.g., text, data or bss).

The output of *nm* may be controlled using the following options:

- `a` Print symbols in the order in which they appear in the symbol table.
- `d` Print the value of a symbol in decimal rather than hexadecimal.
- `e` Print only external and static symbols.
- `h` Do not display the output header data.
- `n` Sort symbols by name before they are printed.
- `o` Print the value and size of a symbol in octal instead of decimal.
- `r` Prepend the name of the object file or archive to each output line.
- `u` Print undefined symbols only.
- `v` Sort symbols by value before they are printed (default).
- `x` Print the value of a symbol in hexadecimal (default).
- `V` Print the version of the *nm* command executing on the standard error output.
- `T` By default, *nm* prints the entire name of the symbols listed. Since object files can have symbols names with an arbitrary number of characters, a name that is longer than the width of the column set aside for names will overflow its column, forcing every column after the name to be misaligned. The –T option causes *nm* to truncate every name which would otherwise overflow its column and place an asterisk as the last character in the displayed name to mark it as truncated.
Options may be used in any order, either singly or in combination.

**SEE ALSO**

as(1), cc(1), ld(1), a.out(4), ar(4).
NAME

nohup - run a command immune to hangups and quits

SYNOPSIS

nohup command [ arguments ]

DESCRIPTION

nohup executes command with hangups and quits ignored. If output is not redirected by the user, both standard output and standard error are sent to nohup.out. If nohup.out is not writable in the current directory, output is redirected to $HOME/nohup.out.

This nohup is a built-in command in the csh. This man page applies to /bin/nohup. The C shell has a built-in nohup command that is different; see csh(1).

EXAMPLE

It is frequently desirable to apply nohup to pipelines or lists of commands. This can be done only by placing pipelines and command lists in a single file, called a shell procedure. One can then issue:

    nohup sh file

and the nohup applies to everything in file. If the shell procedure file is to be executed often, then the need to type sh can be eliminated by giving file execute permission. Add an ampersand and the contents of file are run in the background with interrupts also ignored (see sh(1)):

    nohup file &

An example of what the contents of file could be is:

    sort ofile > nfile

SEE ALSO

chmod(1), nice(1), sh(1), signal(2).

WARNINGS

In the case of the following command

    nohup command1; command2

nohup applies only to command1. The command

    nohup (command1; command2)

is syntactically incorrect.
NAME

nslookup – query name servers interactively

SYNOPSIS

nslookup [ host-to-find | – [ server address | server name ]]

DESCRIPTION

nslookup is a program to query DARPA Internet domain name servers. nslookup has
two modes: interactive and non-interactive. Interactive mode allows the user to
query the name server for information about various hosts and domains or print a
list of hosts in the domain. Non-interactive mode is used to print just the name and
Internet address of a host or domain.

ARGUMENTS

Interactive mode is entered in the following cases:
a) when no arguments are given (the default name server will be used), and
b) when the first argument is a hyphen (–) and the second argument is the host
   name of a name server.

Non-interactive mode is used when the name of the host to be looked up is given as
the first argument. The optional second argument specifies a name server.

INTERACTIVE COMMANDS

Commands may be interrupted at any time by typing a control-C. To exit, type a
control-D (EOF). The command line length must be less than 80 characters. N.B. an
unrecognized command will be interpreted as a host name.

host [server]
   Look up information for host using the current default server or using server if it
   is specified.

server domain
ls server domain
   Change the default server to domain. Lserver uses the initial server to look up
   information about domain while server uses the current default server. If an
   authoritative answer can’t be found, the names of servers that might have the
   answer are returned.

root
   Changes the default server to the server for the root of the domain name space.
Currently, the host sri-nic.arpa is used. (This command is a synonym for the
ls server sri-nic.arpa.) The name of the root server can be changed with the set
root command.

finger [name] [ > filename]
finger [name] [>> filename]
   Connects with the finger server on the current host. The current host is defined
when a previous lookup for a host was successful and returned address infor-
mation (see the set querytype=A command). Name is optional. > and >> can be
used to redirect output in the usual manner.
**ls domain [>> filename]**

List the information available for domain. The default output contains host names and their Internet addresses. The -a option lists aliases of hosts in the domain. The -h option lists CPU and operating system information for the domain. The -d option lists all contents of a zone transfer. When output is directed to a file, hash marks are printed for every 50 records received from the server.

**view filename**

Sorts and lists the output of previous ls command(s) with more(1).

**help**

? Prints a brief summary of commands.

**set keyword=value**

This command is used to change state information that affects the lookups. Valid keywords are:

- **all** Prints the current values of the various options to set. Information about the current default server and host is also printed.

- **[no]debug**
  
  Turn debugging mode on. A lot more information is printed about the packet sent to the server and the resulting answer.
  
  (Default = nodebug, abbreviation = [no]deb)

- **[no]d2**
  
  Turn exhaustive debugging mode on. Essentially all fields of every packet are printed.
  
  (Default = nod2)

- **[no]defname**
  
  Append the default domain name to every lookup.
  
  (Default = defname, abbreviation = [no]def)

- **[no]search**
  
  With defname, search for each name in parent domains of the current domain.
  
  (Default = search)

- **domain=name**
  
  Change the default domain name to name. The default domain name is appended to all lookup requests if the defname option has been set. The search list is set to parents of the domain with at least two components in their names.
  
  (Default = value in hostname or /etc/resolv.conf, abbreviation = do)

- **querytype=value**
type=value
  Change the type of information returned from a query to one of:
  A the host's Internet address (the default).
  CNAME the canonical name for an alias.
  HINFO the host CPU and operating system type.
  MD the mail destination.
  MX the mail exchanger.
  MG the mail group member.
  MINFO the mailbox or mail list information.
  MR the mail rename domain name.
  NS nameserver for the named zone. Other types specified in the
    RFC883 document are valid but aren't very useful.
    (Abbreviation = q)

[no]recurve
  Tell the name server to query other servers if it does not have the infor-
  mation.
  (Default = recurse, abbreviation = [no]rec)

retry=number
  Set the number of retries to number. When a reply to a request is not
  received within a certain amount of time (changed with set timeout), the
  request is resent. The retry value controls how many times a request is
  resent before giving up.
  (Default = 2, abbreviation = ret)

root=host
  Change the name of the root server to host. This affects the root command.
  (Default = sri-nic.arpa, abbreviation = ro)

timeout=number
  Change the time-out interval for waiting for a reply to number seconds.
  (Default = 10 seconds, abbreviation = t)

[no]vc
  Always use a virtual circuit when sending requests to the server.
  (Default = novc, abbreviation = [no]v)

**DIAGNOSTICS**

If the lookup request was not successful, an error message is printed. Possible errors are:

Time-out
  The server did not respond to a request after a certain amount of time (changed
  with set timeout=value) and a certain number of retries (changed with set
  retry=value).

No information
  Depending on the query type set with the set querytype command, no inform-
  ation about the host was available, though the host name is valid.

Non-existent domain
  The host or domain name does not exist.
Connection refused
Network is unreachable
    The connection to the name or finger server could not be made at the current
    time. This error commonly occurs with finger requests.

Server failure
    The name server found an internal inconsistency in its database and could not
    return a valid answer.

Refused
    The name server refused to service the request.

The following error should not occur and it indicates a bug in the program.

Format error
    The name server found that the request packet was not in the proper format.

FILES
    /etc/resolv.conf    initial domain name and name server addresses.

SEE ALSO
    resolver(3), resolver(5), named(8), RFC882, RFC883

AUTHOR
    Andrew Cherenson
NAME

nvram – display or set values of NVRAM variables

SYNOPSIS

/etc/nvram [name=value]

DESCRIPTION

nvram without any parameters displays the current values set in the system non-volatile memory. Each line displayed contains an NVRAM variable name, followed by an equals sign, followed by a string. For example, typing nvram might display

    baud=9600
    etheraddr=00-00-7a-80-00-01
    netaddr=192.9.200.160
    hostname=ca
    screen_font=2
    path=bfs()
    screen_back=403050
    bootmode=a
    screen_fore=cc8800
    rootdev=scsi(0,5,0)
    swapdev=scsi(0,5,1)
    bootfile=scsi(0,5,8)sash
    dumpdev=scsi(0,5,1)
    console=graphics

nvram can also be used to remove an NVRAM variable, add an NVRAM variable, or remove an NVRAM variable. Typing

    nvram name=

removes name from the list of name/value pairs stored.

Typing

    nvram name=value

where name is an existing NVRAM variable name will change its contents to the string typed. If the name does not already exist, then it is added to the list of name/value pairs.

The value part can be any arbitrary collection of character.

WARNING

Only the super-user is allowed to remove or alter NVRAM values.
NAME

oawk – pattern scanning and processing language

SYNOPSIS

oawk [-Fc] [ prog ] [ parameters ] [ files ]

DESCRIPTION

oawk scans each input file for lines that match any of a set of patterns specified in prog. With each pattern in prog there can be an associated action that will be performed when a line of a file matches the pattern. The set of patterns may appear literally as prog, or in a file specified as -f file. The prog string should be enclosed in single quotes (') to protect it from the shell.

Parameters, in the form x=... y=... etc., may be passed to oawk.

Files are read in order; if there are no files, the standard input is read. The file name – means the standard input. Each line is matched against the pattern portion of every pattern-action statement; the associated action is performed for each matched pattern.

An input line is made up of fields separated by white space. (This default can be changed by using FS; see below). The fields are denoted $1, $2, …; $0 refers to the entire line.

A pattern-action statement has the form:

    pattern { action }

A missing action means print the line; a missing pattern always matches. An action is a sequence of statements. A statement can be one of the following:

    if ( conditional ) statement [ else statement ]
    while ( conditional ) statement
    for ( expression ; conditional ; expression ) statement
    break
    continue
    [...] statement ...
    variable = expression
    print [ expression-list ] [ >expression ]
    printf format [ , expression-list ] [ >expression ]
    next    # skip remaining patterns on this input line
    exit    # skip the rest of the input

Statements are terminated by semicolons, new-lines, or right braces. An empty expression-list stands for the whole line. Expressions take on string or numeric values as appropriate, and are built using the operators +, -, *, /, %, and concatenation (indicated by a blank). The C operators ++, --, +=, -=, *=, /=, and %= are also available in expressions. Variables may be scalars, array elements (denoted x[i]) or fields. Variables are initialized to the null string. Array subscripts may be any string, not necessarily numeric; this allows for a form of associative memory. String constants are quoted (").
The `print` statement prints its arguments on the standard output (or on a file if `>expr` is present), separated by the current output field separator, and terminated by the output record separator. The `printf` statement formats its expression list according to the format [see `printf(3)` in the Programmer's Reference Manual].

The built-in function `length` returns the length of its argument taken as a string, or of the whole line if no argument. There are also built-in functions `exp`, `log`, `sqrt`, and `int`. The last truncates its argument to an integer; `substr(s, m, n)` returns the `n`-character substring of `s` that begins at position `m`. The function `sprintf(fmt, expr, expr, ...)` formats the expressions according to the `printf(3)` format given by `fmt` and returns the resulting string.

Patterns are arbitrary Boolean combinations ( |, ||, &&, and parentheses) of regular expressions and relational expressions. Regular expressions must be surrounded by slashes and are as in `egrep` (see `grep(1)`). Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions. A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines between an occurrence of the first pattern and the next occurrence of the second.

A relational expression is one of the following:

- expression matchop regular-expression
- expression relop expression

where a relop is any of the six relational operators in C, and a matchop is either (for `contains`) or ! (for `does not contain`). A conditional is an arithmetic expression, a relational expression, or a Boolean combination of these.

The special patterns BEGIN and END may be used to capture control before the first input line is read and after the last. BEGIN must be the first pattern, END the last.

A single character `c` may be used to separate the fields by starting the program with:

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

or by using the `-Fc` option.

Other variable names with special meanings include `NF`, the number of fields in the current record; `NR`, the ordinal number of the current record; `FILENAME`, the name of the current input file; `OFS`, the output field separator (default blank); `ORS`, the output record separator (default new-line); and `OFMT`, the output format for numbers (default `%g`).

**EXAMPLES**

Print lines longer than 72 characters:

```
length > 72
```

Print first two fields in opposite order:

```
{ print $2, $1 }
```

Add up first column, print sum and average:

```
{ s += $1 }
END { print "sum is", s, " average is", s/NR }
```

Print fields in reverse order:

```
{ for (i = NF; i > 0; --i) print $i }
```

Print all lines between start/stop pairs:
/start/, /stop/
Print all lines whose first field is different from previous one:
   $1 != prev { print; prev = $1 }
Print file, filling in page numbers starting at 5:
   /Page/ { $2 = n++; }
   { print }
command line: oawk -f program n=5 input

SEE ALSO
awk(1), grep(1), lex(1), sed(1), printf(3S)

BUGS
Input white space is not preserved on output if fields are involved.
There are no explicit conversions between numbers and strings. To force an expression to be treated as a number add 0 to it; to force it to be treated as a string concatenate the null string ("") to it.
NAME

od – octal dump

SYNOPSIS

od [ -bcdosx ] [ file ] [[ + ]offset[ . ][ b ]]

DESCRIPTION

od dumps file in one or more formats as selected by the first argument. If the first
argument is missing, -o is default. The meanings of the format options are:

-b   Interpret bytes in octal.
-c   Interpret bytes in ASCII. Certain non-graphic characters appear as C escapes:
     null=\0, backspace=\b, form-feed=\f, new-line=\n, return=\r, tab=\t; others
     appear as 3-digit octal numbers.
-d   Interpret words in unsigned decimal.
-o   Interpret words in octal.
-s   Interpret 16-bit words in signed decimal.
-x   Interpret words in hex.

The file argument specifies which file is to be dumped. If no file argument is
specified, the standard input is used.

The offset argument specifies the offset in the file where dumping is to commence.
This argument is normally interpreted as octal bytes. If . is appended, the offset is
interpreted in decimal. If b is appended, the offset is interpreted in blocks of 512
bytes. If the file argument is omitted, the offset argument must be preceded by +.

Dumping continues until end-of-file.
NAME
pack, pcat, unpack − compress and expand files

SYNOPSIS
pack [ − ] [ −f ] name ...
pcat name ...
unpack name ...

DESCRIPTION
pack attempts to store the specified files in a compressed form. Wherever possible (and useful), each input file name is replaced by a packed file name.z with the same access modes, access and modified dates, and owner as those of name. The -f option will force packing of name. This is useful for causing an entire directory to be packed even if some of the files will not benefit. If pack is successful, name will be removed. Packed files can be restored to their original form using unpack or pcat.

pack uses Huffman (minimum redundancy) codes on a byte-by-byte basis. If the - argument is used, an internal flag is set that causes the number of times each byte is used, its relative frequency, and the code for the byte to be printed on the standard output. Additional occurrences of - in place of name will cause the internal flag to be set and reset.

The amount of compression obtained depends on the size of the input file and the character frequency distribution. Because a decoding tree forms the first part of each .z file, it is usually not worthwhile to pack files smaller than three blocks, unless the character frequency distribution is very skewed, which may occur with printer plots or pictures.

Typically, text files are reduced to 60-75% of their original size. Load modules, which use a larger character set and have a more uniform distribution of characters, show little compression, the packed versions being about 90% of the original size.

pack returns a value that is the number of files that it failed to compress.

No packing will occur if:

the file appears to be already packed;
the file name has more than 12 characters;
the file has links;
the file is a directory;
the file cannot be opened;
no disk storage blocks will be saved by packing;
a file called name.z already exists;
the .z file cannot be created;
an I/O error occurred during processing.

The last segment of the file name must contain no more than 12 characters to allow space for the appended .z extension. Directories cannot be compressed.

Pcat does for packed files what cat(1) does for ordinary files, except that pcat cannot be used as a filter. The specified files are unpacked and written to the standard output. Thus to view a packed file named name.z use:

pcat name.z or just: pcat name

To make an unpacked copy, say nnn, of a packed file named name.z (without destroying name.z) use the command:

pcat name >nnn
Pcat returns the number of files it was unable to unpack. Failure may occur if:
the file name (exclusive of the .x) has more than 12 characters;
the file cannot be opened;
the file does not appear to be the output of pack.

Unpack expands files created by pack. For each file name specified in the command, a
search is made for a file called name.x (or just name, if name ends in .x). If this file
appears to be a packed file, it is replaced by its expanded version. The new file has
the .x suffix stripped from its name, and has the same access modes, access and
modification dates, and owner as those of the packed file.

Unpack returns a value that is the number of files it was unable to unpack. Failure
may occur for the same reasons that it may in pcat, as well as for the following:

a file with the “unpacked” name already exists;
if the unpacked file cannot be created.

SEE ALSO

cat(1).
NAME

pagesize - print system page size

SYNOPSIS

pagesize

DESCRIPTION

Pagesize prints the size of a page of memory in bytes, as returned by getpagesize(2). This program is useful in constructing portable shell scripts.

SEE ALSO

getpagesize(2)
NAME

pal – turn on/off PAL video mode

SYNOPSIS

pal on | off

DESCRIPTION

pal on sets the graphics subsystem into pal mode, and pal off restores the video to high-resolution. In pal mode, the upper left 640 by 480 pixels will be visible on a low resolution RGB monitor.

It is recommended that "pal off" be bound to some key/button combination under the window system, such as <META><SHIFT><CONTROL><Right-Button>. This way, independent of where the cursor is or the state of the window system, the monitor can be returned to high-resolution mode.

SEE ALSO

rs170(1)
NAME

passwd - change login password

SYNOPSIS

passwd [ name ]

DESCRIPTION

This command changes or installs a password associated with the login name.
Ordinary users may change only the password which corresponds to their login name.
passwd prompts ordinary users for their old password, if any. It then prompts for the new password twice. The first time the new password is entered passwd checks to see if the old password has "aged" sufficiently. Password "aging" is the amount of time (usually a certain number of days) that must elapse between password changes. If "aging" is insufficient the new password is rejected and passwd terminates; see passwd(4).

Assuming "aging" is sufficient, a check is made to insure that the new password meets construction requirements. When the new password is entered a second time, the two copies of the new password are compared. If the two copies are not identical the cycle of prompting for the new password is repeated for at most two more times.

Passwords must be constructed to meet the following requirements:

- Each password must have at least six characters. Only the first eight characters are significant.
- Each password must contain at least two alphabetic characters and at least one numeric or special character. In this case, "alphabetic" means upper and lower case letters.
- Each password must differ from the user's login name and any reverse or circular shift of that login name. For comparison purposes, an upper case letter and its corresponding lower case letter are equivalent.
- New passwords must differ from the old by at least three characters. For comparison purposes, an upper case letter and its corresponding lower case letter are equivalent.

One whose effective user ID is zero is called a super-user; see id(1), and su(1). Super-users may change any password; hence, passwd does not prompt super-users for the old password. Super-users are not forced to comply with password aging and password construction requirements. A super-user can create a null password by entering a carriage return in response to the prompt for a new password.

FILES

/etc/passwd

SEE ALSO

chfn(1), chsh(1), id(1M), login(1), su(1M), crypt(3C), passwd(4)
NAME

paste – merge same lines of several files or subsequent lines of one file

SYNOPSIS

paste file1 file2 ...
paste -d list file1 file2 ...
paste -s [-d list] file1 file2 ...

DESCRIPTION

In the first two forms, paste concatenates corresponding lines of the given input files file1, file2, etc. It treats each file as a column or columns of a table and pastes them together horizontally (parallel merging). If you will, it is the counterpart of cat(1) which concatenates vertically, i.e., one file after the other. In the last form above, paste replaces the function of an older command with the same name by combining subsequent lines of the input file (serial merging). In all cases, lines are glued together with the tab character, or with characters from an optionally specified list. Output is to the standard output, so it can be used as the start of a pipe, or as a filter, if – is used in place of a file name.

The meanings of the options are:

- -d Without this option, the new-line characters of each but the last file (or last line in case of the –s option) are replaced by a tab character. This option allows replacing the tab character by one or more alternate characters (see below).

list One or more characters immediately following –d replace the default tab as the line concatenation character. The list is used circularly, i.e., when exhausted, it is reused. In parallel merging (i.e., no –s option), the lines from the last file are always terminated with a new-line character, not from the list. The list may contain the special escape sequences: \n (new-line), \t (tab), \ (backslash), and \0 (empty string, not a null character). Quoting may be necessary, if characters have special meaning to the shell (e.g., to get one backslash, use “ –d “\n” “).

- -s Merge subsequent lines rather than one from each input file. Use tab for concatenation, unless a list is specified with –d option. Regardless of the list, the very last character of the file is forced to be a new-line.

- May be used in place of any file name, to read a line from the standard input. (There is no prompting).

EXAMPLES

ls | paste -d" " -       list directory in one column
ls | paste - - - -       list directory in four columns
paste -s -d"\t\n" file  combine pairs of lines into lines

SEE ALSO

cut(1), grep(1), pr(1).

DIAGNOSTICS

line too long Output lines are restricted to 511 characters.
too many files Except for –s option, no more than 12 input files may be specified.
NAME
pg – file perusal filter for CRTs

SYNOPSIS
pg [-number] [-p string] [-cefns] [+linenum] [+pattern/] [files...]

DESCRIPTION
The pg command is a filter which allows the examination of files one screenful at a
time on a CRT. (The file name – and/or NULL arguments indicate that pg should
read from the standard input.) Each screenful is followed by a prompt. If the user
types a carriage return, another page is displayed; other possibilities are enumerated
below.

This command is different from previous pageinators in that it allows you to back up
and review something that has already passed. The method for doing this is
explained below.

In order to determine terminal attributes, pg scans the terminfo(4) data base for the
terminal type specified by the environment variable TERM. If TERM is not defined,
the terminal type dumb is assumed.

The command line options are:

-number
An integer specifying the size (in lines) of the window that pg is to use instead
of the default. (On a terminal containing 24 lines, the default window size is
23).

-p string
Causes pg to use string as the prompt. If the prompt string contains a ""%d", the
first occurrence of ""%d"" in the prompt will be replaced by the current page
number when the prompt is issued. The default prompt string is ""."".

-c Home the cursor and clear the screen before displaying each page. This option
is ignored if clear_screen is not defined for this terminal type in the terminfo(4)
data base.

-e Causes pg not to pause at the end of each file.

-f Normally, pg splits lines longer than the screen width, but some sequences of
characters in the text being displayed (e.g., escape sequences for underlining)
generate undesirable results. The -f option inhibits pg from splitting lines.

-n Normally, commands must be terminated by a <newline> character. This option
causes an automatic end of command as soon as a command letter is
entered.

-s Causes pg to print all messages and prompts in standout mode (usually inverse
video).

+linenum
Start up at linenum.

+/pattern/
Start up at the first line containing the regular expression pattern.

The responses that may be typed when pg pauses can be divided into three
categories: those causing further perusal, those that search, and those that modify
the perusal environment.

Commands which cause further perusal normally take a preceding address, an
optionally signed number indicating the point from which further text should be
displayed. This address is interpreted in either pages or lines depending on the com-
mand. A signed address specifies a point relative to the current page or line, and an
unsigned address specifies an address relative to the beginning of the file. Each command has a default address that is used if none is provided.

The perusal commands and their defaults are as follows:

(+1)<newline> or <blank>
This causes one page to be displayed. The address is specified in pages.

(+1) I
With a relative address this causes pg to simulate scrolling the screen, forward or backward, the number of lines specified. With an absolute address this command prints a screenful beginning at the specified line.

(+1) d or ^D
Simulates scrolling half a screen forward or backward.

The following perusal commands take no address.

. or ^L
Typing a single period causes the current page of text to be redisplayed.

$ Displays the last windowful in the file. Use with caution when the input is a pipe.

The following commands are available for searching for text patterns in the text. The regular expressions described in ed(1) are available. They must always be terminated by a <newline>, even if the -n option is specified.

i/pattern/
Search forward for the ith (default i=1) occurrence of pattern. Searching begins immediately after the current page and continues to the end of the current file, without wrap-around.

i^pattern^ i?pattern?
Search backwards for the ith (default i=1) occurrence of pattern. Searching begins immediately before the current page and continues to the beginning of the current file, without wrap-around. The ^ notation is useful for Adds 100 terminals which will not properly handle the ?.

After searching, pg will normally display the line found at the top of the screen. This can be modified by appending m or b to the search command to leave the line found in the middle or at the bottom of the window from now on. The suffix t can be used to restore the original situation.

The user of pg can modify the environment of perusal with the following commands:

in Begin perusing the ith next file in the command line. The i is an unsigned number, default value is 1.

ip Begin perusing the ith previous file in the command line. i is an unsigned number, default is 1.

iw Display another window of text. If i is present, set the window size to i.

s filename
Save the input in the named file. Only the current file being perused is saved. The white space between the s and filename is optional. This command must always be terminated by a <newline>, even if the -n option is specified.

h Help by displaying an abbreviated summary of available commands.

q or Q
Quit pg.
"command"

"Command" is passed to the shell, whose name is taken from the SHELL environment variable. If this is not available, the default shell is used. This command must always be terminated by a "newline", even if the "-n" option is specified.

At any time when output is being sent to the terminal, the user can hit the quit key (normally control-\) or the interrupt (break) key. This causes pg to stop sending output, and display the prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

If the standard output is not a terminal, then pg acts just like cat(1), except that a header is printed before each file (if there is more than one).

**EXAMPLE**

A sample usage of pg in reading system news would be

```
news | pg -p "(Page %d):"
```

**NOTES**

While waiting for terminal input, pg responds to BREAK, DEL, and ^ by terminating execution. Between prompts, however, these signals interrupt pg's current task and place the user in prompt mode. These should be used with caution when input is being read from a pipe, since an interrupt is likely to terminate the other commands in the pipeline.

Users of Berkeley's more will find that the z and f commands are available, and that the terminal /, ^, or ? may be omitted from the searching commands.

**FILES**

```
/usr/lib/terminfo/?/*/ terminal information database
/tmp/pg* temporary file when input is from a pipe
```

**SEE ALSO**

ed(1), grep(1), terminfo(4).

**BUGS**

If terminal tabs are not set every eight positions, undesirable results may occur.

When using pg as a filter with another command that changes the terminal I/O options terminal settings may not be restored correctly.
NAME
pled – control the front panel LED

SYNOPSIS
/etc/pled [on | off | flash]

DESCRIPTION
pled controls the red LED (light emitting diode) located beneath the system name plate on the front of the system cabinet. The main purpose is to signal particular phases of the boot procedure. The options are as follows:

on    turns the red LED on
off   turns the red LED off
flash sets the red LED flashing, approximately once per second

SEE ALSO
sysmips(2) in the Programmer’s Reference Manual

WARNING
This command can be run only by the super-user.
**NAME**

pr – print files

**SYNOPSIS**

```
[-l-length] [-s-separator] [-h header] [file ...]
```

```
[-s-separator] [-h header] file1 file2 ...
```

**DESCRIPTION**

`pr` is used to format and print the contents of a file. If `file` is `-`, or if no files are specified, `pr` assumes standard input. `pr` prints the named files on standard output.

By default, the listing is separated into pages, each headed by the page number, a date and time, and the name of the file. Page length is 66 lines which includes 10 lines of header and trailer output. The header is composed of 2 blank lines, 1 line of text (can be altered with `-h`), and 2 blank lines; the trailer is 5 blank lines. For single column output, line width may not be set and is unlimited. For multicolumn output, line width may be set and the default is 72 columns. Diagnostic reports (failed options) are reported at the end of standard output associated with a terminal, rather than interspersed in the output. Pages are separated by series of line feeds rather than form feed characters.

By default, columns are of equal width, separated by at least one space; lines which do not fit are truncated. If the `-s` option is used, lines are not truncated and columns are separated by the `separator` character.

Either `-column` or `-m` should be used to produce multi-column output. `-a` should only be used with `-column` and not `-m`.

Command line options are

`+page`

Begin printing with page numbered `page` (default is 1).

`-column`

Print `column` columns of output (default is 1). Output appears as if `-e` and `-i` are turned on for multi-column output. May not use with `-m`.

`-a`

Print multi-column output across the page one line per column. `columns` must be greater than one. If a line is too long to fit in a column, it is truncated.

`-m`

Merge and print all files simultaneously, one per column. The maximum number of files that may be specified is eight. If a line is too long to fit in a column, it is truncated. May not use with `-column`.

`-d`

Double-space the output. Blank lines that result from double-spacing are dropped when they occur at the top of a page.

`-eck`

Expand input tabs to character positions `k+1`, `2*k+1`, `3*k+1`, etc. If `k` is 0 or is omitted, default tab settings at every eighth position are assumed. Tab characters in the input are expanded into the appropriate number of spaces. If `c` (any non-digit character) is given, it is treated as the input tab character (default for `c` is the tab character).

`-ick`

In output, replace white space wherever possible by inserting tabs to character positions `k+1`, `2*k+1`, `3*k+1`, etc. If `k` is 0 or is omitted, default tab settings at every eighth position are assumed. If `c` (any non-digit character) is given, it is treated as the output tab character (default for `c` is the tab character).
-ack Provide k-digit line numbering (default for k is 5). The number occupies the first k+1 character positions of each column of single column output or each line of -m output. If c (any non-digit character) is given, it is appended to the line number to separate it from whatever follows (default for c is a tab).

-wwidth
Set the width of a line to width character positions (default is 72). This is effective only for multi-column output (-column and -m). There is no line limit for single column output.

-offset
Offset each line by offset character positions (default is 0). The number of character positions per line is the sum of the width and offset.

-llength
Set the length of a page to length lines (default is 66). –10 is reset to –166. When the value of length is 10 or less, –t appears to be in effect since headers and trailers are suppressed. By default, output contains 5 lines of header and 5 lines of trailer leaving 56 lines for user-supplied text. When –llength is used and length exceeds 10, then length-10 lines are left per page for user supplied text. When length is 10 or less, header and trailer output is omitted to make room for user supplied text.

-h header
Use header as the text line of the header to be printed instead of the file name. –h is ignored when –t is specified or –llength is specified and the value of length is 10 or less. (–h is the only pr option requiring space between the option and argument.)

-p Pause before beginning each page if the output is directed to a terminal (pr will ring the bell at the terminal and wait for a carriage return).

-f Use single form-feed character for new pages (default is to use a sequence of line-feeds). Pause before beginning the first page if the standard output is associated with a terminal.

-r Print no diagnostic reports on files that will not open.

-t Print neither the five-line identifying header nor the five-line trailer normally supplied for each page. Quit printing after the last line of each file without spacing to the end of the page. Use of –t overrides the –h option.

-separator
Separate columns by the single character separator instead of by the appropriate number of spaces (default for separator is a tab). Prevents truncation of lines on multicolumn output unless –w is specified.

EXAMPLES
Print file1 and file2 as a double-spaced, three-column listing headed by “file list”:
```
pr –3dh "file list" file1 file2
```
Copy file1 to file2, expanding tabs to columns 10, 19, 28, 37,...:
```
pr –e9 –t <file1 >file2
```
Print file1 and file2 simultaneously in a two-column listing with no header or trailer where both columns have line numbers:
```
pr –t –n file1 | pr –t –m –n file2 –
```

FILES

Stardent 1500/3000 341
/dev/tty* to delay messages enabling them to print at the bottom of files rather than interspersed throughout printed output.

SEE ALSO

cat(1), pg(1).
NAME
printenv – print out the environment

SYNOPSIS
printenv [ name ]

DESCRIPTION
printenv prints out the values of the variables in the environment. If a name is specified, only its value is printed.
If a name is specified and it is not defined in the environment, printenv returns exit status 1, else it returns status 0.

SEE ALSO
sh(1), environ(7), csh(1)
User Commands

PRMAIL(1)

NAME
prmail – print out mail in the post office

SYNOPSIS
prmail [ user... ]

DESCRIPTION
prmail prints the mail which waits for you, or the specified user, in the post office. The mail is not disturbed.

FILES
/usr/spool/mail/*/ post office

SEE ALSO
biff(1), mail(1), from(1) binmail(1)
NAME
prof, mkprof – display profile data

SYNOPSIS
prof [-v] [prog]

mkprof [-sn] [-Tletter] inprog outprog

DESCRIPTION
The prof command interprets a profile file (called mon.out) produced by running a profiled program. The symbol table in the object file prog (a.out by default) is read and correlated with the profile file. For each function the number of calls of that routine and the total time spent in each routine are reported. In the event that the program ran multiprocessor, the number of calls and the total time are reported for each thread of the process. If the [-v] option is in force, the percentage of the total time, and the average time per call are also reported.

There are two ways to profile a program. A program creates a profile file if it has been loaded with the -p option of cc(1) or fc(1). Note that, unlike most other UNIX implementations, it is not necessary to recompile each routine with -p to obtain routine counts; the effect takes place on the loading step.

The other way to profile is to use the mkprof command. To create a profiled version yyy of the program xxx the command mkprof xxx yyy is run. When yyy is run, a mon.out file is produce which can be interpreted by prof.

There are two options that can be accessed by mkprof only. The option -sn sets the scale factor to n. The hardware clock is in ticks (16ths of a microsecond). The scale factor tells how many bits to shift this to obtain the time that is reported by the profiler. By default, the scale factor is 4, which means that the default unit of time is microseconds. By increasing the scale factor, long-running programs can be timed with no danger of overflowing the (32-bit) counters. The scale factor is kept in the mon.out file and is automatically picked up and interpreted by prof.

The option -Tletter allows you to specify a particular timing function. If letter is f, the FPU time is used (the default setting). If letter is c, the CPU time is used. If letter is e, the elapsed time is used. The letter specified in letter can be in either upper or lower case. CPU time and elapsed time are always measured in ticks (hundreths of a second). The -s (scale) option is ignored when either -Tc or -Te is specified.

FILES
mon.out for profile
a.out for namelist

SEE ALSO
cc(1), fc(1), ld(1)

WARNINGS AND BUGS
The times reported in successive identical runs may show variances due to sharing of the cache with other processes and other effects.

Certain effects (e.g., paging, ETLB misses) may have a significant effect on the real-time performance of programs, but will not show up in these profiled numbers.

There is a very slight chance that, on long programs on busy machines, the clock will ‘turn over’ at the wrong time and some routine will be charged for 2**35 (i.e 2 to the power of 35) extra clock ticks. This effect should be reasonably obvious.

Call counts are always recorded precisely.
NAME
prs – print an SCCS file

SYNOPSIS
prs [-d[dataspec]] [-r[SID]] [-e] [-l] [-c[date-time]] [-a] files

DESCRIPTION
prs prints, on the standard output, parts or all of an SCCS file [see scsfile(4)] in a
user-supplied format. If a directory is named, prs behaves as though each file in the
directory were specified as a named file, except that non-SCCS files (last component
of the path name does not begin with .), and unreadable files are silently ignored. If
a name of – is given, the standard input is read; each line of the standard input is
taken to be the name of an SCCS file or directory to be processed; non-SCCS files and
unreadable files are silently ignored.

Arguments to prs, which may appear in any order, consist of keyletter arguments,
and file names.

All the described keyletter arguments apply independently to each named file:

-d[dataspec] Used to specify the output data specification. The dataspec is a
string consisting of SCCS file data keywords (see DATA KEY-
WORDS) interspersed with optional user supplied text.

-r[SID] Used to specify the SCCS IDentification (SID) string of a delta
for which information is desired. If no SID is specified, the SID
of the most recently created delta is assumed.

-e Requests information for all deltas created earlier than and
including the delta designated via the -r keyletter or the date
given by the -c option.

-l Requests information for all deltas created later than and
including the delta designated via the -r keyletter or the date
given by the -c option.

-c date-time The cutoff date-time -c[cutoff]] is in the form:
YY[MM][DD][HH][MM][SS]][]]

Units omitted from the date-time default to their maximum
possible values; that is, -c7502 is equivalent to -c750228235959.
Any number of non-numeric characters may separate the vari-
ous 2-digit pieces of the cutoff date in the form: "-c77/2/2
9:22:25".

-a Requests printing of information for both removed, i.e., delta
type = R, [see rmdel(1)] and existing, i.e., delta type = D, del-
tas. If the -a keyletter is not specified, information for existing
deltas only is provided.

DATA KEYWORDS
Data keywords specify which parts of an SCCS file are to be retrieved and output. All
parts of an SCCS file [see scsfile(4)] have an associated data keyword. There is no
limit on the number of times a data keyword may appear in a dataspec.

The information printed by prs consists of: (1) the user-supplied text; and (2)
appropriate values (extracted from the SCCS file) substituted for the recognized data
keywords in the order of appearance in the dataspec. The format of a data keyword
value is either Simple (S), in which keyword substitution is direct, or Multi-line (M),
in which keyword substitution is followed by a carriage return.
User-supplied text is any text other than recognized data keywords.
A tab is specified by `\t` and carriage return/new-line is specified by `\n`. The default data keywords are:

```
:"Dt:"t:"DL:"n"MRs:"n"MRs:COMMENTS:"n"C:"
```

### TABLE 1. SCCS Files Data Keywords

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Data Item</th>
<th>File Section</th>
<th>Value</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>:Dt:</td>
<td>Delta information</td>
<td>Delta Table</td>
<td>See below*</td>
<td>S</td>
</tr>
<tr>
<td>:DL:</td>
<td>Delta line statistics</td>
<td></td>
<td>iLi:/iLd:/iLu</td>
<td>S</td>
</tr>
<tr>
<td>:Li:</td>
<td>Lines inserted by Delta</td>
<td></td>
<td>nnnnn</td>
<td>S</td>
</tr>
<tr>
<td>:Ld:</td>
<td>Lines deleted by Delta</td>
<td></td>
<td>nnnnn</td>
<td>S</td>
</tr>
<tr>
<td>:Lu:</td>
<td>Lines unchanged by Delta</td>
<td></td>
<td>nnnnn</td>
<td>S</td>
</tr>
<tr>
<td>:DT:</td>
<td>Delta type</td>
<td></td>
<td>D-or-R</td>
<td>S</td>
</tr>
<tr>
<td>:I:</td>
<td>SCCS ID string (SID)</td>
<td></td>
<td>iR::iL::iB::iS:</td>
<td>S</td>
</tr>
<tr>
<td>:R:</td>
<td>Release number</td>
<td></td>
<td>nnn</td>
<td>S</td>
</tr>
<tr>
<td>:L:</td>
<td>Level number</td>
<td></td>
<td>nnn</td>
<td>S</td>
</tr>
<tr>
<td>:B:</td>
<td>Branch number</td>
<td></td>
<td>nnn</td>
<td>S</td>
</tr>
<tr>
<td>:S:</td>
<td>Sequence number</td>
<td></td>
<td>nnn</td>
<td>S</td>
</tr>
<tr>
<td>:D:</td>
<td>Date Delta created</td>
<td></td>
<td>:Dy:/iDmu:/iDd:</td>
<td>S</td>
</tr>
<tr>
<td>:Dy:</td>
<td>Year Delta created</td>
<td></td>
<td>nn</td>
<td>S</td>
</tr>
<tr>
<td>:Dm:</td>
<td>Month Delta created</td>
<td></td>
<td>nn</td>
<td>S</td>
</tr>
<tr>
<td>:Dd:</td>
<td>Day Delta created</td>
<td></td>
<td>nn</td>
<td>S</td>
</tr>
<tr>
<td>:T:</td>
<td>Time Delta created</td>
<td></td>
<td>iThu::Tmu::Ts:</td>
<td>S</td>
</tr>
<tr>
<td>:Th:</td>
<td>Hour Delta created</td>
<td></td>
<td>nn</td>
<td>S</td>
</tr>
<tr>
<td>:Tm:</td>
<td>Minutes Delta created</td>
<td></td>
<td>nn</td>
<td>S</td>
</tr>
<tr>
<td>:Ts:</td>
<td>Seconds Delta created</td>
<td></td>
<td>nn</td>
<td>S</td>
</tr>
<tr>
<td>:P:</td>
<td>Programmer who created Delta</td>
<td></td>
<td>logname</td>
<td>S</td>
</tr>
<tr>
<td>:DS:</td>
<td>Delta sequence number</td>
<td></td>
<td>nnnn</td>
<td>S</td>
</tr>
<tr>
<td>:DF:</td>
<td>Predecessor Delta seq-no.</td>
<td></td>
<td>nnnn</td>
<td>S</td>
</tr>
<tr>
<td>:Dl:</td>
<td>Seq-no. of deltas incl., excl., ignored</td>
<td></td>
<td>:Dn:/iDlx:/Dg:</td>
<td>S</td>
</tr>
<tr>
<td>:Dn:</td>
<td>Deltas included (seq #)</td>
<td></td>
<td>:Ds::iDs::...</td>
<td>S</td>
</tr>
<tr>
<td>:Dx:</td>
<td>Deltas excluded (seq #)</td>
<td></td>
<td>:Ds::iDs::...</td>
<td>S</td>
</tr>
<tr>
<td>:Dg:</td>
<td>Deltas ignored (seq #)</td>
<td></td>
<td>:Ds::iDs::...</td>
<td>S</td>
</tr>
<tr>
<td>:MR:</td>
<td>MR numbers for delta</td>
<td></td>
<td>text</td>
<td>M</td>
</tr>
<tr>
<td>:C:</td>
<td>Comments for delta</td>
<td></td>
<td>text</td>
<td>M</td>
</tr>
<tr>
<td>:UN:</td>
<td>User names</td>
<td>User Names</td>
<td>text</td>
<td>M</td>
</tr>
<tr>
<td>:FL:</td>
<td>Flag list</td>
<td>Flags</td>
<td>text</td>
<td>M</td>
</tr>
<tr>
<td>:Y:</td>
<td>Module type flag</td>
<td></td>
<td>text</td>
<td>S</td>
</tr>
<tr>
<td>:MF:</td>
<td>MR validation flag</td>
<td></td>
<td>yes-or-no</td>
<td>S</td>
</tr>
<tr>
<td>:MP:</td>
<td>MR validation pgm name</td>
<td></td>
<td>text</td>
<td>S</td>
</tr>
<tr>
<td>:KF:</td>
<td>Keyword error/warning flag</td>
<td></td>
<td>yes-or-no</td>
<td>S</td>
</tr>
<tr>
<td>:KV:</td>
<td>Keyword validation string</td>
<td></td>
<td>text</td>
<td>S</td>
</tr>
<tr>
<td>:BF:</td>
<td>Branch flag</td>
<td></td>
<td>yes-or-no</td>
<td>S</td>
</tr>
<tr>
<td>:j:</td>
<td>Joint edit flag</td>
<td></td>
<td>yes-or-no</td>
<td>S</td>
</tr>
<tr>
<td>:LK:</td>
<td>Locked releases</td>
<td></td>
<td>iR:...</td>
<td>S</td>
</tr>
<tr>
<td>:Q:</td>
<td>User-defined keyword</td>
<td></td>
<td>text</td>
<td>S</td>
</tr>
<tr>
<td>:M:</td>
<td>Module name</td>
<td></td>
<td>text</td>
<td>S</td>
</tr>
<tr>
<td>:FB:</td>
<td>Floor boundary</td>
<td></td>
<td>iR:</td>
<td>S</td>
</tr>
</tbody>
</table>
TABLE 1. SCCS Files Data Keywords (continued)

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Data Item</th>
<th>File Section</th>
<th>Value</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>:CB:</td>
<td>Ceiling boundary</td>
<td>&quot;</td>
<td>:R:</td>
<td>S</td>
</tr>
<tr>
<td>:Ds:</td>
<td>Default SID</td>
<td>&quot;</td>
<td>:I:</td>
<td>S</td>
</tr>
<tr>
<td>:ND:</td>
<td>Null delta flag</td>
<td>&quot;</td>
<td>yes-or-no</td>
<td>S</td>
</tr>
<tr>
<td>:FD:</td>
<td>File descriptive text</td>
<td>Comments</td>
<td>text</td>
<td>M</td>
</tr>
<tr>
<td>:BD:</td>
<td>Body</td>
<td>Body</td>
<td>text</td>
<td>M</td>
</tr>
<tr>
<td>:GB:</td>
<td>Gotten body</td>
<td>&quot;</td>
<td>text</td>
<td>M</td>
</tr>
<tr>
<td>:W:</td>
<td>A form of what(1) string</td>
<td>N/A</td>
<td>:Z:aM:\tl:</td>
<td>S</td>
</tr>
<tr>
<td>:A:</td>
<td>A form of what(1) string</td>
<td>N/A</td>
<td>:Z:aYi::M::-:I::Z:</td>
<td>S</td>
</tr>
<tr>
<td>:Z:</td>
<td>what(1) string delimiter</td>
<td>N/A</td>
<td>@(#)</td>
<td>S</td>
</tr>
<tr>
<td>:F:</td>
<td>SCCS file name</td>
<td>N/A</td>
<td>text</td>
<td>S</td>
</tr>
<tr>
<td>:FN:</td>
<td>SCCS file path name</td>
<td>N/A</td>
<td>text</td>
<td>S</td>
</tr>
</tbody>
</table>

* :Dts=--:DTt=:Iit=:Drt=:Tr=:Ip=:--:DSrt=--:DP:

EXAMPLES

prs -d"Users and/or user IDs for :F: are:\ntUN:" s.file
may produce on the standard output:

Users and/or user IDs for s.file are:

xyz
131
abc

prs -d"Newest delta for pgm :M:: :I: Created :D: By :P:" -r s.file
may produce on the standard output:

Newest delta for pgm main.c: 3.7 Created 77/12/1 By cas

As a special case:

prs s.file

may produce on the standard output:

D 1.1 77/12/1 00:00:00 cas 1 000000/00000/00000
MRs:
bl78-12345
bl79-54321
COMMENTS:
this is the comment line for s.file initial delta

for each delta table entry of the "D" type. The only keyletter argument allowed to be used with the special case is the -a keyletter.

FILES

/tmp/pr??????

SEE ALSO

admin(1), delta(1), get(1), help(1), sccsfile(4).

DIAGNOSTICS

Use help(1) for explanations.
NAME
ps – report process status

SYNOPSIS
ps [ options ]

DESCRIPTION
ps prints certain information about active processes. Without options, information is
printed about processes associated with the controlling terminal. The output consists
of a short listing containing only the process ID, terminal identifier, cumulative execu-
tion time, and the command name. Otherwise, the information that is displayed is
controlled by the selection of options.

Options accept names or lists as arguments. Arguments can be either separated from
one another by commas or enclosed in double quotes and separated from one
another by commas or spaces. Values for proclist and grp-list must be numeric.

The options are given in descending order according to volume and range of informa-
tion provided:

-e Print information about every process now running.
-d Print information about all processes except process group leaders.
-a Print information about all processes most frequently requested: all
    those except process group leaders and processes not associated with a
    terminal.
-f Generate a full listing. (See below for significance of columns in a full
    listing.)
-l Generate a long listing. (See below.)
-v Generate a memory usage oriented listing. (See below.)
-w Generate a cpu and memory usage oriented listing. (See below.)
-c corefile Use the file corefile in place of /dev/mem.
-n name Take argument signifying an alternate system name in place of /unix.
-t termlist List only process data associated with the terminal given in termlist.
    Terminal identifiers may be specified in one of two forms: the device’s
    file name (e.g., tty04) or, if the device’s file name starts with tty, just the
    digit identifier (e.g., 04).
-p proclist List only process data whose process ID numbers are given in proclist.
-u uidlist List only process data whose user ID number or login name is given in
    uidlist. In the listing, the numerical user ID will be printed unless you
give the -f option, which prints the login name.
-g grp-list List only process data whose process group leader’s ID number(s)
    appears in grp-list. (A group leader is a process whose process ID
    number is identical to its process group ID number. A login shell is a
    common example of a process group leader.)

Under the -f option, ps tries to determine the command name and arguments given
when the process was created by examining the user block. Failing this, the com-
mmand name is printed, as it would have appeared without the -f option, in square
brackets.

The column headings and the meaning of the columns in a ps listing are given below;
the letters f, I, w, and v indicate the option (full, long cpu, or memory respectively)
that cause the corresponding heading to appear; all means that the heading always
appears. Note that these four options determine only what information is provided
for a process; they do not determine which processes will be listed.
<table>
<thead>
<tr>
<th>F</th>
<th>Flags (hexadecimal and additive) associated with the process</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Process has terminated: process table entry now available.</td>
</tr>
<tr>
<td>01</td>
<td>A system process: always in primary memory.</td>
</tr>
<tr>
<td>02</td>
<td>Parent is tracing process.</td>
</tr>
<tr>
<td>04</td>
<td>Tracing parent’s signal has stopped process: parent is waiting [ptrace(2)].</td>
</tr>
<tr>
<td>08</td>
<td>Process cannot be woken by a signal.</td>
</tr>
<tr>
<td>10</td>
<td>Process currently in primary memory.</td>
</tr>
<tr>
<td>20</td>
<td>Process is locked in memory and cannot be paged out until an event occurs.</td>
</tr>
<tr>
<td>40</td>
<td>Parent’s signal goes remote.</td>
</tr>
<tr>
<td>100</td>
<td>Process on on run queue</td>
</tr>
<tr>
<td>400</td>
<td>Process is doing I/O via /proc.</td>
</tr>
<tr>
<td>1000</td>
<td>Process is open via /proc.</td>
</tr>
<tr>
<td>2000</td>
<td>Signal mask is restored on return from pause.</td>
</tr>
<tr>
<td>4000</td>
<td>Use 4.3BSD signal semantics.</td>
</tr>
<tr>
<td>8000</td>
<td>Process is a 4.3BSD process group leader.</td>
</tr>
<tr>
<td>80000</td>
<td>Process needs to checkpoint the FPU.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>The state of the process:</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>Process is running on a processor.</td>
</tr>
<tr>
<td>S</td>
<td>Sleeping: process is waiting for an event to complete.</td>
</tr>
<tr>
<td>R</td>
<td>Runnable: process is on run queue.</td>
</tr>
<tr>
<td>I</td>
<td>Idle: process is being created.</td>
</tr>
<tr>
<td>Z</td>
<td>Zombie state: process terminated and parent not waiting.</td>
</tr>
<tr>
<td>T</td>
<td>Traced: process stopped by a signal because parent is tracing it.</td>
</tr>
<tr>
<td>X</td>
<td>SXBRK state: process is waiting for more primary memory.</td>
</tr>
</tbody>
</table>

| L | Cpu on which the process last ran. |

| SZ | The size (in pages or clicks) of the swappable process’s image in main memory. |

| UID | (f,l,w) The user ID number of the process owner (the login name is printed under the -f option). |

| PID | (all) The process ID of the process (this datum is necessary in order to kill a process). |

| PPID | (f,l) The process ID of the parent process. |

| C | Processor utilization for scheduling. |

| PRI | (f,l) The priority of the process (higher numbers mean lower priority). |

| NI | (1) Nice value, used in priority computation. |

| WCHAN | (1) The address of an event for which the process is sleeping, or in SXBRK state, (if blank, the process is running or ready to run). |
STIME (f,w) The starting time of the process, given in hours, minutes, and seconds. (A process begun more than twenty-four hours before the ps inquiry is executed is given in months and days.)

TTY (all) The controlling terminal for the process (the message, ?, is printed when there is no controlling terminal).

TIME (all) The cumulative execution time for the process.

SL (v) Number of seconds the process has been asleep. Will not show times in excess of 99 seconds.

RE (v) Number of seconds the process has been resident in main memory. Will not show times in excess of 99 seconds.

%CPU (v,w) Cpu utilization of the process; this is a decaying average over up to a minute of previous (real) time. Since the time base over which this is computed varies (since processes may be very young) it is possible for the sum of all %CPU fields to exceed 100%.

RSS (v,w) Number of kilobytes (1024) of real memory the process is currently using. Does not reflect shared text or data.

SRSS (v,w) Number of kilobytes (1024) of real memory the process is currently using, divided by the number of users of each shared region.

%MEM (v,w) Percentage of real memory in use by this process. Computed by dividing the SRSS figure by the amount of system memory available for user processes.

COMD (all) The command name (the full command name and its arguments are printed under the -f option).

A process that has exited and has a parent, but has not yet been waited for by the parent, is marked <defunct>.

FILES
/dev
/dev/tty*
/dev/kmem kernel virtual memory
/dev/mem memory
/etc/passwd UID information supplier
/etc/ps_data internal data structure
/unix system namelist

SEE ALSO
getty(1M), kill(1), nice(1).

WARNING
Things can change while ps is running; the snap-shot it gives is only true for a split-second, and it may not be accurate by the time you see it. Some data printed for defunct processes is irrelevant.

If no termlist, proclist, uidlist, or grplist is specified, ps checks stdin, stdout, and stderr in that order, looking for the controlling terminal and will attempt to report on processes associated with the controlling terminal. In this situation, if stdin, stdout, and stderr are all redirected, ps will not find a controlling terminal, so there will be no report.
On a heavily loaded system, `ps` may report an `lseek(2)` error and exit. `ps` may seek to an invalid user area address: having got the address of a process' user area, `ps` may not be able to seek to that address before the process exits and the address becomes invalid.

`ps -ef` may not report the actual start of a tty login session, but rather an earlier time, when a `getty` was last respawned on the tty line.
NAME

pwck, grpck – password/group file checkers

SYNOPSIS

/etc/pwck [file]
/etc/grpck [file]

DESCRIPTION

*pwck* scans the password file and notes any inconsistencies. The checks include validation of the number of fields, login name, user ID, group ID, and whether the login directory and the program-to-use-as-Shell exist. The default password file is /etc/passwd.

*Grpck* verifies all entries in the group file. This verification includes a check of the number of fields, group name, group ID, and whether all login names appear in the password file. The default group file is /etc/group.

FILES

/etc/group
/etc/passwd

SEE ALSO

group(4), passwd(4).

DIAGNOSTICS

Group entries in /etc/group with no login names are flagged.
NAME

pwd – working directory name

SYNOPSIS

pwd

DESCRIPTION

pwd prints the path name of the working (current) directory.

SEE ALSO

cd(1).

DIAGNOSTICS

“Cannot open ..” and “Read error in ..” indicate possible file system trouble and should be referred to a UNIX system administrator.
NAME
ratfor – rational Fortran dialect

SYNOPSIS
ratfor [ option ... ] [ filename ... ]

DESCRIPTION
Ratfor converts a rational dialect of Fortran into ordinary irrational Fortran. Ratfor provides control flow constructs essentially identical to those in C:

statement grouping:
   { statement; statement; statement }

decision-making:
   if (condition) statement [ else statement ]
   switch (integer value) {
      case integer: statement
      ...
      [ default: ] statement
   }

loops:
   while (condition) statement
   for (expression; condition; expression) statement
   do limits statement
   repeat statement [ until (condition) ]
   break
   next

and some syntactic sugar to make programs easier to read and write:

free form input:
   multiple statements/line; automatic continuation

comments:
   # this is a comment

translation of relational:
   >, >=, etc., become .GT., .GE., etc.

return (expression)
   returns expression to caller from function

define:
   define name replacement

include:
   include filename

SEE ALSO
NAME

rc0 – run commands performed to stop the operating system

SYNOPSIS

/etc/rc0

DESCRIPTION

This file is executed at each system state change that needs to have the system in an inactive state. It is responsible for those actions that bring the system to a quiescent state, traditionally called “shutdown”.

There are three system states that require this procedure. They are state 0 (the system halt state), state 5 (the firmware state), and state 6 (the reboot state). Whenever a change to one of these states occurs, the /etc/rc0 procedure is run. The entry in /etc/inittab might read:

    s0:056:wait:/etc/rc0 >/dev/console 2>&1 </dev/console

Some of the actions performed by /etc/rc0 are carried out by files in the directory /etc/shutdown.d and files beginning with K in /etc/rc0.d. These files are executed in ASCII order (see FILES below for more information), terminating some system service. The combination of commands in /etc/rc0 and files in /etc/shutdown.d and /etc/rc0.d determines how the system is shut down.

The recommended sequence for /etc/rc0 is:

Stop System Services and Daemons.

Various system services (such as 3BNET Local Area Network or LP Spooler) are gracefully terminated.

When new services are added that should be terminated when the system is shut down, the appropriate files are installed in /etc/shutdown.d and /etc/rc0.d.

Terminate Processes

SIGTERM signals are sent to all running processes by killall(1M). Processes stop themselves cleanly if sent SIGTERM.

Kill Processes

SIGKILL signals are sent to all remaining processes; no process can resist SIGKILL.

At this point the only processes left are those associated with /etc/rc0 and processes 0 and 1, which are special to the operating system.

Unmount All File Systems

Only the root file system (/) remains mounted.

Depending on which system state the systems end up in (0, 5, or 6), the entries in /etc/inittab will direct what happens next. If the /etc/inittab has not defined any other actions to be performed as in the case of system state 0, then the operating system will have nothing to do. It should not be possible to get the system’s attention. The only thing that can be done is to turn off the power or possibly get the attention of a firmware monitor. The command can be used only by the super-user.

FILES

The execution by /bin/sh of any files in /etc/shutdown.d occurs in ascii sort-sequence order. See rc2(1M) for more information.
SEE ALSO

killall(1M), rc2(1M), shutdown(1M).
NAME
rc2 - run commands performed for multi-user environment

SYNOPSIS
/etc/rc2

DESCRIPTION
This file is executed via an entry in /etc/inittab and is responsible for those initializations that bring the system to a ready-to-use state, traditionally state 2, called the "multi-user" state.

The actions performed by /etc/rc2 are found in files in the directory /etc/rc.d and files beginning with S in /etc/rc2.d. These files are executed by /bin/sh in ASCII sort-sequence order (see FILES for more information). When functions are added that need to be initialized when the system goes multi-user, an appropriate file should be added in /etc/rc2.d.

The functions done by /etc/rc2 command and associated /etc/rc2.d files include:
  Setting and exporting the TIMEZONE variable.
  Setting-up and mounting the user (/usr) file system.
  Cleaning up (remaking) the /tmp and /usr/tmp directories.
  Loading the network interface and ports cards with program data and starting the associated processes.
  Starting the cron daemon by executing /etc/cron.
  Cleaning up (deleting) uucp locks status, and temporary files in the /usr/spool/uucp directory.

Other functions can be added, as required, to support the addition of hardware and software features.

EXAMPLES
The following are prototypical files found in /etc/rc2.d. These files are prefixed by an S and a number indicating the execution order of the files.

MOUNTFILESYS
  # Set up and mount file systems

cd /
/etc/mountall /etc/fstab

RMTTMPFILES
  # clean up /tmp
  rm -rf /tmp
  mkdir /tmp
  chmod 777 /tmp
  chgrp sys /tmp
  chown sys /tmp

uucp
  # clean-up uucp locks, status, and temporary files

  rm -rf /usr/spool/locks/*

The file /etc/TIMEZONE is included early in /etc/rc2, thus establishing the default time zone for all commands that follow.
FILES

Here are some hints about files in /etc/rc.d:

The order in which files are executed is important. Since they are executed in ASCII sort-sequence order, using the first character of the file name as a sequence indicator will help keep the proper order. Thus, files starting with the following characters would be:

[0-9]. very early
[A-Z]. early
[a-n]. later
[o-z]. last

3.mountfs

Files in /etc/rc.d that begin with a dot (.) will not be executed. This feature can be used to hide files that are not to be executed for the time being without removing them. The command can be used only by the super-user.

Files in /etc/rc2.d must begin with an S or a K followed by a number and the rest of the file name. Upon entering run level 2, files beginning with S are executed with the start option; files beginning with K, are executed with the stop option. Files beginning with other characters are ignored.

SEE ALSO

shutdown(1M).
NAME

rcp – remote file copy

SYNOPSIS

rcp [-p] file1 file2
rcp [-p] [-r] file ... directory

DESCRIPTION

Rcp copies files between machines. Each file or directory argument is either a remote file name of the form “rhost:path”, or a local file name (containing no ‘/’ characters, or a ‘/’ before any ‘:’s).

If the -r option is specified and any of the source files are directories, rcp copies each subtree rooted at that name; in this case the destination must be a directory.

By default, the mode and owner of file2 are preserved if it already existed; otherwise the mode of the source file modified by the umask(2) on the destination host is used. The -p option causes rcp to attempt to preserve (duplicate) in its copies the modification times and modes of the source files, ignoring the umask.

If path is not a full path name, it is interpreted relative to your login directory on rhost. A path on a remote host may be quoted (using \, ’, or ‘) so that the metacharacters are interpreted remotely.

Rcp does not prompt for passwords; your current local user name must exist on rhost and allow remote command execution via rsh(1C).

Rcp handles third party copies, where neither source nor target files are on the current machine. Hostnames may also take the form “rname@rhost” to use rname rather than the current user name on the remote host. The destination hostname may also take the form “rhost:rname” to support destination machines that are running 4.2BSD versions of rcp.

SEE ALSO

cp(1), ftp(1C), rsh(1C), rlogin(1C)

BUGS

Doesn’t detect all cases where the target of a copy might be a file in cases where only a directory should be legal.

Is confused by any output generated by commands in a .login, .profile, or .cshrc file on the remote host.
NAME

regcmp – regular expression compile

SYNOPSIS

regcmp [-] files

DESCRIPTION

The regcmp command performs a function similar to regcmp(3X) and, in most cases, precludes the need for calling regcmp(3X) from C programs. This saves on both execution time and program size. The command regcmp compiles the regular expressions in file and places the output in file.c. If the – option is used, the output will be placed in file.c. The format of entries in file is a name (C variable) followed by one or more blanks followed by a regular expression enclosed in double quotes. The output of regcmp is C source code. Compiled regular expressions are represented as extern char vectors. File.i files may thus be included in C programs, or file.c files may be compiled and later loaded. In the C program which uses the regcmp output, `regex(abc, line)` will apply the regular expression named abc to line. Diagnostics are self-explanatory.

EXAMPLES

name "([A-Za-z][A-Za-z0-9_.]*)$0"

telno "((0,1)([2-9][01][1-9])$0\[0,1]*"

"(2-9)[0-9-](2)$1[-][0,1]"

"0-[9][4]$2"

In the C program that uses the regcmp output,

regex(telno, line, area, exch, rest)

will apply the regular expression named telno to line.

SEE ALSO

regcmp(3X).
User Commands

REV(1)

NAME
rev - reverse lines of a file

SYNOPSIS
rev [ file ]...

DESCRIPTION
Rev copies the named files to the standard output, reversing the order of characters in every line. If no file is specified, the standard input is copied.
NAME
rlogin – remote login

SYNOPSIS
rlogin rhost [ -e c ] [ -8 ] [ -L ] [ -l username ]
rhost [ -e c ] [ -8 ] [ -L ] [ -l username ]

DESCRIPTION
Rlogin connects your terminal on the current local host system lhost to the remote
host system rhost.

Each host has a file /etc/hosts.equiv which contains a list of rhost's with which it shares
account names. (The host names must be the standard names as described in rsh(1C).)
When you rlogin as the same user on an equivalent host, you don't need to give a
password. Each user may also have a private equivalence list in a file .rhosts in his
login directory. Each line in this file should contain an rhost and a username
separated by a space, giving additional cases where logins without passwords are to
be permitted. If the originating user is not equivalent to the remote user, then a login
and password will be prompted for on the remote machine as in login(1). To avoid
some security problems, the .rhosts file must be owned by either the remote user or
root.

The remote terminal type is the same as your local terminal type (as given in your
environment TERM variable). The terminal or window size is also copied to the
remote system if the server supports the option, and changes in size are reflected as
well. All echoing takes place at the remote site, so that (except for delays) the rlogin
is transparent. Flow control via ^S and ^Q and flushing of input and output on
interrupts are handled properly. The optional argument -8 allows an eight-bit input
data path at all times; otherwise parity bits are stripped except when the remote
side's stop and start characters are other than ^S/^Q. The argument -L allows the
rlogin session to be run in litout mode. A line of the form "^Z" disconnects from the
remote host, where "^Z" is the escape character. Similarly, the line "^Z" (where ^Z,
control-Z, is the suspend character) will suspend the rlogin session. Substitution of
the delayed-suspend character (normally ^Y) for the suspend character suspends the
send portion of the rlogin, but allows output from the remote system. A different
escape character may be specified by the -e option. There is no space separating this
option flag and the argument character.

SEE ALSO
rsh(1C)

FILES
/usr/hosts/* for rhost version of the command

BUGS
More of the environment should be propagated.
NAME

rm, rmdir – remove files or directories

SYNOPSIS

rm [-f] [-i] file ...
rm -r [-f] [-i] dirname ... [file ...]
rmdir [-p] [-s] dirname ...

DESCRIPTION

rm removes the entries for one or more files from a directory. If an entry was the last link to the file, the file is destroyed. Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

If a file has no write permission and the standard input is a terminal, the full set of permissions (in octal) for the file are printed followed by a question mark. This is a prompt for confirmation. If the answer begins with y (for yes), the file is deleted, otherwise the file remains.

Note that if the standard input is not a terminal, the command will operate as if the -f option is in effect.

rmdir removes the named directories, which must be empty.

Three options apply to rm:

-f This option causes the removal of all files (whether write-protected or not) in a directory without prompting the user. In a write-protected directory, however, files are never removed (whatever their permissions are), but no messages are displayed. If the removal of a write-protected directory was attempted, this option cannot suppress an error message.

-r This option causes the recursive removal of any directories and subdirectories in the argument list. The directory will be emptied of files and removed. Note that the user is normally prompted for removal of any write-protected files which the directory contains. The write-protected files are removed without prompting, however, if the -f option is used, or if the standard input is not a terminal and the -i option is not used.

Note that the -r option does not cause rm to follow symbolic links to directories.

If the removal of a non-empty, write-protected directory was attempted, the command will always fail (even if the -f option is used), resulting in an error message.

-i With this option, confirmation of removal of any write-protected file occurs interactively. It overrides the -f option and remains in effect even if the standard input is not a terminal.

Two options apply to rmdir:

-p This option allows users to remove the directory dirname and its parent directories which become empty. A message is printed on standard output as to whether the whole path is removed or part of the path remains for some reason.

-s This option is used to suppress the message printed on standard error when -p is in effect.
DIAGNOSTICS

All messages are generally self-explanatory.
It is forbidden to remove the files "." and "." in order to avoid the consequences of
inadvertently doing something like the following:

    rm -r *

Both `rm` and `rmdir` return exit codes of 0 if all the specified directories are removed
successfully. Otherwise, they return a non-zero exit code.

SEE ALSO

`unlink(2)`, `rmdir(2)`.
NAME

rmdel – remove a delta from an SCCS file

SYNOPSIS

rmdel -rSId files

DESCRIPTION

rmdel removes the delta specified by the SId from each named SCCS file. The delta to be removed must be the newest (most recent) delta in its branch in the delta chain of each named SCCS file. In addition, the specified must not be that of a version being edited for the purpose of making a delta (i.e., if a p-file [see get(1)] exists for the named SCCS file, the specified must not appear in any entry of the p-file).

The -r option is used for specifying the SId (SCCS IDentification) level of the delta to be removed.

If a directory is named, rmdel behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with .) and unreadable files are silently ignored. If a name of - is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed; non-SCCS files and unreadable files are silently ignored.

Simply stated, they are either (1) if you make a delta you can remove it; or (2) if you own the file and directory you can remove a delta.

FILES

x.file  [see delta(1)]
z.file  [see delta(1)]

SEE ALSO

delta(1), get(1), help(1), prs(1), sccsfile(4).

DIAGNOSTICS

Use help(1) for explanations.
NAME
rs170 – turn on/off rs170 video mode

SYNOPSIS
rs170 on | off

DESCRIPTION
rs170 on sets the graphics subsystem into rs170 mode, and rs170 off restores the video to high-resolution. In rs170 mode, the upper left 640 by 480 pixels will be visible on a low resolution RGB monitor.

It is recommended that "rs170 off" be bound to some key/button combination under the window system, such as <META><SHIFT><CONTROL><Right-Button>. This way, independent of where the cursor is or the state of the window system, the monitor can be returned to high-resolution mode.
NAME
rsh – remote shell

SYNOPSIS
rsh host [-l username] [-n] command
host [-l username] [-n] command

DESCRIPTION
Rsh connects to the specified host, and executes the specified command. Rsh copies its
standard input to the remote command, the standard output of the remote command
to its standard output, and the standard error of the remote command to its standard
error. Interrupt, quit and terminate signals are propagated to the remote command;
rsh normally terminates when the remote command does.

The remote username used is the same as your local username, unless you specify a
different remote name with the -l option. This remote name must be equivalent (in
the sense of rlogin(1C)) to the originating account; no provision is made for specifying
a password with a command.

If you omit command, then instead of executing a single command, you will be logged
in on the remote host using rlogin(1C).

Shell metacharacters which are not quoted are interpreted on local machine, while
quoted metacharacters are interpreted on the remote machine. Thus the command

rsh otherhost cat remotefile >> localfile

appends the remote file remotefile to the localfile localfile, while

rsh otherhost cat remotefile ">>" otherremotefile

appends remotefile to otherremotefile.

Host names are given in the file /etc/hosts. Each host has one standard name (the first
name given in the file), which is rather long and unambiguous, and optionally one or
more nicknames. The host names for local machines are also commands in the direc-
tory /usr/hosts; if you put this directory in your search path then the rsh can be omit-
ted.

FILES
/etc/hosts
/usr/hosts/*

SEE ALSO
rlogin(1C)

BUGS
If you are using csh(1) and put a rsh(1C) in the background without redirecting its
input away from the terminal, it will block even if no reads are posted by the remote
command. If no input is desired you should redirect the input of rsh to /dev/null
using the -n option.

You cannot run an interactive command (like vi(1)); use rlogin(1C).

Stop signals stop the local rsh process only; this is arguably wrong, but currently
hard to fix for reasons too complicated to explain here.
NAME
runacct – run daily accounting

SYNOPSIS
/usr/lib/acct/runacct [mmdd [state]]

DESCRIPTION
runacct is the main daily accounting shell procedure. It is normally initiated via
cron(1M). runacct processes connect, fee, disk, and process accounting files. It also
prepares summary files for prdaily or billing purposes. runacct is distributed only to
source code licensees.

runacct takes care not to damage active accounting files or summary files in the event
of errors. It records its progress by writing descriptive diagnostic messages into
active. When an error is detected, a message is written to /dev/console, mail (see
mail(1)) is sent to root and adm, and runacct terminates. runacct uses a series of lock
files to protect against re-invocation. The files lock and lock1 are used to prevent
simultaneous invocation, and lastdate is used to prevent more than one invocation
per day.

runacct breaks its processing into separate, restartable states using statefile to
remember the last state completed. It accomplishes this by writing the state name
into statefile. runacct then looks in statefile to see what it has done and to determine
what to process next. states are executed in the following order:

SETUP Move active accounting files into working files.

WTMPFIX Verify integrity of wtmp file, correcting date changes if necessary.

CONNECT1 Produce connect session records in ctmp.h format.

CONNECT2 Convert ctmp.h records into tacct.h format.

PROCESS Convert process accounting records into tacct.h format.

MERGE Merge the connect and process accounting records.

FEES Convert output of chargefee into tacct.h format and merge with
connect and process accounting records.

DISK Merge disk accounting records with connect, process, and fee
accounting records.

MERGETACCT Merge the daily total accounting records in daytacct with the
summary total accounting records in /usr/adm/acct/sum/tacct.

CMS Produce command summaries.

USEREXIT Any installation-dependent accounting programs can be
included here.

CLEANUP Cleanup temporary files and exit.

To restart runacct after a failure, first check the active file for diagnostics, then fix up
any corrupted data files such as pacct or wtmp. The lock files and lastdate file must be
removed before runacct can be restarted. The argument mmdd is necessary if runacct
is being restarted, and specifies the month and day for which runacct will rerun the
accounting. Entry point for processing is based on the contents of statefile; to over-
ride this, include the desired state on the command line to designate where process-
ing should begin.
EXAMPLES

To start runacct. nohup runacct 2> /usr/adm/acct/nite/fd2log &
To restart runacct. nohup runacct 0601 2>> /usr/adm/acct/nite/fd2log &
To restart runacct at a specific state. nohup runacct 0601 MERGE 2>> /usr/adm/acct/nite/fd2log &

FILES

/etc/wtmp
/usr/adm/pacct*
/usr/src/cmd/acct/tacct.h
/usr/src/cmd/acct/ctmp.h
/usr/adm/acct/nite/active
/usr/adm/acct/nite/daytacct
/usr/adm/acct/nite/lock
/usr/adm/acct/nite/lock1
/usr/adm/acct/nite/lastdate
/usr/adm/acct/nite/statefile
/usr/adm/acct/nite/ptacct*.mmd

SEE ALSO

acct(1M), acct(2), acct(4), acctcms(1M), acctcom(1), acctcon(1M), acctmerge(1M), acctprc(1M), acctsh(1M), cron(1M), fwtmp(1M), mail(1), utmp(4)

BUGS

Normally it is not a good idea to restart runacct in the SETUP state. Run SETUP manually and restart via:

    runacct mmd -d WTMPFIX

If runacct failed in the PROCESS state, remove the last ptacct file because it will not be complete.
NAME
runtime – show host status of local machines

SYNOPSIS
runtime [-a] [-r] [-l] [-t] [-u]

DESCRIPTION
runtime gives a status line like uptime for each machine on the local network; these are formed from packets broadcast by each host on the network once a minute.

Machines for which no status report has been received for 11 minutes are shown as being down.

Users idle an hour or more are not counted unless the -a flag is given.

Normally, the listing is sorted by host name. The -l, -t, and -u flags specify sorting by load average, uptime, and number of users, respectively. The -r flag reverses the sort order.

FILES
/usr/spool/rwho/whod.* data files

SEE ALSO
rwho(1C)
NAME
rwho – who’s logged in on local machines

SYNOPSIS
rwho [-a]

DESCRIPTION
The rwho command produces output similar to who, but for all machines on the local network. If no report has been received from a machine for 5 minutes then rwho assumes the machine is down, and does not report users last known to be logged into that machine.

If a users hasn’t typed to the system for a minute or more, then rwho reports this idle time. If a user hasn’t typed to the system for an hour or more, then the user will be omitted from the output of rwho unless the -a flag is given.

FILES
/usr/spool/rwho/whod.* information about other machines

SEE ALSO
ruptime(1C), rwhod(8C)

BUGS
This is unwieldly when the number of machines on the local net is large.
NAME

sact – print current SCCS file editing activity

SYNOPSIS

sact files

DESCRIPTION

sact informs the user of any impending deltas to a named SCCS file. This situation occurs when get(1) with the –e option has been previously executed without a subsequent execution of delta(1). If a directory is named on the command line, sact behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of – is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

The output for each named file consists of five fields separated by spaces.

Field 1 specifies the SID of a delta that currently exists in the SCCS file to which changes will be made to make the new delta.

Field 2 specifies the SID for the new delta to be created.

Field 3 contains the logname of the user who will make the delta (i.e., executed a get for editing).

Field 4 contains the date that get –e was executed.

Field 5 contains the time that get –e was executed.

SEE ALSO

delta(1), get(1), unget(1).

DIAGNOSTICS

Use help(1) for explanations.
NAME
sar – system activity reporter

SYNOPSIS
sar [-ubdycvaqymprA][-o file][n]
sar [-ubdycvaqymprA][-s time][-e time][-i sec][-f file]

DESCRIPTION
sar, in the first instance, samples cumulative activity counters in the operating system at n intervals of t seconds, where t should be 5 or greater. If the -o option is specified, it saves the samples in file in binary format. The default value of n is 1. In the second instance, with no sampling interval specified, sar extracts data from a previously recorded file, either the one specified by -f option or, by default, the standard system activity daily data file /usr/adm/sa/sadd for the current day dd. The starting and ending times of the report can be bounded via the -s and -e time arguments of the form hh:mm:ss. The -i option selects records at sec second intervals. Otherwise, all intervals found in the data file are reported.

In either case, subsets of data to be printed are specified by option:

-u Report CPU utilization (the default):
  %usr, %sys, %wio, %idle – portion of time running in user mode, running in
  system mode, idle with some process waiting for block I/O, and otherwise idle.
  When used with -D, %sys is split into percent of time servicing requests from
  remote machines (%sys remote) and all other system time (%sys local).

-b Report buffer activity:
  bread/s, bwrit/s – transfers per second of data between system buffers and
  disk or other block devices;
  lread/s, lwrit/s – accesses of system buffers;
  %rcache, %wcache – cache hit ratios, i.e., (1–bread/lread) as a percentage;
  pread/s, pwrit/s – transfers via raw (physical) device mechanism.

-d Report activity for each block device, e.g., disk or tape drive. When data is
  displayed, the device specification disk is generally used to represent a disk
  drive. The device specification used to represent a tape drive is machine
  dependent. The activity data reported is:
  %busy, avque – portion of time device was busy servicing a transfer request,
  average number of requests outstanding during that time;
  r+w/s, blks/s – number of data transfers from or to device, number of bytes
  transferred in 512-byte units;
  avwait, avserv – average time in ms. that transfer requests wait idly on queue,
  and average time to be serviced (which for disks includes seek, rotational
  latency and data transfer times).

-y Report TTY device activity:
  rawch/s, canch/s, outhc/s – input character rate, input character rate pro-
  cessed by canon, output character rate;
  rcvin/s, xmtin/s, madmin/s – receive, transmit and modem interrupt rates.

-c Report system calls:
  scall/s – system calls of all types;
  sread/s, swrit/s, fork/s, exec/s – specific system calls;
  rchar/s, wchar/s – characters transferred by read and write system calls.
  When used with -D, the system calls are split into incoming, outgoing, and
  strictly local calls.
-w Report system swapping and switching activity:
   swpin/s, swpot/s, bswin/s, bswot/s – number of transfers and number of
   512-byte units transferred for swapins and swapouts (including initial loading
   of some programs);
   pswch/s – process switches.

-a Report use of file access system routines:
   igin/s, namei/s, dirblk/s.

-q Report average queue length while occupied, and % of time occupied:
   runq-sz, %runocc – run queue of processes in memory and runnable;
   swpq-sz, %swpocc – swap queue of processes swapped out but ready to run.

-v Report status of process, i-node, file tables:
   text-sz, proc-sz, ind-sz, file-sz, lock-sz – entries/size for each table, evaluated
   once at sampling point;
   ov – overflows that occur between sampling points for each table.

-m Report message and semaphore activities:
   msg/s, sema/s – primitives per second.

-p Report paging activities:
   vflt/s – address translation page faults (valid page not in memory);
   pflt/s – page faults from protection errors (illegal access to page) or "copy-on-
           writes";
   pgfil/s – vflt/s satisfied by page-in from file system;
   rclm/s – valid pages reclaimed for free list.

-r Report unused memory pages and disk blocks:
   freemem – average pages available to user processes;
   freeswap – disk blocks available for process swapping.

-A Report all data. Equivalent to -udqbwcaayvmp.

EXAMPLES
To see today’s CPU activity so far:
   sar
To watch CPU activity evolve for 10 minutes and save data:
   sar -o temp 60 10
To later review disk and tape activity from that period:
   sar -d -f temp

FILES
/usr/adm/sa/sadd
daily data file, where dd are digits representing the day of the
month.

SEE ALSO
sar(1M)
SA1, SA2, SADC – system activity report package

SYNOPSIS

/usr/lib/sa/sadc [t n] [ofile]
/usr/lib/sa/sa1 [t n]
/usr/lib/sa/sa2 [-ubdycwqaqvmpsrDA] [-s time] [-e time] [-i sec]

DESCRIPTION

System activity data can be accessed at the special request of a user (see sar(1)) and automatically on a routine basis as described here. The operating system contains a number of counters that are incremented as various system actions occur. These include counters for CPU utilization, buffer usage, disk and tape I/O activity, TTY device activity, switching and system-call activity, file-access, queue activity, interprocess communications, paging and Remote File Sharing.

SADC and shell procedures, SA1 and SA2, are used to sample, save, and process this data.

SADC, the data collector, samples system data n times every t seconds and writes in binary format to ofile or to standard output. If t and n are omitted, a special record is written. This facility is used at system boot time, when booting to a multiuser state, to mark the time at which the counters restart from zero. For example, the /etc/init.d/perf file writes the restart mark to the daily data by the command entry:

    su sys -c "/usr/lib/sa/sadc /usr/adm/sa/SA\ date +%d%"

The shell script SA1, a variant of SADC, is used to collect and store data in binary file /usr/adm/sa/sadd where dd is the current day. The arguments t and n cause records to be written n times at an interval of t seconds, or once if omitted. The entries in /usr/spool/cron/crontabs/sys (see cron(1M)):

    0 * * * 0-6 /usr/lib/sa/sa1
    20,40 8-17 * * 1-5 /usr/lib/sa/sa1

will produce records every 20 minutes during working hours and hourly otherwise.

The shell script SA2, a variant of SAR(1), writes a daily report in file /usr/adm/sa/sardd. The options are explained in sar(1). The /usr/spool/cron/crontabs/sys entry:

    5 18 * * 1-5 /usr/lib/sa/sa2 -s 8:00 -e 18:01 -i 1200 -A

will report important activities hourly during the working day.

The structure of the binary daily data file is:

    struct sa {
    struct sysinfo si; /* see /usr/include/sys/sysinfo.h */
    struct minfo mi; /* defined in sys/sysinfo.h */
    struct dinfo di; /* RFS info defined in sys/sysinfo.h */
    int minserve, maxserve; /* RFS server low and high water marks */
    int szinode; /* current size of inode table */
    int szfile; /* current size of file table */
    int zproc; /* current size of proc table */
    int szckf; /* current size of file record header table */
    int szckr; /* current size of file record lock table */
    int mzninode; /* size of inode table */
    int mzsfile; /* size of file table */
}
int mszproc; /* size of proc table */
int mszlkf;  /* maximum size of file record header table */
int mszlkcr; /* maximum size of file record lock table */
long inodeovf; /* cumulative overflows of inode table */
long fileovf; /* cumulative overflows of file table */
long procovf; /* cumulative overflows of proc table */
time_t ts;    /* time stamp, seconds */
long devio[NDEVS][4];    /* device unit information */
#define IO_OPS   0 /* cumulative I/O requests */
#define IO_BCNT  1 /* cumulative blocks transferred */
#define IO_ACT   2 /* cumulative drive busy time in ticks */
#define IO_RESP  3 /* cumulative I/O resp time in ticks */

FILES

/usr/adm/sa/sadd   daily data file
/usr/adm/sa/sardd  daily report file
/tmp/sa.adrfl     address file

SEE ALSO
cron(1M), sag(1G), sar(1), timex(1).
NAME
sccs – front end for the SCCS subsystem

SYNOPSIS
sccs [-r] [-dpath] [-ppath] command [flags] [args]

DESCRIPTION
Sccs is a front end to the SCCS programs that helps them mesh more cleanly with the rest of UNIX. It also includes the capability to run “set user id” to another user to provide additional protection.

Basically, sccs runs the command with the specified flags and args. Each argument is normally modified to be prepended with “SCCS/s.”.

Flags to be interpreted by the sccs program must be before the command argument. Flags to be passed to the actual SCCS program must come after the command argument. These flags are specific to the command and are discussed in the documentation for that command.

Besides the usual SCCS commands, several “pseudo-commands” can be issued. These are:

edit  Equivalent to “get –e”.
delget Perform a delta on the named files and then get new versions. The new versions will have id keywords expanded, and will not be editable. The -m, -p, -r, -s, and -y flags will be passed to delta, and the -b, -c, -e, -i, -k, -l, -s, and -x flags will be passed to get.
deledit Equivalent to “delget” except that the “get” phase includes the “-e” flag. This option is useful for making a “checkpoint” of your current editing phase. The same flags will be passed to delta as described above, and all the flags listed for “get” above except -e and -k are passed to “edit”.
create Creates an SCCS file, taking the initial contents from the file of the same name. Any flags to “admin” are accepted. If the creation is successful, the files are renamed with a comma on the front. These should be removed when you are convinced that the SCCS files have been created successfully.
fix Must be followed by a -r flag. This command essentially removes the named delta, but leaves you with a copy of the delta with the changes that were in it. It is useful for fixing small compiler bugs, etc. Since it doesn’t leave audit trails, it should be used carefully.
clean This routine removes everything from the current directory that can be recreated from SCCS files. It will not remove any files being edited. If the -b flag is given, branches are ignored in the determination of whether they are being edited; this is dangerous if you are keeping the branches in the same directory.
unedit This is the opposite of an “edit” or a “get –e”. It should be used with extreme caution, since any changes you made since the get will be irretrievably lost.
info Gives a listing of all files being edited. If the -b flag is given, branches (i.e., SID's with two or fewer components) are ignored. If the -u flag is given (with an optional argument) then only files being edited by you (or the named user) are listed.
check

Like “info” except that nothing is printed if nothing is being edited, and a non-zero exit status is returned if anything is being edited. The intent is to have this included in an “install” entry in a makefile to insure that everything is included into the SCCS file before a version is installed.

tell

Gives a newline-separated list of the files being edited on the standard output. Takes the -b and -u flags like “info” and “check”.

diffs

Gives a “diff” listing between the current version of the program(s) you have out for editing and the versions in SCCS format. The -r, -c, -l, -x, and -t flags are passed to get; the -l, -s, -e, -f, -h, and -b options are passed to diff. The -C flag is passed to diff as -c.

print

This command prints out verbose information about the named files.

The -r flag runs sccs as the real user rather than as whatever effective user sccs is “set user id” to. The -d flag gives a root directory for the SCCS files. The default is the current directory. The -p flag defines the pathname of the directory in which the SCCS files will be found; “SCCS” is the default. The -p flag differs from the -d flag in that the -d argument is prepended to the entire pathname and the -p argument is inserted before the final component of the pathname. For example, “sccs -d/x -py get a/b” will convert to “get /x/a/y/s.b”. The intent here is to create aliases such as “alias sysccs sccs -d/usr/src” which will be used as “sysccs get cmd/who.c”. Also, if the environment variable PROJECT is set, its value is used to determine the -d flag. If it begins with a slash, it is taken directly; otherwise, the home directory of a user of that name is examined for a subdirectory “src” or “source”. If such a directory is found, it is used.

Certain commands (such as admin) cannot be run “set user id” by all users, since this would allow anyone to change the authorizations. These commands are always run as the real user.

EXAMPLES

To get a file for editing, edit it, and produce a new delta:

    sccs get -e file.c
    ex file.c
    sccs delta file.c

To get a file from another directory:

    sccs -p/usr/src/sccs/s. get cc.c

or

    sccs get /usr/src/sccs/s.cc

To make a delta of a large number of files in the current directory:

    sccs delta *.c

To get a list of files being edited that are not on branches:

    sccs info -b

To delta everything being edited by you:

    sccs delta `sccs tell -u`

In a makefile, to get source files from an SCCS file if it does not already exist:

    SRCS = <list of source files>
    $(SRCS):
      sccs get $(REL) $@
SEE ALSO

admin(SCCS), chghist(SCCS), comb(SCCS), delta(SCCS), get(SCCS), help(SCCS),
prt(SCCS), rmdel(SCCS), sccsdiff(SCCS), what(SCCS)
Eric Allman, *An Introduction to the Source Code Control System*

BUGS

It should be able to take directory arguments on pseudo-commands like the SCCS
commands do.
NAME
sccsdiff – compare two versions of an SCCS file

SYNOPSIS
sccsdiff -rSID1 -rSID2 [-p] [-sn] files

DESCRIPTION
sccsdiff compares two versions of an SCCS file and generates the differences between the two versions. Any number of SCCS files may be specified, but arguments apply to all files.

- rSID?
  SID1 and SID2 specify the deltas of an SCCS file that are to be compared. Versions are passed to bdiff(1) in the order given.

- p
  pipe output for each file through pr(1).

- sn
  n is the file segment size that bdiff will pass to diff(1). This is useful when diff fails due to a high system load.

FILES
/tmp/get????? Temporary files

SEE ALSO
bdiff(1), get(1), help(1), pr(1).

DIAGNOSTICS
"file: No differences" If the two versions are the same.
Use help(1) for explanations.
NAME
screendump – dump the screen display

SYNOPSIS
screendump [ -n ]

DESCRIPTION
screendump captures the display and outputs an image file to standard output. This image file may be output to the Tektronix printer through the lp(1) command using printer "TEK".

screendump knows all about pseudo and true color and double buffer windows, and captures the screen as it appears to the user. Use xwd(1) to capture an individual window.

By default, screendump compresses the output image using a simple run-length encoding scheme. This compression may be suppressed with the -n switch.

WARNING
An uncompressed screendump is nearly 4 megabytes large.

SEE ALSO
lp(1), xwd(1)
NAME

script – make typescript of terminal session

SYNOPSIS

script [-a] [ file ]

DESCRIPTION

Script makes a typescript of everything printed on your terminal. The typescript is written to file, or appended to file if the -a option is given. It can be sent to the line printer later with lpr. If no file name is given, the typescript is saved in the file typescript.

The script ends when the forked shell exits.

This program is useful when using a crt and a hard-copy record of the dialog is desired, as for a student handing in a program that was developed on a crt when hard-copy terminals are in short supply.

BUGS

Script places everything in the log file. This is not what the naive user expects.
NAME
sdiff – side-by-side difference program

SYNOPSIS
sdiff [ options ... ] file1 file2

DESCRIPTION
sdiff uses the output of diff(1) to produce a side-by-side listing of two files indicating those lines that are different. Each line of the two files is printed with a blank gutter between them if the lines are identical, a < in the gutter if the line only exists in file1, a > in the gutter if the line only exists in file2, and a l for lines that are different.

For example:

```
x   y
a   a
b <
c <
d   d
    >
```

The following options exist:

- **-w n** Use the next argument, n, as the width of the output line. The default line length is 130 characters.

- **-l** Only print the left side of any lines that are identical.

- **-s** Do not print identical lines.

- **-o output** Use the next argument, output, as the name of a third file that is created as a user-controlled merging of file1 and file2. Identical lines of file1 and file2 are copied to output. Sets of differences, as produced by diff(1), are printed; where a set of differences share a common gutter character. After printing each set of differences, sdiff prompts the user with a % and waits for one of the following user-typed commands:

```
  l append the left column to the output file
  r append the right column to the output file
  s turn on silent mode; do not print identical lines
  v turn off silent mode
  e l call the editor with the left column
  e r call the editor with the right column
  e b call the editor with the concatenation of left and right
  e call the editor with a zero length file
  q exit from the program
```

On exit from the editor, the resulting file is concatenated on the end of the output file.

SEE ALSO
diff(1), ed(1).
NAME

sed - stream editor

SYNOPSIS

sed [ -n ] [ -e script ] [ -f sfile ] [ files ]

DESCRIPTION

sed copies the named files (standard input default) to the standard output, edited according to a script of commands. The -f option causes the script to be taken from file sfile; these options accumulate. If there is just one -e option and no -f options, the flag -e may be omitted. The -n option suppresses the default output. A script consists of editing commands, one per line, of the following form:

[ address [, address ] ] function [ arguments ]

In normal operation, sed cyclically copies a line of input into a pattern space (unless there is something left after a D command), applies in sequence all commands whose addresses select that pattern space, and at the end of the script copies the pattern space to the standard output (except under -n) and deletes the pattern space.

Some of the commands use a hold space to save all or part of the pattern space for subsequent retrieval.

An address is either a decimal number that counts input lines cumulatively across files, a $ that addresses the last line of input, or a context address, i.e., a regular expression in the style of ed(1) modified thus:

In a context address, the construction \?regular expression?, where ? is any character, is identical to regular expression/. Note that in the context address \xabc\xdefx, the second x stands for itself, so that the regular expression is abcd ef.

The escape sequence \n matches a new-line embedded in the pattern space.

A period . matches any character except the terminal new-line of the pattern space.

A command line with no addresses selects every pattern space.

A command line with one address selects each pattern space that matches the address.

A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line is selected.) Thereafter the process is repeated, looking again for the first address.

Editing commands can be applied only to non-selected pattern spaces by use of the negation function ! (below).

In the following list of functions the maximum number of permissible addresses for each function is indicated in parentheses.

The text argument consists of one or more lines, all but the last of which end with \ to hide the new-line. Backslashes in text are treated like backslashes in the replacement string of an s command, and may be used to protect initial blanks and tabs against the stripping that is done on every script line. The rfile or wfile argument must terminate the command line and must be preceded by exactly one blank. Each wfile is created before processing begins. There can be at most 10 distinct wfile arguments.

(1) a\text

Append. Place text on the output before reading the next input line.
(2) b label Branch to the : command bearing the label. If label is empty, branch to the end of the script.

(2) c\text
Change. Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, place text on the output. Start the next cycle.

(2) d Delete the pattern space. Start the next cycle.

(2) D Delete the initial segment of the pattern space through the first new-line. Start the next cycle.

(2) g Replace the contents of the pattern space by the contents of the hold space.

(2) G Append the contents of the hold space to the pattern space.

(2) h Replace the contents of the hold space by the contents of the pattern space.

(2) H Append the contents of the pattern space to the hold space.

(1) i\text
Insert. Place text on the standard output.

(2) l List the pattern space on the standard output in an unambiguous form.
Non-printing characters are spelled in two-digit ASCII and long lines are folded.

(2) n Copy the pattern space to the standard output. Replace the pattern space with the next line of input.

(2) N Append the next line of input to the pattern space with an embedded new-line. (The current line number changes.)

(2) p Print. Copy the pattern space to the standard output.

(2) P Copy the initial segment of the pattern space through the first new-line to the standard output.

(1) q Quit. Branch to the end of the script. Do not start a new cycle.

(2) r rfile Read the contents of rfile. Place them on the output before reading the next input line.

(2) s/regular expression/replacement/flags
Substitute the replacement string for instances of the regular expression in the pattern space. Any character may be used instead of /. For a fuller description see ed(1). Flags is zero or more of:

n n=1 - 512. Substitute for just the n th occurrence of the regular expression.

G Global. Substitute for all nonoverlapping instances of the regular expression rather than just the first one.

p Print the pattern space if a replacement was made.

w wfile
Write. Append the pattern space to wfile if a replacement was made.

(2) t label Test. Branch to the : command bearing the label if any substitutions have been made since the most recent reading of an input line or execution of a t. If label is empty, branch to the end of the script.

(2) w wfile Write. Append the pattern space to wfile.

(2) x Exchange the contents of the pattern and hold spaces.

(2) y/string1/string2/
Transform. Replace all occurrences of characters in string1 with the corresponding character in string2. The lengths of string1 and string2 must be equal.

(2) ! function
Don't. Apply the function (or group, if function is !) only to lines not selected by the address(es).
(0): label  This command does nothing; it bears a label for b and t commands to branch to.

(1)= Place the current line number on the standard output as a line.
(2){ Execute the following commands through a matching } only when the pattern space is selected.
(0) An empty command is ignored.
(0)# If a # appears as the first character on the first line of a script file, then that entire line is treated as a comment, with one exception. If the character after the # is an 'n', then the default output will be suppressed. The rest of the line after #n is also ignored. A script file must contain at least one non-comment line.

SEE ALSO

awk(1), ed(1), grep(1).
NAME
setmnt – establish mount table

SYNOPSIS
/etc/setmnt

DESCRIPTION
setmnt creates the /etc/mnttab table which is needed for both the mount(1M) and umount commands. setmnt reads standard input and creates a mnttab entry for each line. Input lines have the format:

filesystem node

where filesystem is the name of the file system's special file (e.g., /dev/dsk/c?d?s?) and node is the root name of that file system. Thus filesystem and node become the first two strings in the mount table entry.

FILES
/etc/mnttab

SEE ALSO
mount(1M).

BUGS
Problems may occur if filesystem or node are longer than 32 characters. setmnt silently enforces an upper limit on the maximum number of mnttab entries.
NAME

sh, rsh – shell, the standard/restricted command programming language

SYNOPSIS

sh [ -acefhiknrstuvx ] [ args ]
/lib/rsh [ -acefhiknrstuvx ] [ args ]

DESCRIPTION

sh is a command programming language that executes commands read from a terminal or a file. rsh is a restricted version of the standard command interpreter sh; it is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. See "Invocation" below for the meaning of arguments to the shell.

Definitions

A blank is a tab or a space. A name is a sequence of letters, digits, or underscores beginning with a letter or underscore. A parameter is a name, a digit, or any of the characters *, @, #, ?, -, $, and !.

Commands

A simple-command is a sequence of non-blank words separated by blanks. The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see exec(2)). The value of a simple-command is its exit status if it terminates normally, or (octal) 200+status if it terminates abnormally (see signal(2) for a list of status values).

A pipeline is a sequence of one or more commands separated by |. The standard output of each command but the last is connected by a pipe(2) to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate. The exit status of a pipeline is the exit status of the last command.

A list is a sequence of one or more pipelines separated by ; & & & & or ||, and optionally terminated by ; or &. Of these four symbols, ; and & & have equal precedence, which is lower than that of & & and ||. The symbols & & and || also have equal precedence. A semicolon ; causes sequential execution of the preceding pipeline; an ampersand (&) causes asynchronous execution of the preceding pipeline (i.e., the shell does not wait for that pipeline to finish). The symbol & & ( || ) causes the list following it to be executed only if the preceding pipeline returns a zero (non-zero) exit status. An arbitrary number of new-lines may appear in a list, instead of semicolons, to delimit commands.

A command is either a simple-command or one of the following. Unless otherwise stated, the value returned by a command is that of the last simple-command executed in the command.

for name [ in word ... ] do list done

Each time a for command is executed, name is set to the next word taken from the in word list. If in word ... is omitted, then the for command executes the do list once for each positional parameter that is set (see Parameter Substitution below). Execution ends when there are no more words in the list.

case word in [ pattern [ | pattern ... ] list ] ... esac

A case command executes the list associated with the first pattern that matches word. The form of the patterns is the same as that used for file-name generation (see "File Name Generation") except that a slash, a leading dot, or a dot immediately following a slash need not be matched explicitly.
if list then list [ elif list then list ] ... [ else list ] fi
The list following if is executed and, if it returns a zero exit status, the list following the first then is executed. Otherwise, the list following elif is executed and, if its value is zero, the list following the next then is executed. Failing that, the else list is executed. If no else list or then list is executed, then the if command returns a zero exit status.

while list do list done
A while command repeatedly executes the while list and, if the exit status of the last command in the list is zero, executes the do list; otherwise the loop terminates. If no commands in the do list are executed, then the while command returns a zero exit status; until may be used in place of while to negate the loop termination test.

(list)
Execute list in a sub-shell.

{list}
list is executed in the current (that is, parent) shell.

name () {list}
Define a function which is referenced by name. The body of the function is the list of commands between { and }. Execution of functions is described below (see Execution).

The following words are only recognized as the first word of a command and when not quoted:

if then else elif fi case esac for while until do done { ]

Comments
A word beginning with # causes that word and all the following characters up to a new-line to be ignored.

Command Substitution
The shell reads commands from the string between two grave accents (` ``) and the standard output from these commands may be used as all or part of a word. Trailing new-lines from the standard output are removed.

No interpretation is done on the string before the string is read, except to remove backslashes (\) used to escape other characters. Backslashes may be used to escape a grave accent (``) or another backslash (\) and are removed before the command string is read. Escaping grave accents allows nested command substitution. If the command substitution lies within a pair of double quotes (" ... ` ... "), a backslash used to escape a double quote (\") will be removed; otherwise, it will be left intact.

If a backslash is used to escape a new-line character (\new-line), both the backslash and the new-line are removed (see the later section on "Quoting"). In addition, backslashes used to escape dollar signs (\$) are removed. Since no interpretation is done on the command string before it is read, inserting a backslash to escape a dollar sign has no effect. Backslashes that precede characters other than \, `", new-line, and $ are left intact when the command string is read.

Parameter Substitution
The character $ is used to introduce substitutable parameters. There are two types of parameters, positional and keyword. If parameter is a digit, it is a positional parameter. Positional parameters may be assigned values by set. Keyword parameters (also known as variables) may be assigned values by writing:

name=value [ name=value ]...
Pattern-matching is not performed on value. There cannot be a function and a variable with the same name.

${parameter}$
The value, if any, of the parameter is substituted. The braces are required only when parameter is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. If parameter is * or @, all the positional parameters, starting with $1$, are substituted (separated by spaces). Parameter $0$ is set from argument zero when the shell is invoked.

${parameter=word}$
If parameter is set and is non-null, substitute its value; otherwise substitute word.

${parameter:=word}$
If parameter is not set or is null set it to word; the value of the parameter is substituted. Positional parameters may not be assigned to in this way.

${parameter?:word}$
If parameter is set and is non-null, substitute its value; otherwise, print word and exit from the shell. If word is omitted, the message “parameter null or not set” is printed.

${parameter:+word}$
If parameter is set and is non-null, substitute word; otherwise substitute nothing.

In the above, word is not evaluated unless it is to be used as the substituted string, so that, in the following example, pwd is executed only if d is not set or is null:

```bash
echo ${d:-~ pwd~}
```

If the colon (:) is omitted from the above expressions, the shell only checks whether parameter is set or not.

The following parameters are automatically set by the shell:

- `#` The number of positional parameters in decimal.
- `~` Flags supplied to the shell on invocation or by the set command.
- `?` The decimal value returned by the last synchronously executed command.
- `$` The process number of this shell.
- `!` The process number of the last background command invoked.

The following parameters are used by the shell:

- `HOME` The default argument (home directory) for the cd command.
- `PATH` The search path for commands (see Execution below). The user may not change PATH if executing under rsh.
- `CDPATH` The search path for the cd command.
- `MAIL` If this parameter is set to the name of a mail file and the MAILPATH parameter is not set, the shell informs the user of the arrival of mail in the specified file.
- `MAILCHECK` This parameter specifies how often (in seconds) the shell will check for the arrival of mail in the files specified by the MAILPATH or MAIL parameters. The default value is 600 seconds (10 minutes). If set to 0, the shell
will check before each prompt.

MAILPATH
A colon (:) separated list of file names. If this parameter is set, the shell informs the user of the arrival of mail in any of the specified files. Each file name can be followed by % and a message that will be printed when the modification time changes. The default message is you have mail.

PS1 Primary prompt string, by default "$ ''.  
PS2 Secondary prompt string, by default "> ''.  
IFS Internal field separators, normally space, tab, and new-line. 
SHACCT
If this parameter is set to the name of a file writable by the user, the shell will write an accounting record in the file for each shell procedure executed.

SHELL
When the shell is invoked, it scans the environment (see "Environment" below) for this name. If it is found and 'rsh' is the file name part of its value, the shell becomes a restricted shell.  

The shell gives default values to PATH, PS1, PS2, MAILCHECK and IFS. HOME and MAIL are set by login(1). 

Blank Interpretation
After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in IFS) and split into distinct arguments where such characters are found. Explicit null arguments ("" or «») are retained. Implicit null arguments (those resulting from parameters that have no values) are removed.

Input/Output
A command's input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a simple-command or may precede or follow a command and are not passed on as arguments to the invoked command. Note that parameter and command substitution occurs before word or digit is used.

<word Use file word as standard input (file descriptor 0).

>word Use file word as standard output (file descriptor 1). If the file does not exist it is created; otherwise, it is truncated to zero length.

>>word Use file word as standard output. If the file exists output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.

<[-]word After parameter and command substitution is done on word, the shell input is read up to the first line that literally matches the resulting word, or to an end-of-file. If, however, - is appended to <<:

1) leading tabs are stripped from word before the shell input is read (but after parameter and command substitution is done on word),

2) leading tabs are stripped from the shell input as it is read and before each line is compared with word, and

3) shell input is read up to the first line that literally matches the resulting word, or to an end-of-file.

If any character of word is quoted (see "Quoting," later), no additional processing is done to the shell input. If no characters of word are quoted:
1) parameter and command substitution occurs,
2) (escaped) \new-line is ignored, and
3) \ must be used to quote the characters \
   \$, and \\.

The resulting document becomes the standard input.

<&digit> Use the file associated with file descriptor digit as standard input.
Similarly for the standard output using >&digit.

<&--> The standard input is closed. Similarly for the standard output using
>&--.

If any of the above is preceded by a digit, the file descriptor which will be associated
with the file is that specified by the digit (instead of the default 0 or 1). For example:

... 2>&1

associates file descriptor 2 with the file currently associated with file descriptor 1.

The order in which redirections are specified is significant. The shell evaluates
redirections left-to-right. For example:

... 1>xxx 2>&1

first associates file descriptor 1 with file xxx. It associates file descriptor 2 with the
file associated with file descriptor 1 (i.e., xxx). If the order of redirections were
reversed, file descriptor 2 would be associated with the terminal (assuming file
descriptor 1 had been) and file descriptor 1 would be associated with file xxx.

Using the terminology introduced on the first page, under "Commands," if a com-
mand is composed of several simple commands, redirection will be evaluated for
the entire command before it is evaluated for each simple command. That is, the shell
evaluates redirection for the entire list, then each pipeline within the list, then each
command within each pipeline, then each list within each command.

If a command is followed by & the default standard input for the command is the
empty file /dev/null. Otherwise, the environment for the execution of a command
contains the file descriptors of the invoking shell as modified by input/output
specifications.

Redirection of output is not allowed in the restricted shell.

**File Name Generation**

Before a command is executed, each command word is scanned for the characters *,?,
and [ ]. If one of these characters appears the word is regarded as a pattern. The word
is replaced with alphabetically sorted file names that match the pattern. If no file
name is found that matches the pattern, the word is left unchanged. The character .
at the start of a file name or immediately following a /, as well as the character / itself,
must be matched explicitly.

* Matches any string, including the null string.
? Matches any single character.

[... ] Matches any one of the enclosed characters. A pair of characters
separated by – matches any character lexically between the pair, inclusive.
If the first character following the opening "[" is a "!" any character not
enclosed is matched.

**Quoting**

The following characters have a special meaning to the shell and cause termination of
a word unless quoted:
; & ( ) | ^ < > new-line space tab

A character may be quoted (i.e., made to stand for itself) by preceding it with a
classlash (\) or inserting it between a pair of quote marks (" " or ")"." During
processing, the shell may quote certain characters to prevent them from taking on a spe-
cial meaning. Backslashes used to quote a single character are removed from the
word before the command is executed. The pair \new-line is removed from a word
before command and parameter substitution.

All characters enclosed between a pair of single quote marks (""), except a single
quote, are quoted by the shell. Backslash has no special meaning inside a pair of sin-
gle quotes. A single quote may be quoted inside a pair of double quote marks (for
example,""""").

Inside a pair of double quote marks (""""), parameter and command substitution
occurs and the shell quotes the results to avoid blank interpretation and file name
generation. If $ is within a pair of double quotes, the positional parameters are sub-
stituted and quoted, separated by quoted spaces ("$1 $2 ...""); however, if $ is
within a pair of double quotes, the positional parameters are substituted and quoted,
separated by unquoted spaces ("$1" "$2" ... ). \ quotes the characters \, \, ", and $.
The pair \new-line is removed before parameter and command substitution. If a
backslash precedes characters other than \, \, ", $, and new-line, then the backslash
itself is quoted by the shell.

Prompting

When used interactively, the shell prompts with the value of PS1 before reading a
command. If at any time a new-line is typed and further input is needed to complete
a command, the secondary prompt (i.e., the value of PS2) is issued.

Environment

The environment (see environ(5)) is a list of name-value pairs that is passed to an ex-
cuted program in the same way as a normal argument list. The shell interacts with
the environment in several ways. On invocation, the shell scans the environment and
creates a parameter for each name found, giving it the corresponding value. If the
user modifies the value of any of these parameters or creates new parameters, none
of these affects the environment unless the export command is used to bind the
shell's parameter to the environment (see also set -a). A parameter may be removed
from the environment with the unset command. The environment seen by any exe-
cuted command is thus composed of any unmodified name-value pairs originally
inherited by the shell, minus any pairs removed by unset, plus any modifications or
additions, all of which must be noted in export commands.

The environment for any simple-command may be augmented by prefixing it with one
or more assignments to parameters. Thus:

    TERM=450 cmd
    and
    (export TERM; TERM=450; cmd)

are equivalent (as far as the execution of cmd is concerned).

If the -k flag is set, all keyword arguments are placed in the environment, even if
they occur after the command name. In the following example, the first echo com-
mand prints a=b c; after the set command is issued, the second echo command prints
c:

    echo a=b c
    set -k
    echo a=b c
Signals

The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by &; otherwise signals have the values inherited by the shell from its parent, with the exception of signal 11 (but see also the trap command below).

Execution

Each time a command is executed, the above substitutions are carried out. If the command name matches one of the Special Commands listed below, it is executed in the shell process. If the command name does not match a Special Command, but matches the name of a defined function, the function is executed in the shell process (note how this differs from the execution of shell procedures). The positional parameters $1, $2, ..., are set to the arguments of the function. If the command name matches neither a Special Command nor the name of a defined function, a new process is created and an attempt is made to execute the command via exec(2).

The shell parameter PATH defines the search path for the directory containing the command. Alternative directory names are separated by a colon (:). The default path is /bin/usr/bin (specifying the current directory, /bin, and /usr/bin, in that order). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign, between two colon delimiters anywhere in the path list, or at the end of the path list. If the command name contains a / the search path is not used; such commands will not be executed by the restricted shell. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an a.out file, it is assumed to be a file containing shell commands. A sub-shell is spawned to read it. A parenthesized command is also executed in a sub-shell.

The location in the search path where a command was found is remembered by the shell (to help avoid unnecessary execs later). If the command was found in a relative directory, its location must be re-determined whenever the current directory changes. The shell forgets all remembered locations whenever the PATH variable is changed or the hash -r command is executed (see below).

Special Commands

Input/output redirection is now permitted for these commands. File descriptor 1 is the default output location.

: No effect; the command does nothing. A zero exit code is returned.

, file Read and execute commands from file and return. The search path specified by PATH is used to find the directory containing file.

break [ n ]
Exit from the enclosing for or while loop, if any. If n is specified break n levels.

continue [ n ]
Resume the next iteration of the enclosing for or while loop. If n is specified resume at the n-th enclosing loop.

cd [ arg ]
Change the current directory to arg. The shell parameter HOME is the default arg. The shell parameter CDPATH defines the search path for the directory containing arg. Alternative directory names are separated by a colon (:). The default path is <null> (specifying the current directory). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If arg begins with a / the search path is not used. Otherwise, each directory in the path is searched for arg. The cd command may not be executed by rsh.
echo [ arg ... ]
          Echo arguments. See echo(1) for usage and description.

eval [ arg ... ]
          The arguments are read as input to the shell and the resulting command(s) executed.

eexec [ arg ... ]
          The command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, cause the shell input/output to be modified.

exit [ n ]
          Causes a shell to exit with the exit status specified by n. If n is omitted the exit status is that of the last command executed (an end-of-file will also cause the shell to exit.)

export [ name ... ]
          The given names are marked for automatic export to the environment of subsequently-executed commands. If no arguments are given, variable names that have been marked for export during the current shell's execution are listed. (Variable names exported from a parent shell are listed only if they have been exported again during the current shell's execution.) Function names are not exported.

getopts
          Use in shell scripts to support command syntax standards (see intro(1)); it parses positional parameters and checks for legal options. See getopts(1) for usage and description.

hash [ -r ] [ name ... ]
          For each name, the location in the search path of the command specified by name is determined and remembered by the shell. The -r option causes the shell to forget all remembered locations. If no arguments are given, information about remembered commands is presented. Hits is the number of times a command has been invoked by the shell process. Cost is a measure of the work required to locate a command in the search path. If a command is found in a "relative" directory in the search path, after changing to that directory, the stored location of that command is recalculated. Commands for which this will be done are indicated by an asterisk (*) adjacent to the hits information. Cost will be incremented when the recalculation is done.

newgrp [ arg ... ]
          Equivalent to exec newgrp arg .... See newgrp(1) for usage and description.

pwd
          Print the current working directory. See pwd(1) for usage and description.

read [ name ... ]
          One line is read from the standard input and, using the internal field separator, IFS (normally space or tab), to delimit word boundaries, the first word is assigned to the first name, the second word to the second name, etc., with leftover words assigned to the last name. Lines can be continued using \new-line. Characters other than new-line can be quoted by preceding them with a backslash. These backslashes are removed before words are assigned to names, and no interpretation is done on the character that follows the backslash. The return code is 0 unless an end-of-file is encountered.

readonly [ name ... ]
          The given names are marked readonly and the values of the these names may not be changed by subsequent assignment. If no arguments are given, a list of all readonly names is printed.
return [ n ]
Causes a function to exit with the return value specified by n. If n is omitted, the return status is that of the last command executed.

set [ --aefknntvx [ arg ... ] ]
-a Mark variables which are modified or created for export.
-e Exit immediately if a command exits with a non-zero exit status.
-f Disable file name generation
-h Locate and remember function commands as functions are defined (function commands are normally located when the function is executed).
-k All keyword arguments are placed in the environment for a command, not just those that precede the command name.
-n Read commands but do not execute them.
-t Exit after reading and executing one command.
-u Treat unset variables as an error when substituting.
-v Print shell input lines as they are read.
-x Print commands and their arguments as they are executed.
- Do not change any of the flags; useful in setting $S1 to --.
Using + rather than – causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in $S-.
The remaining arguments are positional parameters and are assigned, in order, to $S1, $S2, … If no arguments are given the values of all names are printed.

shift [ n ]
The positional parameters from $S(n+1) … are renamed $S1 … If n is not given, it is assumed to be 1.

test
Evaluate conditional expressions. See test(1) for usage and description.

times
Print the accumulated user and system times for processes run from the shell.

trap [ arg ] [ n ] …
The command arg is to be read and executed when the shell receives signal(s) n. (Note that arg is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. An attempt to trap on signal 11 (memory fault) produces an error. If arg is absent all trap(s) n are reset to their original values. If arg is the null string this signal is ignored by the shell and by the commands it invokes. If n is 0 the command arg is executed on exit from the shell. The trap command with no arguments prints a list of commands associated with each signal number.

type [ name … ]
For each name, indicate how it would be interpreted if used as a command name.

ulimit [ n ]
Impose a size limit of n blocks on files written by the shell and its child processes (files of any size may be read). If n is omitted, the current limit is printed. You may lower your own ulimit, but only a super-user (see su(1M)) can raise a ulimit.

umask [ nnn ]
The user file-creation mask is set to nnn (see umask(1)). If nnn is omitted, the current value of the mask is printed.
unset [ name ... ]

For each name, remove the corresponding variable or function. The variables PATH, PS1, PS2, MAILCHECK and IFS cannot be unset.

wait [ n ]

Wait for your background process whose process id is n and report its termination status. If n is omitted, all your shell’s currently active background processes are waited for and the return code will be zero.

Invocation

If the shell is invoked through exec(2) and the first character of argument zero is –, commands are initially read from /etc/profile and from $HOME/.profile, if such files exist. Thereafter, commands are read as described below, which is also the case when the shell is invoked as /bin/sh. The flags below are interpreted by the shell on invocation only; Note that unless the −c or −s flag is specified, the first argument is assumed to be the name of a file containing commands, and the remaining arguments are passed as positional parameters to that command file:

−c string If the −c flag is present commands are read from string.

−s If the −s flag is present or if no arguments remain commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output (except for Special Commands) is written to file descriptor 2.

−i If the −i flag is present or if the shell input and output are attached to a terminal, this shell is interactive. In this case TERMINATE is ignored (so that kill 0 does not kill an interactive shell) and INTERRUPT is caught and ignored (so that wait is interruptible). In all cases, QUIT is ignored by the shell.

−r If the −r flag is present the shell is a restricted shell.

The remaining flags and arguments are described under the set command above.

rsh Only

rsh is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of rsh are identical to those of sh, except that the following are disallowed:
changing directory (see cd(1)), setting the value of $PATH, specifying path or command names containing /, redirecting output (> and >>).

The restrictions above are enforced after .profile is interpreted.

A restricted shell can be invoked in one of the following ways: (1) rsh is the file name part of the last entry in the /etc/passwd file (see passwd(4)); (2) the environment variable SHELL exists and rsh is the file name part of its value; (3) the shell is invoked and rsh is the file name part of argument 0; (4) the shell is invoke with the −r option.

When a command to be executed is found to be a shell procedure, rsh invokes sh to execute it. Thus, it is possible to provide to the end-user shell procedures that have access to the full power of the standard shell, while imposing a limited menu of commands; this scheme assumes that the end-user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the .profile (see profile(4)) has complete control over user actions by performing guaranteed setup actions and leaving the user in an appropriate directory (probably not the login directory).
The system administrator often sets up a directory of commands (i.e., /usr/bin) that can be safely invoked by a restricted shell. Some systems also provide a restricted editor, red.

**EXIT STATUS**

Errors detected by the shell, such as syntax errors, cause the shell to return a non-zero exit status. If the shell is being used non-interactively execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also the exit command above).

**FILES**

/etc/profile  
$HOME/.profile  
/tmp/sh*  
/dev/null

**SEE ALSO**

cd(1), dup(2), echo(1), env(1), exec(2), fork(2), getopt(1), intro(1), login(1), newgrp(1), pipe(2), profile(4), pwd(1), signal(2), test(1), ulimit(2), umask(1), wait(1).

**CAVEATS**

Words used for filenames in input/output redirection are not interpreted for filename generation (see "File Name Generation," above). For example, cat file1 >a* will create a file named a*.

Because commands in pipelines are run as separate processes, variables set in a pipeline have no effect on the parent shell.

If you get the error message cannot fork, too many processes, try using the wait(1) command to clean up your background processes. If this doesn’t help, the system process table is probably full or you have too many active foreground processes. (There is a limit to the number of process ids associated with your login, and to the number the system can keep track of.)

**BUGS**

If a command is executed, and a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to exec the original command. Use the hash command to correct this situation.

If you move the current directory or one above it, pwd may not give the correct response. Use the cd command with a full path name to correct this situation.

Not all the processes of a 3- or more-stage pipeline are children of the shell, and thus cannot be waited for.

For wait n, if n is not an active process id, all your shell’s currently active background processes are waited for and the return code will be zero.
NAME

shutdown – shut down system, change system state

SYNOPSIS

/etc/shutdown [ -y ] [ -g grace_period ] [ -i init_state ]

DESCRIPTION

This command is executed by the super-user to change the state of the machine. By default, it brings the system to a state where only the console has access to the UNIX system. This state is traditionally called "single-user".

The command sends a warning message and a final message before it starts actual shutdown activities. By default, the command asks for confirmation before it starts shutting down daemons and killing processes. The options are used as follows:

- **-y** pre-answers the confirmation question so the command can be run without user intervention. A default of 60 seconds is allowed between the warning message and the final message. Another 60 seconds is allowed between the final message and the confirmation.

- **-g grace_period**
  allows the super-user to change the number of seconds from the 60-second default.

- **-i init_state**
  specifies the state that init(1M) is to be put in following the warnings, if any. By default, system state "s" is used (the same as states "1" and "S").

Other recommended system state definitions are:

state 0
Shut the machine down so it is safe to remove the power. Have the machine remove power if it can. The /etc/rc0 procedure is called to do this work.

state 1, s, S
Bring the machine to the state traditionally called single-user. The /etc/rc0 procedure is called to do this work. (Though s and 1 are both used to go to single user state, s only kills processes spawned by init and does not unmount file systems. State 1 unmounts everything except root and kills all user processes, except those that relate to the console.)

state 5
Stop the UNIX system and go to the firmware monitor.

state 6
Stop the UNIX system and reboot to the state defined by the initdefault entry in /etc/inittab.

FILES

/etc/shutdown.d

SEE ALSO

init(1M), inittab(4), rc0(8), rc2(8)
NAME
size – print section sizes in bytes of common object files

SYNOPSIS
size [-doxV] files

DESCRIPTION
The size command produces section size information in bytes for each loaded section in the common object files. The size of the text, data, and bss (uninitialized data) sections is printed, as well as the sum of the sizes of these sections. If an archive file is input to the size command the information for all archive members is displayed.

Numbers will be printed in decimal unless either the -o or the -x option is used, in which case they will be printed in octal or in hexadecimal, respectively.

The -V flag will supply the version information on the size command.

SEE ALSO
as(1), cc(1), ld(1), a.out(4), ar(4).

CAVEAT
Since the size of bss sections is not known until link-edit time, the size command will not give the true total size of pre-linked objects.
NAME

sleep – suspend execution for an interval

SYNOPSIS

sleep time

DESCRIPTION

`sleep` suspends execution for `time` seconds. It is used to execute a command after a certain amount of time, as in:

(sleep 105; command)&

or to execute a command every so often, as in:

while true
do
  command
  sleep 37
  done

SEE ALSO

alarm(2), sleep(3C).
NAME

sort – sort and/or merge files

SYNOPSIS


DESCRIPTION

sort sorts lines of all the named files together and writes the result on the standard output. The standard input is read if – is used as a file name or no input files are named.

Comparisons are based on one or more sort keys extracted from each line of input. By default, there is one sort key, the entire input line, and ordering is lexicographic by bytes in machine collating sequence.

The following options alter the default behavior:

-c Check that the input file is sorted according to the ordering rules; give no output unless the file is out of sort.

-m Merge only, the input files are already sorted.

-u Unique: suppress all but one in each set of lines having equal keys.

-ooutput

The argument given is the name of an output file to use instead of the standard output. This file may be the same as one of the inputs. There may be optional blanks between –o and output.

-ykmem

The amount of main memory used by the sort has a large impact on its performance. Sorting a small file in a large amount of memory is a waste. If this option is omitted, sort begins using a system default memory size, and continues to use more space as needed. If this option is presented with a value, kmem, sort will start using that number of kilobytes of memory, unless the administrative minimum or maximum is violated, in which case the corresponding extremum will be used. Thus, –y0 is guaranteed to start with minimum memory. By convention, –y (with no argument) starts with maximum memory.

-zrecesz

The size of the longest line read is recorded in the sort phase so buffers can be allocated during the merge phase. If the sort phase is omitted via the –c or –m options, a popular system default size will be used. Lines longer than the buffer size will cause sort to terminate abnormally. Supplying the actual number of bytes in the longest line to be merged (or some larger value) will prevent abnormal termination.

The following options override the default ordering rules.

-d ‘Dictionary’ order: only letters, digits and blanks (spaces and tabs) are significant in comparisons.

-f Fold lower case letters into upper case.

-i Ignore characters outside the ASCII range 040-0176 in non-numeric comparisons.

-M Compare as months. The first three non-blank characters of the field are folded to upper case and compared so that ”JAN” < ”FEB” < … < ”DEC”. Invalid fields compare low to ”JAN”. The –M option implies the –b option (see below).

-n An initial numeric string, consisting of optional blanks, optional minus sign, and zero or more digits with optional decimal point, is sorted by arithmetic value. The –n option implies the –b option (see below). Note that the –b
option is only effective when restricted sort key specifications are in effect.

-\texttt{r} Reverse the sense of comparisons.

-\texttt{dir}

\texttt{dir} is the name of a directory in which temporary files should be created.

When ordering options appear before restricted sort key specifications, the requested ordering rules are applied globally to all sort keys. When attached to a specific sort key (described below), the specified ordering options override all global ordering options for that key.

The notation +\texttt{pos1} –\texttt{pos2} restricts a sort key to one beginning at \texttt{pos1} and ending just before \texttt{pos2}. The characters at position \texttt{pos1} and just before \texttt{pos2} are included in the sort key (provided that \texttt{pos2} does not precede \texttt{pos1}). A missing –\texttt{pos2} means the end of the line.

Specifying \texttt{pos1} and \texttt{pos2} involves the notion of a field, a minimal sequence of characters followed by a field separator or a new-line. By default, the first blank (space or tab) of a sequence of blanks acts as the field separator. All blanks in a sequence of blanks are considered to be part of the next field; for example, all blanks at the beginning of a line are considered to be part of the first field. The treatment of field separators can be altered using the options:

-\texttt{b} Ignore leading blanks when determining the starting and ending positions of a restricted sort key. If the –\texttt{b} option is specified before the first +\texttt{pos1} argument, it will be applied to all +\texttt{pos1} arguments. Otherwise, the \texttt{b} flag may be attached independently to each +\texttt{pos1} or –\texttt{pos2} argument (see below).

-\texttt{x} Use \texttt{x} as the field separator character; \texttt{x} is not considered to be part of a field (although it may be included in a sort key). Each occurrence of \texttt{x} is significant (for example, \texttt{xx} delimits an empty field).

\texttt{Pos1} and \texttt{pos2} each have the form \texttt{m.n} optionally followed by one or more of the flags \texttt{bdfmn}. A starting position specified by +\texttt{m.n} is interpreted to mean the \texttt{n}+1st character in the \texttt{m}+1st field. A missing \texttt{n} means \texttt{0}, indicating the first character of the \texttt{m}+1st field. If the \texttt{b} flag is in effect \texttt{n} is counted from the first non-blank in the \texttt{m}+1st field; +\texttt{m.0b} refers to the first non-blank character in the \texttt{m}+1st field.

A last position specified by –\texttt{m.n} is interpreted to mean the \texttt{n}th character (including separators) after the last character of the \texttt{m}th field. A missing \texttt{n} means \texttt{0}, indicating the last character of the \texttt{m}th field. If the \texttt{b} flag is in effect \texttt{n} is counted from the last leading blank in the \texttt{m}+1st field; –\texttt{m.1b} refers to the first non-blank in the \texttt{m}+1st field.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant.

\textbf{EXAMPLES}

Sort the contents of \texttt{infile} with the second field as the sort key:

\begin{verbatim}
sort +1 –2 infile
\end{verbatim}

Sort, in reverse order, the contents of \texttt{infile1} and \texttt{infile2}, placing the output in \texttt{outfile} and using the first character of the second field as the sort key:

\begin{verbatim}
sort –r –o outfile +1.0 –1.2 infile1 infile2
\end{verbatim}

Sort, in reverse order, the contents of \texttt{infile1} and \texttt{infile2} using the first non-blank character of the second field as the sort key:
sort -r +1.0b -1.1b infile1 infile2

Print the password file (passwd(4)) sorted by the numeric user ID (the third colon-separated field):

    sort -t: +2n -3 /etc/passwd

Print the lines of the already sorted file infile, suppressing all but the first occurrence of lines having the same third field (the options -um with just one input file make the choice of a unique representative from a set of equal lines predictable):

    sort -um +2 -3 infile

FILES

/usr/tmp/stm???

SEE ALSO

comm(1), join(1), uniq(1).

WARNINGS

Comments and exits with non-zero status for various trouble conditions (for example, when input lines are too long), and for disorder discovered under the -c option. When the last line of an input file is missing a new-line character, sort appends one, prints a warning message, and continues.

sort does not guarantee preservation of relative line ordering on equal keys.
NAME

spell, hashmake, spellin, hashcheck – find spelling errors

SYNOPSIS

spell [-v] [-b] [-x] [-i] [ +local_file ] [ files ]
/usr/lib/spell/hashmake
/usr/lib/spell/spellin n
/usr/lib/spell/hashcheck spelling_list

DESCRIPTION

spell collects words from the named files and looks them up in a spelling list. Words
that neither occur among nor are derivable (by applying certain inflections, prefixes,
and/or suffixes) from words in the spelling list are printed on the standard output.
If no files are named, words are collected from the standard input.

spell ignores most troff(1), tbl(1), and eqn(1) constructions.

Under the -v option, all words not literally in the spelling list are printed, and plausible
derivations from the words in the spelling list are indicated.

Under the -b option, British spelling is checked. Besides preferring centre, colour,
programme, speciality, travelled, etc., this option insists upon -ise in words like standardise,
Fowler and the OED to the contrary notwithstanding.

Under the -x option, every plausible stem is printed with = for each word.

By default, spell (like deroff(1)) follows chains of included files (.so and .nx troff(1)
requests), unless the names of such included files begin with /usr/lib. Under the -l
option, spell will follow the chains of all included files.

Under the +local_file option, words found in local_file are removed from spell's output.
Local_file is the name of a user-provided file that contains a sorted list of words,
one per line. With this option, the user can specify a set of words that are correct
spellings (in addition to spell's own spelling list) for each job.

The spelling list is based on many sources, and while more haphazard than an ordinary
dictionary, is also more effective with respect to proper names and popular
technical words. Coverage of the specialized vocabularies of biology, medicine, and
chemistry is light.

Pertinent auxiliary files may be specified by name arguments, indicated below with
their default settings (see FILES). Copies of all output are accumulated in the history
file. The stop list filters out misspellings (e.g., thier=th-y+ier) that would otherwise
pass.

Three routines help maintain and check the hash lists used by spell:

hashmake Reads a list of words from the standard input and writes the correspond-
ing nine-digit hash code on the standard output.

spellin Reads n hash codes from the standard input and writes a compressed
spelling list on the standard output.

hashcheck Reads a compressed spelling_list and recreates the nine-digit hash codes
for all the words in it; it writes these codes on the standard output.

FILES

D_SPELL=/usr/lib/spell/hlist[ab] hashed spelling lists, American & British
S_SPELL=/usr/lib/spell/hstop hashed stop list
H_SPELL=/usr/lib/spell/spellhist history file
/usr/lib/spell/spellprogs  program

SEE ALSO
deroff(1), eqn(1), sed(1), sort(1), tbl(1), tee(1), troff(1).

BUGS
The spelling list's coverage is uneven; new installations will probably wish to monitor the output for several months to gather local additions; typically, these are kept in a separate local file that is added to the hashed spelling_list via spellin.
NAME
split – split a file into pieces

SYNOPSIS
split [ -n ] [ file [ name ] ]

DESCRIPTION
split reads file and writes it in n-line pieces (default 1000 lines) onto a set of output files. The name of the first output file is name with aa appended, and so on lexicographically, up to zz (a maximum of 676 files). Name cannot be longer than 12 characters. If no output name is given, x is default.

If no input file is given, or if — is given in its stead, then the standard input file is used.

SEE ALSO
bfs(1), csplit(1).
NAME
strings – find the printable strings in an object, or other binary, file

SYNOPSIS
strings [-] [-o] [-number] file ...

DESCRIPTION
strings looks for ascii strings in a binary file. A string is any sequence of 4 or more printing characters ending with a newline or a null. Unless the – flag is given, strings only looks in the initialized data space of object files. If the –o flag is given, then each string is preceded by its offset in the file (in octal). If the –number flag is given then number is used as the minimum string length rather than 4.

strings is useful for identifying random object files and many other things.

SEE ALSO
od(1)

BUGS
The algorithm for identifying strings is extremely primitive.
NAME
strip – strip symbol and line number information from a common object file

SYNOPSIS
strip [-l] [-x] [-b] [-r] [-V] filename ...

DESCRIPTION
The strip command renders debugging information in executables undecipherable by
the debugger and by disassembler tools. This helps third-party software to be better
protected from unfriendly users. Note that the Stardent 1500/3000 strip command
does not completely remove this information, as do strip commands on most other
systems. As a result, using strip does not result in a very large reduction of disk
usage and is of little value to users who do not want protection against disassembly.

Stardent strongly discourages users from using the strip command. One of the
important tools provided by Stardent is a postloading capability. Among other things,
the profiler, the –opct loader option, and the adapt(1) utility (which transforms Star-
dent 1500 executables so that they will run on Stardent 3000) are all postloader-based
programs. Postloading require much of the same debugging information that is
removed by the standard strip command, so that a program that has been stripped in
the normal sense cannot be postloaded. The Stardent 1500/3000 strip command
retains enough information to allow current postloaders to function on the resulting
 executable. However, Stardent cannot guarantee that future postloaders will be able
to function on executables that have been stripped by the Stardent 1500/3000 strip
command; in particular, it is very likely that postloaders that convert Stardent 3000
 a.out files to be executable on future Stardent machines will not work on stripped exe-
cutables. Consequently, Stardent strongly discourages users from using the strip
command.

strip accepts the options listed above to maintain compatability with existing sys-
tems. However, excepting –V, which prints version information, strip performs the
same operations regardless of any given options. strip operates only on executables;
when called on archives or object files, it silently does nothing.

FILES
  TMPDIR/str    temporary files

TMPDIR is usually /usr/tmp but can be redefined by setting the TMPDIR environment variable (see
  tempnam() in tmppnam(3)).

SEE ALSO
  a.out(4), ar(1), ar(4) as(1), ld(1), tmppnam(3)

DIAGNOSTICS
  strip: name: cannot open
  –if name cannot be read
  strip: name: bad magic
  –if name is not an appropriate file
NAME

stty – set the options for a terminal

SYNOPSIS

stty [ -a ] [ -g ] [ options ]

DESCRIPTION

stty sets certain terminal I/O options for the device that is the current standard input; without arguments, it reports the settings of certain options.

In this report, if a character is preceded by a caret (^), then the value of that option is the corresponding CTRL character (e.g., "^h" is CTRL-h; in this case, recall that CTRL-h is the same as the "back-space" key.) The sequence "^" means that an option has a null value. For example, normally stty -a will report that the value of switch is "^"; however, if shl (1) or layers (1) has been invoked, stty -a will have the value "^z".

-a reports all of the option settings;

-g reports current settings in a form that can be used as an argument to another stty command.

Options in the last group are implemented using options in the previous groups. Note that many combinations of options make no sense, but no sanity checking is performed. The options are selected from the following:

Control Modes

parenb (-parenb) enable (disable) parity generation and detection.
parodd (-parodd) select odd (even) parity.
cs5 cs6 cs7 cs8 select character size (see termio(7)).
0 hang up phone line immediately.
110 300 600 1200 1800 2400 4800 9600 19200 38400

hupcl (-hupcl) hang up (do not hang up) Dataphone connection on last close.
hup (-hup) same as hupcl (-hupcl).
cstopb (-cstopb) use two (one) stop bits per character.
cread (-cread) enable (disable) the receiver.
clokal (-clokal) n assume a line without (with) modem control.
loblk (-loblk) block (do not block) output from a non-current layer.

Input Modes

ignbrk (-ignbrk) ignore (do not ignore) break on input.
brkint (-brkint) signal (do not signal) INTR on break.
ignpar (-ignpar) ignore (do not ignore) parity errors.
parmkrk (-parmkrk) mark (do not mark) parity errors (see termio(7)).
inpck (-inpck) enable (disable) input parity checking.
istrip (-istrip) strip (do not strip) input characters to seven bits.
inlcr (-inlcr) map (do not map) NL to CR on input.
igncr (-igncr) ignore (do not ignore) CR on input.
icrnl (-icrnl) map (do not map) CR to NL on input.
iuclcr (-iuclcr) map (do not map) upper-case alphabetics to lower case on input.
ixon (-ixon) enable (disable) START/STOP output control. Output is stopped by sending an ASCII DC3 and started by sending an ASCII DC1.
ixany (--ixany) allow any character (only DC1) to restart output.
ixo ff (--ixo ff) request that the system send (not send) START/STOP characters when the input queue is nearly empty/full.

Output Modes

OPost (--opost) post-process output (do not post-process output; ignore all other output modes).
olcuc (--olcuc) map (do not map) lower-case alphabetics to upper case on output.
onlcr (--onlcr) map (do not map) NL to CR-NL on output.
ocrnl (--ocrnl) map (do not map) CR to NL on output.
onocr (--onocr) do not (do) output CRs at column zero.
onlret (--onlret) on the terminal NL performs (does not perform) the CR function.
ofill (--ofill) use fill characters (use timing) for delays.
ofdel (--ofdel) fill characters are DELs (NULs).
cr0 cr1 cr2 cr3 select style of delay for carriage returns (see termio(7)).
nl0 nl1 select style of delay for line-feeds (see termio(7)).
tab0 tab1 tab2 tab3 select style of delay for horizontal tabs (see termio(7)).
bs0 bs1 select style of delay for backspaces (see termio(7)).
ff0 ff1 select style of delay for form-feeds (see termio(7)).
vt0 vt1 select style of delay for vertical tabs (see termio(7)).

Local Modes

isig (--isig) enable (disable) the checking of characters against the special control characters INTR, QUIT, and SWTCH.
icanon (--icanon) enable (disable) canonical input (ERASE and KILL processing).
xcase (--xcase) canonical (unprocessed) upper/lower-case presentation.
echo (--echo) echo back (do not echo back) every character typed.
echoe (--echoe) echo (do not echo) ERASE character as a backspace-spacebackspace string. Note: this mode will erase the ERASEd character on many CRT terminals; however, it does not keep track of column position and, as a result, may be confusing on escaped characters, tabs, and backspaces.
echok (--echok) echo (do not echo) NL after KILL character.
lfc (--lfc) the same as echok (--echok); obsolete.
echonl (--echonl) echo (do not echo) NL.
nofish (--nosh) disable (enable) flush after INTR, QUIT, or SWTCH.
stwrap (--stwrap) disable (enable) truncation of lines longer than 79 characters on a synchronous line. (Does not apply to the 3B2.)
stflush (--stflush) enable (disable) flush on a synchronous line after every write(2). (Does not apply to the 3B2.)
stappl (--stappl) use application mode (use line mode) on a synchronous line. (Does not apply to the 3B2.)

Control Assignments

ccontrol-character c set control-character to c, where control-character is erase, kill, intr, quiet, switch, eof, ctab, or time (ctab is used with --stappl; min and time are used with --icanon; see termio(7)). If c is preceded by an (escaped from the shell) caret (^), then the value used is the corresponding CTRL character (e.g., “^d” is a CTRL-d); “^?” is interpreted as DEL and “^~” is interpreted as undefined.
line \( i \) set line discipline to \( i \) (0 < \( i \) < 127).

**Combination Modes**

- `evenp or parity`: enable `parenb` and `cs7`.
- `oddp`: enable `parenb`, `cs7`, and `parodd`.
- `−parity, −evenp, or −oddp`: disable `parenb`, and set `cs8`.
- `raw (−raw or cooked)`: enable (disable) raw input and output (no ERASE, KILL, INTR, QUIT, SWTCH, EOT, or output post processing).
- `nl (−nl)`: unset (set) `icrnl`, `onlcr`. In addition `−nl` unsets `inlcr`, `igncr`, `ocrnl`, and `onlret`.
- `lcase (−lcase)`: set (unset) `xcase`, `lucap`, and `olcuc`.
- `LCASE (−LCASE)`: same as `lcase (−lcase)`.
- `tabs (−tabs or tab3)`: preserve (expand to spaces) tabs when printing.
- `ek`: reset ERASE and KILL characters back to normal # and @.
- `sane`: resets all modes to some reasonable values.
- `term`: set all modes suitable for the terminal type `term`, where `term` is one of `tty33`, `tty37`, `vt05`, `tn300`, `ti700`, or `tek`.

**SEE ALSO**

`ioctl(2)`, `tabs(1)`, `termio(7)`.
NAME

stty – set terminal options

SYNOPSIS

stty [ option ... ]

DESCRIPTION

Stty sets certain I/O options on the current output terminal, placing its output on the
diagnostic output. With no argument, it reports the speed of the terminal and the
settings of the options which are different from their defaults. Use of one of the fol-
lowing options modifies the output as described:

all All normally used option settings are reported.
everything Everything stty knows about is printed.
speed The terminal speed alone is printed on the standard output.
size The terminal (window) sizes are printed on the standard output, first
rows and then columns.

The option strings are selected from the following set:
even allow even parity input
-even disallow even parity input
odd allow odd parity input
-odd disallow odd parity input
raw raw mode input (no input processing (erase, kill, interrupt, ...); parity bit
passed back)
-raw negate raw mode
cooked same as ‘-raw’
cbreak make each character available to read(2) as received; no erase and kill pro-
cessing, but all other processing (interrupt, suspend, ...) is performed
-cbreak make characters available to read only when newline is received
-nl allow carriage return for new-line, and output CR-LF for carriage return
or new-line
nl accept only new-line to end lines
echo echo back every character typed
-echo do not echo characters
lcase map upper case to lower case
-lcase do not map case
tandem enable flow control, so that the system sends out the stop character when
its internal queue is in danger of overflowing on input, and sends the
start character when it is ready to accept further input
-tandem disable flow control
-tabs replace tabs by spaces when printing
tabs preserve tabs
ek set erase and kill characters to # and @

For the following commands which take a character argument c, you may also
specify c as the “*" or "undef", to set the value to be undefined. A value of "^x", a 2
character sequence, is also interpreted as a control character, with "^?" representing
delete.
erase c set erase character to c (default ‘#’, but often reset to ^H.)
kill c set kill character to c (default ‘@’, but often reset to ^U.)
intr c set interrupt character to c (default DEL or ^? (delete), but often reset to
^C.)
4.3 BSD Utility

**quit c** set quit character to c (default control \.)

**start c** set start character to c (default control Q.)

**stop c** set stop character to c (default control S.)

**eof c** set end of file character to c (default control D.)

**brk c** set break character to c (default undefined.) This character is an additional character causing wakeup.

**cr0 cr1 cr2 cr3**

select style of delay for carriage return (see ioctl(2))

**nl0 nl1 nl2 nl3**

select style of delay for linefeed

**tab0 tab1 tab2 tab3**

select style of delay for tab

**ff0 ff1**

select style of delay for form feed

**bs0 bs1**

select style of delay for backspace

**tty33**

set all modes suitable for the Teletype Corporation Model 33 terminal.

**tty37**

set all modes suitable for the Teletype Corporation Model 37 terminal.

**vt05**

set all modes suitable for Digital Equipment Corp. VT05 terminal

**dec**

set all modes suitable for Digital Equipment Corp. operating systems users; (erase, kill, and interrupt characters to ^?, ^U, and ^C, decstq and "newcrt")

**tn300**

set all modes suitable for a General Electric TermiNet 300

**ti700**

set all modes suitable for Texas Instruments 700 series terminal

**tek**

set all modes suitable for Tektronix 4014 terminal

**0**

hang up phone line immediately

<table>
<thead>
<tr>
<th>50</th>
<th>75</th>
<th>110</th>
<th>134</th>
<th>150</th>
<th>200</th>
<th>300</th>
<th>600</th>
<th>1200</th>
<th>1800</th>
<th>2400</th>
<th>4800</th>
<th>9600</th>
<th>exta</th>
<th>extb</th>
</tr>
</thead>
</table>

Set terminal baud rate to the number given, if possible. (These are the speeds supported by the DH-11 interface).

**rows n** The terminal size is recorded as having n rows.

**columns n** The terminal size is recorded as having n columns.

**cols n** is an alias for columns.

A teletype driver which supports the job control processing of csh(1) and more functionality than the basic driver is fully described in tty(4). The following options apply only to it.

**new**

Use new driver (switching flushes typeahead).

**crt**

Set options for a CRT (crts, ctclecho and, if >= 1200 baud, crterase and ctkill.)

**crts**

Echo backspaces on erase characters.

**prterase**

For printing terminal echo erased characters backwards within "\" and "/".

**crterase**

Wipe out erased characters with "backspace-space-backspace."

**-crterase**

Leave erased characters visible; just backspace.

**crtkill**

Wipe out input on like kill ala crterase.

**-crtkill**

Just echo line kill character and a newline on line kill.

**ctclecho**

Echo control characters as "\x" (and delete as "^?".) Print two backspaces following the EOT character (control D).

**-ctclecho**

Control characters echo as themselves; in cooked mode EOT (control-D) is not echoed.

**decstq**

After output is suspended (normally by ^S), only a start character (normally ^Q) will restart it. This is compatible with DEC's vendor supplied systems.
-dectlq  After output is suspended, any character typed will restart it; the start character will restart output without providing any input. (This is the default.)
tostop  Background jobs stop if they attempt terminal output.
-tostop  Output from background jobs to the terminal is allowed.
tilde  Convert "~" to "'" on output (for Hazeltine terminals).
-tilde  Leave poor "~" alone.
flusho  Output is being discarded usually because user hit control O (internal state bit).
-flusho  Output is not being discarded.
pendin  Input is pending after a switch from cbreak to cooked and will be re-input when a read becomes pending or more input arrives (internal state bit).
-pendin  Input is not pending.
pass8  Passes all 8 bits through on input, in any mode.
-pass8  Strips the 0200 bit on input except in raw mode.
mdmbuf  Start/stop output on carrier transitions (not implemented).
-mdmbuf  Return error if write attempted after carrier drops.
soft  Send output characters without any processing.
-litout  Do normal output processing, inserting delays, etc.
nohang  Don't send hangup signal if carrier drops.
-nohang  Send hangup signal to control process group when carrier drops.
etack  Diablo style etx/ack handshaking (not implemented).
The following special characters are applicable only to the new teletype driver and are not normally changed.
susp c  set suspend process character to c (default control Z).
dsusp c  set delayed suspend process character to c (default control Y).
rrpnt c  set reprint line character to c (default control R).
flush c  set flush output character to c (default control O).
werase c  set word erase character to c (default control W).
lnext c  set literal next character to c (default control V).

SEE ALSO
ioctl(2), tabs(1), tset(1), tty(4)
NAME

su – become super-user or another user

SYNOPSIS

su [-] [name [arg ...]]

DESCRIPTION

su allows one to become another user without logging off. The default user name is root (i.e., super-user).

To use su, the appropriate password must be supplied (unless one is already root). If the password is correct, su will execute a new shell with the real and effective user ID set to that of the specified user. The new shell will be the optional program named in the shell field of the specified user’s password file entry (see passwd(4)), or /bin/sh if none is specified (see sh(1)). To restore normal user ID privileges, type an EOF (ctrl-d) to the new shell.

Any additional arguments given on the command line are passed to the program invoked as the shell. When using programs like sh(1), an arg of the form -c string executes string via the shell and an arg of -r will give the user a restricted shell.

The following statements are true only if the optional program named in the shell field of the specified user’s password file entry is like sh(1). If the first argument to su is a -, the environment will be changed to what would be expected if the user actually logged in as the specified user. This is done by invoking the program used as the shell with an arg0 value whose first character is -, causing first the system’s profile (/etc/profile) and then the specified user’s profile (.profile in the new HOME directory) to be executed. Otherwise, the environment is passed along with the possible exception of $PATH, which is set to /bin:/etc:/usr/bin for root. Note that if the optional program used as the shell is /bin/sh, the user’s .profile can check arg0 for -sh or -su to determine if it was invoked by login(1) or su(1), respectively. If the user’s program is other than /bin/sh, then .profile is invoked with an arg0 of -program by both login(1) and su(1).

All attempts to become another user using su are logged in the log file /usr/adm/sulog.

EXAMPLES

To become user bin while retaining your previously exported environment, execute:

```
su bin
```

To become user bin but change the environment to what would be expected if bin had originally logged in, execute:

```
su - bin
```

To execute command with the temporary environment and permissions of user bin, type:

```
su - bin -c "command args"
```

FILES

/etc/passwd  system’s password file
/etc/profile  system’s profile
$HOME/.profile user’s profile
/usr/adm/sulog  log file

SEE ALSO

env(1), login(1), passwd(4), profile(4), sh(1).
NAME
sum – print checksum and block count of a file

SYNOPSIS
sum [ -r ] file

DESCRIPTION
sum calculates and prints a 16-bit checksum for the named file, and also prints the
number of blocks in the file. It is typically used to look for bad spots, or to validate a
file communicated over some transmission line. The option -r causes an alternate
algorithm to be used in computing the checksum.

SEE ALSO
wc(1).

DIAGNOSTICS
"Read error" is indistinguishable from end of file on most devices; check the block
count.
NAME

swap – swap administrative interface

SYNOPSIS

/etc/swap -a swapdev swaplow swaplen
/etc/swap -d swapdev swaplow
/etc/swap -l

DESCRIPTION

swap provides a method of adding, deleting, and monitoring the system swap areas used by the memory manager. The following options are recognized:

-a Add the specified swap area. swapdev is the name of the block special device, e.g., /dev/dsk/c0d1s4. swaplow is the offset in 512-byte blocks into the device where the swap area should begin. swaplen is the length of the swap area in 512-byte blocks. This option can only be used by the super-user. Swap areas are normally added by the system start up routine /etc/rc when going into multi-user mode.

-d Delete the specified swap area. swapdev is the name of block special device, e.g., /dev/dsk/c0d1s4. swaplow is the offset in 512-byte blocks into the device where the swap area should begin. Using this option marks the swap area as "INDEL" (in process of being deleted). The system will not allocate any new blocks from the area, and will try to free swap blocks from it. The area will remain in use until all blocks from it are freed. This option can only be used by the super-user.

-l List the status of all the swap areas. The output has four columns:

DEV The swapdev special file for the swap area if one can be found in the /dev/dsk or /dev directories, and its major/minor device number in decimal.

LOW The swaplow value for the area in 512-byte blocks.

LEN The swaplen value for the area in 512-byte blocks.

FREE The number of free 512-byte blocks in the area. If the swap area is being deleted, this column will be marked INDEL.

WARNINGS

No check is done to see if a swap area being added overlaps with an existing swap area or file system.
NAME

sync – update the super block

SYNOPSIS

sync

DESCRIPTION

sync executes the sync system primitive. If the system is to be stopped, sync must be
called to insure file system integrity. It will flush all previously unwritten system
buffers out to disk, thus assuring that all file modifications up to that point will be
saved. See sync(2) for details.

SEE ALSO

sync(2)
NAME
tabs – set tabs on a terminal

SYNOPSIS
tabs [tabspec] [-Ttype] [+mn]

DESCRIPTION
tabs sets the tab stops on the user’s terminal according to the tab specification tabspec, after clearing any previous settings. The user’s terminal must have remotely-settable hardware tabs.

tabspec Four types of tab specification are accepted for tabspec. They are described below: canned (-code), repetitive (-n), arbitrary (n1,n2,...), and file (-file). If no tabspec is given, the default value is -8, i.e., UNIX system “standard” tabs. The lowest column number is 1. Note that for tabs, column 1 always refers to the leftmost column on a terminal, even one whose column markers begin at 0, e.g., the DASI 300, DASI 300s, and DASI 450.

-code Use one of the codes listed below to select a canned set of tabs. The legal codes and their meanings are as follows:
- a 1,10,16,36,72
  Assembler, IBM S/370, first format
- a2 1,10,16,40,72
  Assembler, IBM S/370, second format
- c 1,8,12,16,20,55
  COBOL, normal format
- c2 1,6,10,14,49
  COBOL compact format (columns 1-6 omitted). Using this code, the first typed character corresponds to card column 7, one space gets you to column 8, and a tab reaches column 12. Files using this tab setup should include a format specification as follows (see fspec(4)):
  <1-tc2 m6 s66 d:>
- c3 1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67
  COBOL compact format (columns 1-6 omitted), with more tabs than -c2. This is the recommended format for COBOL. The appropriate format specification is (see fspec(4)):
  <1-tc3 m6 s66 d:>
- f 1,7,11,15,19,23
  FORTRAN
- p 1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61
  PL/I
- s 1,10,55
  SNOBOL
- u 1,12,20,44
  UNIVAC 1100 Assembler
A repetitive specification requests tabs at columns 1+n, 1+2*n, etc. Of particular importance is the value 8: this represents the UNIX system “standard” tab setting, and is the most likely tab setting to be found at a terminal. Another special case is the value 0, implying no tabs at all.

The arbitrary format permits the user to type any chosen set of numbers, separated by commas, in ascending order. Up to 40 numbers are allowed. If any number (except the first one) is preceded by a plus sign, it is taken as an increment to be added to the previous value. Thus, the formats 1,10,20,30, and 1,10,+10,+10 are considered identical.

If the name of a file is given, tabs reads the first line of the file, searching for a format specification (see fspec(4)). If it finds one there, it sets the tab stops according to it, otherwise it sets them as –8. This type of specification may be used to make sure that a tabbed file is printed with correct tab settings, and would be used with the pr(1) command:

tabs -- file; pr file

Any of the following also may be used; if a given flag occurs more than once, the last value given takes effect:

- T

tabs usually needs to know the type of terminal in order to set tabs and always needs to know the type to set margins. type is a name listed in term(5). If no –T flag is supplied, tabs uses the value of the environment variable TERM. If TERM is not defined in the environment (see environ(5)), tabs tries a sequence that will work for many terminals.

The margin argument may be used for some terminals. It causes all tabs to be moved over n columns by making column n+1 the left margin. If +m is given without a value of n, the value assumed is 10. For a TermiNet, the first value in the tab list should be 1, or the margin will move even further to the right. The normal (leftmost) margin on most terminals is obtained by +m0. The margin for most terminals is reset only when the +m flag is given explicitly.

Tab and margin setting is performed via the standard output.

EXAMPLES

tabs –a example using –code (canned specification) to set tabs to the settings required by the IBM assembler: columns 1, 10, 16, 36, 72.

tabs –8 example of using –n (repetitive specification), where n is 8, causes tabs to be set every eighth position:
1+(1+8), 1+(2+8), ... which evaluate to columns 9, 17, ...

tabs 1,8,36
example of using n1,n2,... (arbitrary specification) to set tabs at columns 1, 8, and 36.

tabs --$HOME/fspec.list/att4425
example of using --file (file specification) to indicate that tabs should be set according to the first line of $HOME/fspec.list/att4425 (see fspec(4)).

DIAGNOSTICS

illegal tabs when arbitrary tabs are ordered incorrectly
illegal increment when a zero or missing increment is found in an arbitrary specification
unknown tab code when a canned code cannot be found
can't open if --file option used, and file can't be opened
file indication if —file option used and the specification in that file points to yet another file. Indirection of this form is not permitted

SEE ALSO

newform(1), pr(1), tput(1), fspec(4), terminfo(4), environ(5), term(5).

NOTE

There is no consistency among different terminals regarding ways of clearing tabs and setting the left margin.

tabs clears only 20 tabs (on terminals requiring a long sequence), but is willing to set 64.

WARNING

The tabspec used with the tabs command is different from the one used with the newform(1) command. For example, tabs -8 sets every eighth position; whereas newform -1:8 indicates that tabs are set every eighth position.
NAME
tail – deliver the last part of a file

SYNOPSIS
tail [ ±number][lbcrf ]] [ file ]

DESCRIPTION
tail copies the named file to the standard output beginning at a designated place. If no file is named, the standard input is used.

Copying begins at distance +number from the beginning, or -number from the end of the input (if number is null, the value 10 is assumed). Number is counted in units of lines, blocks, or characters, according to the appended option l, b, or c. When no units are specified, counting is by lines.

Specifying r causes tail to print lines from the end of the file in reverse order. The default for r is to print the entire file this way.

With the -f ("follow") option, if the input file is not a pipe, the program will not terminate after the line of the input file has been copied, but will enter an endless loop, wherein it sleeps for a second and then attempts to read and copy further records from the input file. Thus it may be used to monitor the growth of a file that is being written by some other process.

For example, the command:

tail -f fred

will print the last ten lines of the file fred, followed by any lines that are appended to fred between the time tail is initiated and killed. As another example, the command:

tail -15cf fred

will print the last 15 characters of the file fred, followed by any lines that are appended to fred between the time tail is initiated and killed.

SEE ALSO
dd(1M).

BUGS

Tails relative to the end of the file are stored in a buffer, and thus are limited in length. Various kinds of anomalous behavior may happen with character special files.
NAME
tar – tape archiver

SYNOPSIS
tar [ key ] [ name ... ]

DESCRIPTION
tar saves and restores multiple files on a single file (usually a magnetic tape, but it can be any file). tar’s actions are controlled by the key argument. The key is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to tar are file or directory names specifying which files to dump or restore. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

The function portion of the key is specified by one of the following letters:

r The named files are written on the end of the tape. The c function implies this.

x The named files are extracted from the tape. If the named file matches a directory whose contents had been written onto the tape, this directory is (recursively) extracted. The owner, modification time, and mode are restored (if possible). If no file argument is given, the entire content of the tape is extracted. Note that if multiple entries specifying the same file are on the tape, the last one overwrites all earlier.

t The names of the specified files are listed each time they occur on the tape. If no file argument is given, all of the names on the tape are listed.

u The named files are added to the tape if either they are not already there or have been modified since last put on the tape.

c Create a new tape; writing begins on the beginning of the tape instead of after the last file. This command implies r.

The following characters may be used in addition to the function letter.

o On output, tar normally places information specifying owner and modes of directories in the archive. Former versions of tar, when encountering this information will give error message of the form

"<name>/: cannot create".

This modifier will suppress the directory information.

p This modifier says to restore files to their original modes, ignoring the present umask(2). Setuid and sticky information will also be restored to the super-user.

0, ..., 9 This modifier selects an alternate drive on which the tape is mounted. The default is drive 0 at 1600 bpi, which is normally /dev/rmt8.

v Normally tar does its work silently. The v (verbose) option makes tar print the name of each file it treats preceded by the function letter. With the t function, the verbose option gives more information about the tape entries than just their names.

w tar prints the action to be taken followed by file name, then wait for user confirmation. If a word beginning with ‘y’ is given, the action is done. Any other input means don’t do it.

f tar uses the next argument as the name of the archive instead of /dev/rmt?. If the name of the file is ‘-‘, tar writes to standard output or reads from standard input, whichever is appropriate. Thus, tar can be used as the head or tail of a filter chain. tar can also be used to move
hierarchies with the command
   cd fromdir; tar cf - . l (cd todir; tar xf -)

**b**

*tar* uses the next argument as the blocking factor for tape records. The
default is 20 (the maximum). This option should only be used with raw
magnetic tape archives (See f above). The block size is determined
automatically when reading tapes (key letters 'x' and 't').

**l**

 tells *tar* to complain if it cannot resolve all of the links to the files dumped.
If this is not specified, no error messages are printed.

**m**

 tells *tar* not to restore the modification times. The modification time will
be the time of extraction.

**h**

 Force *tar* to follow symbolic links as if they were normal files or direc-
tories. Normally, *tar* does not follow symbolic links.

**I**

 *tar* uses the next argument as a filename that contains a list of files for *tar*
to write to an archive. The list consists of one file per line. Some options
are allowed in the file list, discussed below. The I keyletter is ignored
when reading an archive.

**B**

 Forces input and output blocking to 20 blocks per record. This option
was added so that *tar* can work across a communications channel where
the blocking may not be maintained.

**C**

 If a file name is preceded by -C, then *tar* will perform a *chdir*(2) to that file
name. This allows multiple directories not related by a close common
parent to be archived using short relative path names. For example, to
archive files from /usr/include and from /etc, one might use
   tar c -C /usr include -C /etc

If the I keyletter is used, each filename in the list of files may have one of the follow-
ing options preceding it (separated by a space): -C has the same meaning as above,
causing a *chdir* to the named directory. -d forces a particular directory to be written
without getting its contents. It is necessary to list a directory in this manner before
listing all the files in that directory. -h and +h override the non-existence or
existence of the h keyletter on the command line. The h keyletter sets a flag telling
*tar* to follow symbolic links. The -h option has the same effect on a per-file basis, or
sets the flag for subsequent files if on a line by itself. The +h option resets the flag
either for a particular file or for subsequent files in the same manner.

Previous restrictions dealing with *tar*'s inability to properly handle blocked archives
have been lifted.

**FILES**

 /dev/rmt?
 /tmp/tar*

**SEE ALSO**

 tar(5)

**DIAGNOSTICS**

 Complaints about bad key characters and tape read/write errors.
 Complaints if enough memory is not available to hold the link tables.

**BUGS**

 There is no way to ask for the n-th occurrence of a file.
 Tape errors are handled ungracefully.
 The u option can be slow. When extracting tapes created with the r or u options,
directory modification times may not be set correctly.
The current limit on file name length is 100 characters.
NAME
tcopy – copy a mag tape

SYNOPSIS
tcopy src [ dest ]

DESCRIPTION
Tcopy is designed to copy magnetic tapes. The only assumption made about the tape
is that there are two tape marks at the end. Tcopy with only a source tape specified
will print information about the sizes of records and tape files. If a destination is
specified, then, a copy will be made of the source tape. The blocking on the destina-
tion tape will be identical to that used on the source tape. Copying a tape will yield
the same output as if just printing the sizes.

SEE ALSO
mtio(4)
NAME

tee – pipe fitting

SYNOPSIS

tee [-i] [ -a ] [ file ] ...

DESCRIPTION

tee transcribes the standard input to the standard output and makes copies in the files. The
-i ignore interrupts;
-a causes the output to be appended to the files rather than overwriting them.
NAME
telnet – user interface to the TELNET protocol

SYNOPSIS
telnet [ host [ port ] ]

DESCRIPTION
telnet is used to communicate with another host using the TELNET protocol. If telnet is invoked without arguments, it enters command mode, indicated by its prompt ("telnet>"). In this mode, it accepts and executes the commands listed below. If it is invoked with arguments, it performs an open command (see below) with those arguments.

Once a connection has been opened, telnet enters an input mode. The input mode entered will be either “character at a time” or “line by line” depending on what the remote system supports.

In “character at a time” mode, most text typed is immediately sent to the remote host for processing.

In “line by line” mode, all text is echoed locally, and (normally) only completed lines are sent to the remote host. The “local echo character” (initially “^E”) may be used to turn off and on the local echo (this would mostly be used to enter passwords without the password being echoed).

In either mode, if the localchars toggle is TRUE (the default in line mode; see below), the user’s quit, intr, and flush characters are trapped locally, and sent as TELNET protocol sequences to the remote side. There are options (see toggle autoflush and toggle autosynch below) which cause this action to flush subsequent output to the terminal (until the remote host acknowledges the TELNET sequence) and flush previous terminal input (in the case of quit and intr).

While connected to a remote host, telnet command mode may be entered by typing the telnet “escape character” (initially “^]”). When in command mode, the normal terminal editing conventions are available.

COMMANDS
The following commands are available. Only enough of each command to uniquely identify it need be typed (this is also true for arguments to the mode, set, toggle, and display commands).

open host [ port ]
Open a connection to the named host. If no port number is specified, telnet will attempt to contact a TELNET server at the default port. The host specification may be either a host name (see hosts(5)) or an Internet address specified in the “dot notation” (see inet(3N)).

close
Close a TELNET session and return to command mode.

quit
Close any open TELNET session and exit telnet. An end of file (in command mode) will also close a session and exit.

z
Suspend telnet. This command only works when the user is using the csh(1).

mode type
Type is either line (for “line by line” mode) or character (for “character at a time” mode). The remote host is asked for permission to go into the requested mode. If the remote host is capable of entering that mode, the requested mode will be
entered.

**status**
Show the current status of *telnet*. This includes the peer one is connected to, as well as the current mode.

**display [ argument... ]**
Displays all, or some, of the *set* and *toggle* values (see below).

**? [ command ]**
Get help. With no arguments, *telnet* prints a help summary. If a command is specified, *telnet* will print the help information for just that command.

**send arguments**
Sends one or more special character sequences to the remote host. The following are the arguments which may be specified (more than one argument may be specified at a time):

**escape**
Sends the current *telnet* escape character (initially "\^]").

**synch**
Sends the TELNET SYNCH sequence. This sequence causes the remote system to discard all previously typed (but not yet read) input. This sequence is sent as TCP urgent data (and may not work if the remote system is a 4.2 BSD system -- if it doesn't work, a lower case "\r" may be echoed on the terminal).

**brk**
Sends the TELNET BRK (Break) sequence, which may have significance to the remote system.

**ip**
Sends the TELNET IP (Interrupt Process) sequence, which should cause the remote system to abort the currently running process.

**ao**
Sends the TELNET AO (Abort Output) sequence, which should cause the remote system to flush all output from the remote system to the user's terminal.

**ayt**
Sends the TELNET AYT (Are You There) sequence, to which the remote system may or may not choose to respond.

**ec**
Sends the TELNET EC (Erase Character) sequence, which should cause the remote system to erase the last character entered.

**el**
Sends the TELNET EL (Erase Line) sequence, which should cause the remote system to erase the line currently being entered.

**ga**
Sends the TELNET GA (Go Ahead) sequence, which likely has no significance to the remote system.

**nop**
Sends the TELNET NOP (No OPeration) sequence.

**?**
Prints out help information for the *send* command.
set argument value

Set any one of a number of telnet variables to a specific value. The special value "off" turns off the function associated with the variable. The values of variables may be interrogated with the display command. The variables which may be specified are:

echo

This is the value (initially "^E") which, when in "line by line" mode, toggles between doing local echoing of entered characters (for normal processing), and suppressing echoing of entered characters (for entering, say, a password).

escape

This is the telnet escape character (initially "[^]") which causes entry into telnet command mode (when connected to a remote system).

interrupt

If telnet is in localchars mode (see toggle localchars below) and the interrupt character is typed, a TELNET IP sequence (see send ip above) is sent to the remote host. The initial value for the interrupt character is taken to be the terminal's intr character.

quit

If telnet is in localchars mode (see toggle localchars below) and the quit character is typed, a TELNET BRK sequence (see send brk above) is sent to the remote host. The initial value for the quit character is taken to be the terminal's quit character.

flushoutput

If telnet is in localchars mode (see toggle localchars below) and the flushoutput character is typed, a TELNET AO sequence (see send ao above) is sent to the remote host. The initial value for the flush character is taken to be the terminal's flush character.

erase

If telnet is in localchars mode (see toggle localchars below), and if telnet is operating in "character at a time" mode, then when this character is typed, a TELNET EC sequence (see send ec above) is sent to the remote system. The initial value for the erase character is taken to be the terminal's erase character.

kill

If telnet is in localchars mode (see toggle localchars below), and if telnet is operating in "character at a time" mode, then when this character is typed, a TELNET EL sequence (see send el above) is sent to the remote system. The initial value for the kill character is taken to be the terminal's kill character.

eof

If telnet is operating in "line by line" mode, entering this character as the first character on a line will cause this character to be sent to the remote system. The initial value of the eof character is taken to be the terminal's eof character.

toggle arguments...

Toggle (between TRUE and FALSE) various flags that control how telnet responds to events. More than one argument may be specified. The state of these flags may be interrogated with the display command. Valid arguments are:
localchars
If this is TRUE, then the flush, interrupt, quit, erase, and kill characters (see set above) are recognized locally, and transformed into (hopefully) appropriate TELNET control sequences (respectively ao, ip, brk, ec, and el; see send above). The initial value for this toggle is TRUE in "line by line" mode, and FALSE in "character at a time" mode.

autoflush
If autoflush and localchars are both TRUE, then when the ao, intr, or quit characters are recognized (and transformed into TELNET sequences; see set above for details), telnet refuses to display any data on the user's terminal until the remote system acknowledges (via a TELNET Timing Mark option) that it has processed those TELNET sequences. The initial value for this toggle is TRUE if the terminal user had not done an "sitty nofish", otherwise FALSE (see stty(1)).

autosynch
If autosynch and localchars are both TRUE, then when either the intr or quit characters is typed (see set above for descriptions of the intr and quit characters), the resulting TELNET sequence sent is followed by the TELNET SYCH sequence. This procedure should cause the remote system to begin throwing away all previously typed input until both of the TELNET sequences have been read and acted upon. The initial value of this toggle is FALSE.

crmod
Toggle carriage return mode. When this mode is enabled, most carriage return characters received from the remote host will be mapped into a carriage return followed by a line feed. This mode does not affect those characters typed by the user, only those received from the remote host. This mode is not very useful unless the remote host only sends carriage return, but never line feed. The initial value for this toggle is FALSE.

debug
Toggles socket level debugging (useful only to the superuser). The initial value for this toggle is FALSE.

options
Toggles the display of some internal telnet protocol processing (having to do with TELNET options). The initial value for this toggle is FALSE.

netdata
Toggles the display of all network data (in hexadecimal format). The initial value for this toggle is FALSE.

? Displays the legal toggle commands.

BUGS
There is no adequate way for dealing with flow control.
On some remote systems, echo has to be turned off manually when in "line by line" mode.
There is enough settable state to justify a .telnetrc file.
No capability for a .telnetrc file is provided.
In "line by line" mode, the terminal's eof character is only recognized (and sent to the remote system) when it is the first character on a line.
NAME

test -- condition evaluation command

SYNOPSIS

test expr
[ expr ]

DESCRIPTION

test evaluates the expression expr and, if its value is true, sets a zero (true) exit status;
otherwise, a non-zero (false) exit status is set; test also sets a non-zero exit status if
there are no arguments. When permissions are tested, the effective user ID of the
process is used.

All operators, flags, and brackets (brackets used as shown in the second SYNOPSIS
line) must be separate arguments to the test command; normally these items are
separated by spaces.

The following primitives are used to construct expr:

- r file true if file exists and is readable.
- w file true if file exists and is writable.
- x file true if file exists and is executable.
- f file true if file exists and is a regular file.
- d file true if file exists and is a directory.
- c file true if file exists and is a character special file.
- b file true if file exists and is a block special file.
- p file true if file exists and is a named pipe (fifo).
- u file true if file exists and its set-user-ID bit is set.
- g file true if file exists and its set-group-ID bit is set.
- k file true if file exists and its sticky bit is set.
- l file true if file is a symbolic link
- s file true if file exists and has a size greater than zero.
- t [ fildes ] true if the open file whose file descriptor number is fildes (1 by default)
is associated with a terminal device.

- z s1 true if the length of string s1 is zero.
- n s1 true if the length of the string s1 is non-zero.
s1 = s2 true if strings s1 and s2 are identical.
s1 != s2 true if strings s1 and s2 are not identical.
s1 true if s1 is not the null string.
n1 -eq n2 true if the integers n1 and n2 are algebraically equal. Any of the com-
parisons -ne, -gt, -ge, -lt, and -le may be used in place of -eq.

These primaries may be combined with the following operators:

! unary negation operator.
-a binary and operator.
-o binary or operator (¬a has higher precedence than ¬o).
( expr ) parentheses for grouping. Notice also that parentheses are meaningful to the shell and, therefore, must be quoted.

SEE ALSO

find(1), sh(1).

WARNING

If you test a file you own (the -r, -w, or -x tests), but the permission tested does not have the owner bit set, a non-zero (false) exit status will be returned even though the file may have the group or other bit set for that permission. The correct exit status will be set if you are super-user.

The = and != operators have a higher precedence than the -r through -n operators, and = and != always expect arguments; therefore, = and != cannot be used with the -r through -n operators.

If more than one argument follows the -r through -n operators, only the first argument is examined; the others are ignored, unless a -a or a -o is the second argument.
NAME
tftp – transfer files using DARPA Trivial File Transfer Protocol

SYNOPSIS
tftp [ host ]

DESCRIPTION
tftp is the user interface to the Internet TFTP (Trivial File Transfer Protocol), which allows users to transfer files to and from a remote machine. The remote host may be specified on the command line, in which case tftp uses host as the default host for future transfers (see the connect command below).

COMMANDS
Once tftp is running, it issues the prompt tftp> and recognizes the following commands:

connect host-name [ port ]
Set the host (and optionally port) for transfers. Note that the TFTP protocol, unlike the FTP protocol, does not maintain connections between transfers; thus, the connect command does not actually create a connection, but merely remembers what host is to be used for transfers. You do not have to use the connect command; the remote host can be specified as part of the get or put commands.

mode transfer-mode
Set the mode for transfers; transfer-mode may be one of ascii or binary. The default is ascii.

put file
put localfile remotefile
put file1 file2 ... fileN remote-directory
Put a file or set of files to the specified remote file or directory. The destination can be in one of two forms: a filename on the remote host, if the host has already been specified, or a string of the form host:filename to specify both a host and filename at the same time. If the latter form is used, the hostname specified becomes the default for future transfers. If the remote-directory form is used, the remote host is assumed to be a UNIX machine.

get filename
get remotename localname
get file1 file2 ... fileN
Get a file or set of files from the specified sources. Source can be in one of two forms: a filename on the remote host, if the host has already been specified, or a string of the form host:filename to specify both a host and filename at the same time. If the latter form is used, the last hostname specified becomes the default for future transfers.

quit Exit tftp. An end of file also exits.

verbose
Toggle verbose mode.

trace Toggle packet tracing.

status
Show current status.

rexmt retransmission-timeout
Set the per-packet retransmission timeout, in seconds.
timeout total-transmission-timeout
   Set the total transmission timeout, in seconds.

ascii  Shorthand for "mode ascii"

binary  Shorthand for "mode binary"

? [ command-name ... ]
   Print help information.

SEE ALSO
tftpd(8C), RFC 783 – The TFTP Protocol (Network Information Center, SRI International)

BUGS
Because there is no user-login or validation within the TFTP protocol, the remote site will probably have some sort of file-access restrictions in place. The exact methods are specific to each site and therefore difficult to document here.

tftp does nothing special with "mail" mode transfers. Most servers don't handle the mail mode either.
tic – terminfo compiler

tic [-v[n]] [-c] file

tic translates a terminfo(4) file from the source format into the compiled format. The results are placed in the directory /usr/lib/terminfo. The compiled format is necessary for use with the library routines described in curses(3X).

- vn (verbose) output to standard error trace information showing tic’s progress. The optional integer n is a number from 1 to 10, inclusive, indicating the desired level of detail of information. If n is omitted, the default level is 1. If n is specified and greater than 1, the level of detail is increased.

- c only check file for errors. Errors in use= links are not detected.

file contains one or more terminfo(4) terminal descriptions in source format (see terminfo(4)). Each description in the file describes the capabilities of a particular terminal. When a use=entry-name field is discovered in a terminal entry currently being compiled, tic reads in the binary from /usr/lib/terminfo to complete the entry. (Entries created from file will be used first. If the environment variable TERMINFO is set, that directory is searched instead of /usr/lib/terminfo.) tic duplicates the capabilities in entry-name for the current entry, with the exception of those capabilities that explicitly are defined in the current entry.

If the environment variable TERMINFO is set, the compiled results are placed there instead of /usr/lib/terminfo.

FILES

/usr/lib/terminfo/?.*/ compiled terminal description data base

SEE ALSO

curses(3X), term(4), terminfo(4).

WARNINGS

Total compiled entries cannot exceed 4096 bytes. The name field cannot exceed 128 bytes.

Terminal names exceeding 14 characters will be truncated to 14 characters and a warning message will be printed.

When the –c option is used, duplicate terminal names will not be diagnosed; however, when –c is not used, they will be.

BUGS

To allow existing executables from the previous release of the UNIX System to continue to run with the compiled terminfo entries created by the new terminfo compiler, cancelled capabilities will not be marked as cancelled within the terminfo binary unless the entry name has a ‘+’ within it. (Such terminal names are only used for inclusion within other entries via a use= entry. Such names would not be used for real terminal names.)

For example:
4415+nl, kf1@, kf2@, ....

4415+base, kf1=\EOc, kf2=\EOd, ....

4415-nl 4415 terminal without keys,
use=4415+nl, use=4415+base,

The above example works as expected; the definitions for the keys do not show up in the 4415-nl entry. However, if the entry 4415+nl did not have a plus sign within its name, the cancellations would not be marked within the compiled file and the definitions for the function keys would not be cancelled within 4415-nl.

**DIAGNOSTICS**

Most diagnostic messages produced by *tic* during the compilation of the source file are preceded with the approximate line number and the name of the terminal currently being worked on.

*mkdir* ... returned bad status
The named directory could not be created.

File does not start with terminal names in column one
The first thing seen in the file, after comments, must be the list of terminal names.

Token after a *seek*(2) not NAMES
Somehow the file being compiled changed during the compilation.

Not enough memory for use_list element
or
Out of memory
Not enough free memory was available (*malloc*(3) failed).

Can’t open ...
The named file could not be created.

Error in writing ...
The named file could not be written to.

Can’t link ... to ...
A link failed.

Error in re-reading compiled file ...
The compiled file could not be read back in.

Premature EOF
The current entry ended prematurely.

Backspaced off beginning of line
This error indicates something wrong happened within *tic*.

Unknown Capability - "...
The named invalid capability was found within the file.

Wrong type used for capability "...
For example, a string capability was given a numeric value.

Unknown token type
Tokens must be followed by '@' to cancel, '/' for booleans, '#' for numbers, or '=' for strings.

"...": bad term name
or
Line ...
Illegal terminal name - "...
Terminal names must start with a letter or digit
   The given name was invalid. Names must not contain white space or slashes,
   and must begin with a letter or digit.

"...": terminal name too long.
   An extremely long terminal name was found.

"...": terminal name too short.
   A one-letter name was found.

"...", filename too long, truncating to "...
   The given name was truncated to 14 characters due to UNIX file name length
   limitations.

"...", defined in more than one entry. Entry being used is "...
   An entry was found more than once.

Terminal name "...", synonym for itself
   A name was listed twice in the list of synonyms.

At least one synonym should begin with a letter.
   At least one of the names of the terminal should begin with a letter.

Illegal character - "...
   The given invalid character was found in the input file.

Newline in middle of terminal name
   The trailing comma was probably left off of the list of names.

Missing comma
   A comma was missing.

Missing numeric value
   The number was missing after a numeric capability.

NULL string value
   The proper way to say that a string capability does not exist is to cancel it.

Very long string found. Missing comma?
   self-explanatory

Unknown option. Usage is:
   An invalid option was entered.

Too many file names. Usage is:
   self-explanatory

"...", non-existant or permission denied
   The given directory could not be written into.

"...", is not a directory
   self-explanatory

"...", Permission denied
   access denied.

"...": Not a directory
   tic wanted to use the given name as a directory, but it already exists as a file

SYSTEM ERROR!! Fork failed!!!
   A fork(2) failed.
   Error in following up use-links. Either there is a loop in the links or they
   reference non-existant terminals. The following is a list of the entries
   involved:
A `terminfo(4)` entry with a `use=name` capability either referenced a non-existent terminal called `name` or `name` somehow referred back to the given entry.
NAME

time – time a command

SYNOPSIS

time command

DESCRIPTION

The *command* is executed; after it is complete, *time* prints the elapsed time during the
command, the time spent in the system, and the time spent in execution of the com-
mand. Times are reported in seconds.

The times are printed on standard error.

SEE ALSO

See *csh(1)* for a description of the *csh* built-in command *time*.

times(2).
NAME
timex – time a command; report process data and system activity

SYNOPSIS
timex [ options ] command

DESCRIPTION
The given \textit{command} is executed; the elapsed time, user time and system time spent in
execution are reported in seconds. Optionally, process accounting data for the \textit{command}
and all its children can be listed or summarized, and total system activity during
the execution interval can be reported.
The output of \textit{timex} is written on standard error.

Options are:
\begin{itemize}
\item \texttt{\textasciitilde p}\quad List process accounting records for \textit{command} and all its children. Suboptions \texttt{f}, \texttt{h}, \texttt{k}, \texttt{m}, \texttt{r}, and \texttt{t} modify the data items reported. The options are as follows:
  \begin{itemize}
  \item \texttt{\textasciitilde f}\quad Print the \texttt{fork/exec} flag and system exit status columns in the output.
  \item \texttt{\textasciitilde h}\quad Instead of mean memory size, show the fraction of total available CPU
time consumed by the process during its execution. This “hog factor” is
  computed as:
  \begin{equation}
  \frac{\text{total CPU time}}{\text{elapsed time}}.
  \end{equation}
  \item \texttt{\textasciitilde k}\quad Instead of memory size, show total \texttt{kcore}-minutes.
  \item \texttt{\textasciitilde m}\quad Show mean core size (the default).
  \item \texttt{\textasciitilde r}\quad Show CPU factor (user time/(system-time + user-time)).
  \item \texttt{\textasciitilde t}\quad Show separate system and user CPU times. The number of blocks read
  or written and the number of characters transferred are always reported.
  \end{itemize}
\item \texttt{\textasciitilde o}\quad Report the total number of blocks read or written and total characters
  transferred by \textit{command} and all its children.
\item \texttt{\textasciitilde s}\quad Report total system activity (not just that due to \textit{command}) that occurred during
  the execution interval of \textit{command}. All the data items listed in \texttt{sar(1)} are
  reported.
\end{itemize}

SEE ALSO
sar(1).

WARNING
Process records associated with \textit{command} are selected from the accounting file
\texttt{/usr/adm/pacct} by inference, since process genealogy is not available. Background
processes having the same user-id, terminal-id, and execution time window will be
spuriously included.

EXAMPLES
A simple example:
\begin{verbatim}
timex \textasciitilde ops sleep 60
\end{verbatim}
A terminal session of arbitrary complexity can be measured by timing a sub-shell:
\begin{verbatim}
timex \textasciitilde opskmt sh
  session commands
  EOT
\end{verbatim}
NAME

touch – update access and modification times of a file

SYNOPSIS

touch [ -amc ] [ mmddhhmm[yy] ] files

DESCRIPTION

touch causes the access and modification times of each argument to be updated. The file name is created if it does not exist. If no time is specified (see date(1)) the current time is used. The -a and -m options cause touch to update only the access or modification times respectively (default is -am). The -c option silently prevents touch from creating the file if it did not previously exist.

The return code from touch is the number of files for which the times could not be successfully modified (including files that did not exist and were not created).

SEE ALSO

date(1), utime(2).
**NAME**

tput – initialize a terminal or query terminfo database

**SYNOPSIS**

tput [-Ttype] capname [parms ...]
tput [-Ttype] init
tput [-Ttype] reset
tput [-Ttype] longname

**DESCRIPTION**

tput uses the terminfo(4) database to make the values of terminal-dependent capabilities and information available to the shell (see sh(1)), to initialize or reset the terminal, or return the long name of the requested terminal type. tput outputs a string if the attribute (capability name) is of type string, or an integer if the attribute is of type integer. If the attribute is of type boolean, tput simply sets the exit code (0 for TRUE if the terminal has the capability, 1 for FALSE if it does not), and produces no output. Before using a value returned on standard output, the user should test the exit code ($?, see sh(1)) to be sure it is 0. (See EXIT CODES and DIAGNOSTICS below.) For a complete list of capabilities and the capname associated with each, see terminfo(4).

-Ttype indicates the type of terminal. Normally this option is unnecessary, because the default is taken from the environment variable TERM. If -T is specified, then the shell variables LINES and COLUMNS and the layer size (see layers(1)) will not be referenced.

capname indicates the attribute from the terminfo(4) database.

parms If the attribute is a string that takes parameters, the arguments parms will be instantiated into the string. An all numeric argument will be passed to the attribute as a number.

init If the terminfo(4) database is present and an entry for the user’s terminal exists (see -Ttype, above), the following will occur: (1) if present, the terminal’s initialization strings will be output (is1, is2, is3, if, iprog), (2) any delays (e.g., newline) specified in the entry will be set in the tty driver, (3) tabs expansion will be turned on or off according to the specification in the entry, and (4) if tabs are not expanded, standard tabs will be set (every 8 spaces). If an entry does not contain the information needed for any of the four above activities, that activity will silently be skipped.

reset Instead of putting out initialization strings, the terminal’s reset strings will be output if present (rs1, rs2, rs3, rf). If the reset strings are not present, but initialization strings are, the initialization strings will be output. Otherwise, reset acts identically to init.

longname If the terminfo(4) database is present and an entry for the user’s terminal exists (see -Ttype above), then the long name of the terminal will be put out. The long name is the last name in the first line of the terminal’s description in the terminfo(4) database (see term(5)).

**EXAMPLES**

tput init Initialize the terminal according to the type of terminal in the environmental variable TERM. This command should be included in everyone’s .profile after the environmental variable TERM has been exported, as illustrated on the profile(4) manual page.
tput -T5620 reset
Reset an AT&T 5620 terminal, overriding the type of terminal in the environmental variable TERM.

tput cup 0 0
Send the sequence to move the cursor to row 0, column 0 (the upper left corner of the screen, usually known as the "home" cursor position).

tput clear
Echo the clear-screen sequence for the current terminal.

tput cols
Print the number of columns for the current terminal.

tput -T450 cols
Print the number of columns for the 450 terminal.

bold='tput smso'
offbold='tput rmso'
Set the shell variables bold, to begin stand-out mode sequence, and offbold, to end standout mode sequence, for the current terminal. This might be followed by a prompt:
echo "$(bold)Please type in your name: $(offbold)\n"

tput hc
Set exit code to indicate if the current terminal is a hardcopy terminal.

tput cup 23 4
Send the sequence to move the cursor to row 23, column 4.

tput longname
Print the long name from the terminfo(4) database for the type of terminal specified in the environmental variable TERM.

FILES
/usr/lib/terminfo/*
compiled terminal description database
/usr/include/curses.h
curses(3X) header file
/usr/include/term.h
terminfo(4) header file
/usr/lib/tabset/*
tab settings for some terminals, in a format appropriate to be output to the terminal (escape sequences that set margins and tabs); for more information, see the "Tabs and Initialization" section of terminfo(4)

SEE ALSO
stty (1), tabs (1), profile(4), terminfo(4).

EXIT CODES
If capname is of type boolean, a value of 0 is set for TRUE and 1 for FALSE.

If capname is of type string, a value of 0 is set if the capname is defined for this terminal type (the value of capname is returned on standard output); a value of 1 is set if capname is not defined for this terminal type (a null value is returned on standard output).

If capname is of type integer, a value of 0 is always set, whether or not capname is defined for this terminal type. To determine if capname is defined for this terminal type, the user must test the value of standard output. A value of -1 means that capname is not defined for this terminal type.

Any other exit code indicates an error; see DIAGNOSTICS, below.

DIAGNOSTICS
tput prints the following error messages and sets the corresponding exit codes.

exit
code
error message

0
-1 (capname is a numeric variable that is not specified in the terminfo(4) database for this terminal type, e.g.
tput -T450 lines and tput -T2621 xmc)
<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>no error message is printed, see EXIT CODES, above.</td>
</tr>
<tr>
<td>2</td>
<td>usage error</td>
</tr>
<tr>
<td>3</td>
<td>unknown terminal type or no terminfo(4) database</td>
</tr>
<tr>
<td>4</td>
<td>unknown terminfo(4) capability capname</td>
</tr>
</tbody>
</table>
NAME

tr – translate characters

SYNOPSIS

tr [-cds] [ string1 [ string2 ]]

DESCRIPTION

tr copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in string1 are mapped into the corresponding characters of string2. Any combination of the options -cds may be used:

-c Complements the set of characters in string1 with respect to the universe of characters whose ASCII codes are 001 through 377 octal.

-d Deletes all input characters in string1.

-s Squeezes all strings of repeated output characters that are in string2 to single characters.

The following abbreviation conventions may be used to introduce ranges of characters or repeated characters into the strings:

[a-z] Stands for the string of characters whose ASCII codes run from character a to character z, inclusive.

[a*n] Stands for n repetitions of a. If the first digit of n is 0, n is considered octal; otherwise, n is taken to be decimal. A zero or missing n is taken to be huge; this facility is useful for padding string2.

The escape character \ may be used as in the shell to remove special meaning from any character in a string. In addition, \ followed by 1, 2, or 3 octal digits stands for the character whose ASCII code is given by those digits.

EXAMPLE

The following example creates a list of all the words in file1 one per line in file2, where a word is taken to be a maximal string of alphabets. The strings are quoted to protect the special characters from interpretation by the shell; 012 is the ASCII code for newline.

    tr -cs "[A-Z][a-z]" "[\012*]" <file1 >file2

SEE ALSO

ed(1), sh(1), ascii(5).

BUGS

Will not handle ASCII NUL in string1 or string2; always deletes NUL from input.
NAME
true, false – provide truth values

SYNOPSIS
true
false

DESCRIPTION
true does nothing, successfully. False does nothing, unsuccessfully. They are typically used in input to sh(1) such as:

while true
do command done

SEE ALSO
sh(1).

DIAGNOSTICS
true has exit status zero, false nonzero.
TSET(1)

NAME
tset, reset – terminal dependent initialization

SYNOPSIS
tset [ options ] [-m [ident][test baudrate]:type ] ... [ type ]
reset [ options ] [-m [ident][test baudrate]:type ] ... [ type ]

DESCRIPTION
tset sets up your terminal when you first log in to a UNIX system. It does terminal
dependent processing such as setting erase and kill characters, setting or resetting
delays, sending any sequences needed to properly initialize the terminal, and the
like. It first determines the type of terminal involved, and then does necessary initial-
izations and mode settings. Type names for terminals may be found in the termcap(5)
database. If a port is not wired permanently to a specific terminal (not hardwired) it
will be given an appropriate generic identifier such as dialup.

In the case where no arguments are specified, tset simply reads the terminal type out
of the environment variable TERM and re-initializes the terminal. The rest of this
manual concerns itself with mode and environment initialization, typically done
once at login, and options used at initialization time to determine the terminal type
and set up terminal modes.

When used in a startup script (.profile for sh(1) users or .login for csh(1) users) it is
desirable to give information about the type of terminal you will usually use on ports
which are not hardwired. These ports are identified in /etc/tty as dialup or plugboard
or network, etc. To specify what terminal type you usually use on these ports, the -m
(map) option flag is followed by the appropriate port type identifier, an optional
baud rate specification, and the terminal type. (The effect is to "map" from some
conditions to a terminal type, that is, to tell tset "If I'm on this kind of port, guess that
I'm on that kind of terminal".) If more than one mapping is specified, the first appli-
cable mapping prevails. A missing port type identifier matches all identifiers. Any
of the alternate generic names given in termcap may be used for the identifier.

A baudrate is specified as with stty(1), and is compared with the speed of the diagno-
sic output (which should be the control terminal). The baud rate test may be any
combination of: >, @, <, and !; @ means "at" and ! inverts the sense of the test. To
avoid problems with metacharacters, it is best to place the entire argument to -m
within "" characters; users of csh(1) must also put a "\" before any "!" used here.
Thus

```
tset -m 'dialup>300:adm3a' -m dialup:dw2 -m 'plugboard:adm3a'
```

causes the terminal type to be set to an adm3a if the port in use is a dialup at a speed
greater than 300 baud; to a dw2 if the port is (otherwise) a dialup (i.e. at 300 baud or
less). If the type finally determined by tset begins with a question mark, the user is
asked if s/he really wants that type. A null response means to use that type; other-
wise, another type can be entered which will be used instead. Thus, in the above
case, the user will be queried on a plugboard port as to whether they are actually
using an adm3a.

If no mapping applies and a final type option, not preceded by a -m, is given on the
command line then that type is used; otherwise the type found in the /etc/tty file
database will be taken to be the terminal type. This should always be the case for
hardwired ports.

It is usually desirable to return the terminal type, as finally determined by tset, and
information about the terminal's capabilities to a shell's environment. This can be
done using the - option; using the Bourne shell, sh(1):
export TERM; TERM='tset - options...'

or using the C shell, csh(1):

    setenv TERM 'tset - options...'

With csh it is preferable to use the following command in your .login file to initialize the TERM and TERMCAP environment variables at the same time.

    eval 'tset -s options...'

(The -s option will produce either C shell or Bourne shell commands based on the value of the SHELL environment variable.)

It is also convenient to make an alias in your .cshrc:

    alias Tset 'set noglob; eval 'tset -s \!*''

This allows the command:

    Tset 2621

to be invoked at any time to set the terminal and environment. Note to Bourne Shell users: It is not possible to get this aliasing effect with a shell script, because shell scripts cannot set the environment of their parent. (If a process could set its parent's environment, none of this nonsense would be necessary in the first place.)

These commands cause tset to place the name of your terminal in the variables TERM and TERMCAP in the environment; see environ(5). NOTE: System V UNIX systems do not in general make use of the TERMCAP environment variable, since System V usually uses the compiled form of termcap called terminfo instead. It is useful, however, to set TERMCAP if you are making any extensive use of 4.2 or 4.3 BSD utilities that do use termcap.

Once the terminal type is known, tset engages in terminal driver mode setting. This normally involves sending an initialization sequence to the terminal, setting the single character erase (and optionally the line-kill (full line erase)) characters, and setting special character delays. Tab and newline expansion are turned off during transmission of the terminal initialization sequence.

On terminals that can backspace but not overstrike (such as a CRT), and when the erase character is the default erase character ('h on standard systems), the erase character is changed to BACKSPACE (Control-H).

The options are:

- `--ec` set the erase character to be the named character c on all terminals, the default being the backspace character on the terminal, usually ^H. The character c can either be typed directly, or entered using the hat notation used here.

- `--Ec` set the erase character to c on all terminals except those which cannot backspace (e.g., a TTY 33). c defaults to control-H.

- `--kc` is similar to `--e` but for the line kill character rather than the erase character; c defaults to ^X (for purely historical reasons). The kill character is left alone if `--k` is not specified. The hat notation can also be used for this option.

- `--ic` is similar to `--e` but for the interrupt character rather than the erase character; c defaults to ^C. The hat notation can also be used for this option.

- The name of the terminal finally decided upon is output on the standard output. This is intended to be captured by the shell and placed in the environment variable TERM.
-s Print the sequence of csh or sh commands to initialize the environment variables TERM and TERMCAPE based on the name of the terminal finally decided upon. This option will produce either C shell or Bourne shell commands based on the value of the SHELL environment variable.

-S -- Similar to -s but outputs 2 strings suitable for use in csh .login files as follows:
set noglob
set term="(tset -S ....."
setenv TERM $term[1]
setenv TERMCAPE "$term[2]"
unset term
unset noglob

-m map the system identified type to some user specified type. The mapping can be baud rate dependent. The syntax is:
  -m identifier [test baudrate] :type
where: "identifier" can be any of the alternate generic names given in termcap (absence of an identifier matches any identifier); "test" may be any combination of \( > = < ! \) ; "baudrate" is as with stty(1); "type" is the actual terminal type to use if the mapping condition is met. Multiple maps are scanned in order and the first match prevails.

-A ask user for terminal type.

-n On systems with the Berkeley 4BSD tty driver, specifies that the new tty driver modes should be initialized for this terminal. For a CRT, the CRTERASE and CRTKILL modes are set only if the baud rate is 1200 or greater. Not applicable to the Stardent 1500/3000.

-v On virtual terminal systems, don't set up a virtual terminal. Otherwise tset will tell the operating system what kind of terminal you are on (if it is a known terminal) and fix up the output of -s to use virtual terminal sequences. Not applicable to the Stardent 1500/3000.

-r report to user, in addition to other flags.

-I suppresses transmitting terminal initialization strings.

-Q suppresses printing the "Erase set to" and "Kill set to" messages.

If tset is invoked as reset, it will set cooked and echo modes, turn off cbreak and raw modes, turn on newline translation, and restore special characters to a sensible state before any terminal dependent processing is done. Any special character that is found to be NULL or "-1" is reset to its default value. All arguments to tset may be used with reset.

This is most useful after a program dies leaving a terminal in a funny state. You may have to type "<LF>reset<LF>" to get it to work since <CR> may not work in this state. Often none of this will echo.

EXAMPLES

In the csh, a user might typically put a call to tset in his .login file. The following example indicates that if the user has dialed in, he is on a vt100 and if he's coming in through the network, his hardwired terminal type should be used. (If the user is working from a hardwired port, his terminal type will be set to $TERM.) The -s flag causes the environment variables TERM and TERMCAPE to be set. The erase character is set to backspace (^H). The "set noglob" is necessary to turn off csh's processing of special characters. (No "unset noglob" is necessary because the -s flag automatically produces one.)
set noglob; eval `tset -s -e -Q -m dialup:vt100 -m network:TERM`

In addition, the csh user might like to include the following alias in his .cshrc to allow easy resetting of the terminal type:

```
alias Tset "set noglob; eval `tset -s \!`"
```

The rest of the examples all assume the Bourne shell and use the – option. If you use csh, use one of the variations described above. Note that a typical use of tset in a .profile or .login will also use the –e and –k options, and often the –m or –Q options as well. These options have not been included here to keep the examples small. (NOTE: Some of the examples given here appear to take up more than one line, for text processing reasons. When you type in real tset commands, you must enter them entirely on one line.)

At the moment, you are on a 2621. This is suitable for typing by hand but not for a .profile, unless you are always on a 2621.

```
export TERM; TERM=`tset –2621`
```

You have an h19 at home which you dial up on, but your office terminal is hardwired and known in /etc/ttys.

```
export TERM; TERM=`tset –m dialup:h19`
```

You have a switch which connects everything to everything, making it nearly impossible to key on what port you are coming in on. You use a vt100 in your office at 9600 baud, and dial up to switch ports at 1200 baud from home on a 2621. Sometimes you use someone else’s terminal at work, so you want it to ask you to make sure what terminal type you have at high speeds, but at 1200 baud you are always on a 2621. Note the placement of the question mark, and the quotes to protect the greater than and question mark from interpretation by the shell.

```
export TERM; TERM=`tset –m 'switch>1200?vt100' –m 'switch<=1200:2621`
```

All of the above entries will fall back on the terminal type specified in /etc/ttys if none of the conditions hold. The following entry is appropriate if you always dial up, always at the same baud rate, on many different kinds of terminals. Your most common terminal is an adm3a. It always asks you what kind of terminal you are on, defaulting to adm3a.

```
export TERM; TERM=`tset –?adm3a`
```

If the file /etc/ttys is not properly installed and you want to key entirely on the baud rate, the following can be used:

```
export TERM; TERM=`tset –m ’>1200:vt100’ 2621`
```

Here is a fancy example to illustrate the power of tset and to hopelessly confuse anyone who has made it this far. You dial up at 1200 baud or less on a Concept100, sometimes over switch ports and sometimes over regular dialups. You use various terminals at speeds higher than 1200 over switch ports, most often the terminal in your office, which is a vt100. However, sometimes you log in from the university you used to go to, over the ARPANET; in this case you are on an ALTO emulating a dm2500. You also often log in on various hardwired ports, such as the console, all of which are properly entered in /etc/ttys. You want your erase character set to control H, your kill character set to control U, and don’t want tset to print the “Erase set to Backspace, Kill set to Control U” message.

```
export TERM; TERM=`tset –e –k^U –Q – m ’switch<=1200:concept100’ –m ’switch?:vt100’ –m dialup:concept100 –m arpanet:dm2500`
```
FILES

/etc/ttys port name to terminal type mapping database
/etc/termcap terminal capability database

SEE ALSO
csh(1), environ(5), sh(1), stty(1), termcap(5)

AUTHORS
Eric Allman
David Wasley
Mark Horton

BUGS

The tset command is one of the first commands a user must master when getting started on a UNIX system. Unfortunately, it is one of the most complex, largely because of the extra effort the user must go through to get the environment of the login shell set. Something needs to be done to make all this simpler, either the login(1) program should do this stuff, or a default shell alias should be made, or a way to set the environment of the parent should exist.

This program can’t intuit personal choices for erase, interrupt and line kill characters, so it leaves these set to the local system standards.
NAME
tsort – topological sort

SYNOPSIS
tsor [file]

DESCRIPTION
The tsort command produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input file. If no file is specified, the standard input is understood.

The input consists of pairs of items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

DIAGNOSTICS
Odd data: there is an odd number of fields in the input file.
NAME
tty – get the name of the terminal

SYNOPSIS
tty [-l] [-s]

DESCRIPTION
tty prints the path name of the user’s terminal.
-1  prints the synchronous line number to which the user’s terminal is connected, if it is on an active synchronous line.
-s  inhibits printing of the terminal path name, allowing one to test just the exit code.

EXIT CODES
  2   if invalid options were specified,
  0   if standard input is a terminal,
  1   otherwise.

DIAGNOSTICS
“not on an active synchronous line” if the standard input is not a synchronous terminal and -l is specified.
“not a tty” if the standard input is not a terminal and -s is not specified.
NAME

uadmin – administrative control

SYNOPSIS

/etc/uadmin cmd fcn

DESCRIPTION

The uadmin command provides control for basic administrative functions. This command is tightly coupled to the system administration procedures and is not intended for general use. It may be invoked only by the superuser.

The arguments cmd (command) and fcn (function) are converted to integers and passed to the uadmin(2) system call.

SEE ALSO

uadmin(2)
NAME

ul – do underlining

SYNOPSIS

ul [-i] [-t terminal] [name ...]

DESCRIPTION

ul reads the named files (or standard input if none are given) and translates occurrences of underscores to the sequence which indicates underlining for the terminal in use, as specified by the environment variable TERM. The -t option overrides the terminal kind specified in the environment. The file /etc/termcap is read to determine the appropriate sequences for underlining. If the terminal is incapable of underlining, but is capable of a standout mode then that is used instead. If the terminal can overstrike, or handles underlining automatically, ul degenerates to cat(1). If the terminal cannot underline, underlining is ignored.

The -i option causes ul to indicate underlining onto by a separate line containing appropriate dashes ‘-’; this is useful when you want to look at the underlining which is present in an nroff output stream on a crt-terminal.

SEE ALSO

man(1), nroff(1), colcrt(1)

BUGS

nroff usually outputs a series of backspaces and underlines intermixed with the text to indicate underlining. No attempt is made to optimize the backward motion.
NAME

umask – set file-creation mode mask

SYNOPSIS

umask [ ooo ]

DESCRIPTION

The user file-creation mode mask is set to ooo. The three octal digits refer to read/write/execute permissions for owner, group, and others, respectively (see chmod(2) and umask(2)). The value of each specified digit is subtracted from the corresponding “digit” specified by the system for the creation of a file (see creat(2)). For example, umask 022 removes group and others write permission (files normally created with mode 777 become mode 755; files created with mode 666 become mode 644).

If ooo is omitted, the current value of the mask is printed.

umask is recognized and executed by the shell.

umask can be included in the user’s .profile (see profile(4)) and invoked at login to automatically set the user’s permissions on files or directories created.

SEE ALSO

chmod(1), sh(1), chmod(2), creat(2), umask(2), profile(4).
NAME

uname – print name of current UNIX system

SYNOPSIS

uname [ –snrvma ]
uname [ –S system name ]

DESCRIPTION

uname prints the current system name of the UNIX system on the standard output file. It is mainly useful to determine which system one is using. The options cause selected information returned by uname(2) to be printed:

–s    print the system name (default).
–n    print the nodename (the nodename is the name by which the system is known to a communications network).
–r    print the operating system release.
–v    print the operating system version.
–m    print the machine hardware name.
–a    print all the above information.

The system name and the nodename may be changed by specifying a system name argument to the –S option. The system name argument is restricted to 8 characters. Only the super-user is allowed this capability.

SEE ALSO

uname(2).
Source Code Control System Utilities

UNGET(1)

NAME
unget – undo a previous get of an SCCS file

SYNOPSIS
unget [-rSID] [-s] [-n] files

DESCRIPTION
unget undoes the effect of a get -e done prior to creating the intended new delta. If a
directory is named, unget behaves as though each file in the directory were specified
as a named file, except that non-SCCS files and unreadable files are silently ignored.
If a name of – is given, the standard input is read with each line being taken as the
name of an SCCS file to be processed.

Keyletter arguments apply independently to each named file.

-rSID
   Uniquely identifies which delta is no longer intended. (This
   would have been specified by get as the “new delta”). The use of
   this keyletter is necessary only if two or more outstanding gets for
   editing on the same SCCS file were done by the same person
   (login name). A diagnostic results if the specified SID is ambigu-
   ous, or if it is necessary and omitted on the command line.

-s
   Suppresses the printout, on the standard output, of the intended
delta’s SID.

-n
   Causes the retention of the gotten file which would normally be
   removed from the current directory.

SEE ALSO
delta(1), get(1), help(1), sact(1).

DIAGNOSTICS
Use help(1) for explanations.
NAME
unifdef – remove ifdef’ed lines

SYNOPSIS
unifdef [ -t -l -c -Dsym -Usym -idsym -iusym ] ... [ file ]

DESCRIPTION
unifdef is useful for removing ifdef’ed lines from a file while otherwise leaving the file alone. unifdef is like a stripped-down C preprocessor: it is smart enough to deal with the nested ifdefs, comments, single and double quotes of C syntax so that it can do its job, but it doesn’t do any including or interpretation of macros. Neither does it strip out comments, though it recognizes and ignores them. You specify which symbols you want defined –Dsym or undefined –Usym and the lines inside those ifdefs will be copied to the output or removed as appropriate. The ifdef, ifndef, else, and endif lines associated with sym will also be removed. Ifdefs involving symbols you don’t specify are untouched and copied out along with their associated ifdef, else, and endif lines. If an ifdef X occurs nested inside another ifdef X, then the inside ifdef is treated as if it were an unrecognized symbol. If the same symbol appears in more than one argument, only the first occurrence is significant.

The –l option causes unifdef to replace removed lines with blank lines instead of deleting them.

If you use ifdefs to delimit non-C lines, such as comments or code which is under construction, then you must tell unifdef which symbols are used for that purpose so that it won’t try to parse for quotes and comments in those ifdef’ed lines. You specify that you want the lines inside certain ifdefs to be ignored but copied out with –idsym and –iusym similar to –Dsym and –Usym above.

If you want to use unifdef for plain text (not C code), use the –t option. This makes unifdef refrain from attempting to recognize comments and single and double quotes. unifdef copies its output to stdout and will take its input from stdin if no file argument is given. If the –c argument is specified, then the operation of unifdef is complemented, i.e. the lines that would have been removed or blanked are retained and vice versa.

SEE ALSO
diff(1)

DIAGNOSTICS
Premature EOF, inappropriate else or endif.
Exit status is 0 if output is exact copy of input, 1 if not, 2 if trouble.

BUGS
Does not know how to deal with cpp constructs such as

    #if defined(X) || defined(Y)

AUTHOR
Dave Yost
Directory and File Management Utilities

UNIQ(1) UNIQ(1)

NAME
uniq - report repeated lines in a file

SYNOPSIS
uniq [-uA] [ +n ] [ -n ] [ input [ output ]]

DESCRIPTION
uniq reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. Input and output should always be different. Note that repeated lines must be adjacent in order to be found; see sort(1). If the -u flag is used, just the lines that are not repeated in the original file are output. The -d option specifies that one copy of just the repeated lines is to be written. The normal mode output is the union of the -u and -d mode outputs.

The -c option supersedes -u and -d and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

The n arguments specify skipping an initial portion of each line in the comparison:

- n  The first n fields together with any blanks before each are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.

+ n  The first n characters are ignored. Fields are skipped before characters.

SEE ALSO
comm(1), sort(1).
NAME

units – conversion program

SYNOPSIS

units

DESCRIPTION

units converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively in this fashion:

You have: inch
You want: cm
    * 2.54000e+00
    / 3.937008e–01

A quantity is specified as a multiplicative combination of units optionally preceded by a numeric multiplier. Powers are indicated by suffixed positive integers, division by the usual sign:

You have: 15 lbs force/in2
You want: atm
    * 1.020689e+00
    / 9.797299e–01

units only does multiplicative scale changes; thus it can convert Kelvin to Rankine, but not Celsius to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, together with a generous leavening of exotica and a few constants of nature including:

pi ratio of circumference to diameter,
c speed of light,
e charge on an electron,
g acceleration of gravity,
force same as g,
mole Avogadro's number,
water pressure head per unit height of water,
au astronomical unit.

Pound is not recognized as a unit of mass; lb is. Compound names are run together, (e.g., lightyear). British units that differ from their U.S. counterparts are prefixed thus: brgallon. For a complete list of units, type:

    cat /usr/lib/unittab

FILES

/usr/lib/unittab
NAME

uptime – show how long system has been up

SYNOPSIS

uptime

DESCRIPTION

uptime prints the current time, the length of time the system has been up, and the average number of jobs in the run queue over the last 1, 5 and 15 minutes. It is, essentially, the first line of a w(1) command.

FILES

/unix system name list

SEE ALSO

w(1)
NAME
users – compact list of users who are on the system

SYNOPSIS
users

DESCRIPTION
users lists the login names of the users currently on the system in a compact, one-line format.

FILES
/etc/utmp

SEE ALSO
who(1)
NAME
uucheck – check the uucp directories and permissions file

SYNOPSIS
/usr/lib/uucp/uucheck [ -v ] [ -x debug_level ]

DESCRIPTION
uucheck checks for the presence of the uucp system required files and directories. Within the uucp makefile, it is executed before the installation takes place. It also checks for some obvious errors in the Permissions file (/usr/lib/uucp/Permissions). When executed with the -v option, it gives a detailed explanation of how the uucp programs will interpret the Permissions file. The -x option is used for debugging. debug-option is a single digit in the range 1-9; the higher the value, the greater the detail.

Note that uucheck can only be used by the super-user or uucp.

FILES
/usr/lib/uucp/Systems
/usr/lib/uucp/Permissions
/usr/lib/uucp/Devices
/usr/lib/uucp/Maxuuscheds
/usr/lib/uucp/Maxuuxqts
/usr/spool/uucp/*
/usr/spool/locks/LCK*
/usr/spool/uucppublic/*

SEE ALSO
uucico(1M), uucp(1C), uuscheds(1M), uustat(1C), uux(1C).

BUGS
The program does not check file/directory modes or some errors in the Permissions file such as duplicate login or machine name.
NAME

uucico – file transport program for the uucp system

SYNOPSIS

/usr/lib/uucp/uucico [-r role_number] [-x debug_level] [-i interface] [-d spool_directory] -s system_name

DESCRIPTION

uucico is the file transport program for uucp work file transfers. Role numbers for the
-r are the digit 1 for master mode or 0 for slave mode (default). The -r option should
be specified as the digit 1 for master mode when uucico is started by a program or
cron. Uux and uucp both queue jobs that will be transferred by uucico. It is normally
started by the scheduler, uusched, but can be started manually; this is done for
debugging. For example, the shell Uutry starts uucico with debugging turned on. A
single digit must be used for the -x option with higher numbers for more debugging.
The -i option defines the interface used with uucico. This interface only affects slave
mode. Known interfaces are UNIX (default), TLI (basic Transport Layer Interface),
and TLIS (Transport Layer Interface with STREAMS modules, read/write).

FILES

/usr/lib/uucp/Systems
/usr/lib/uucp/Permissions
/usr/lib/uucp/Devices
/usr/lib/uucp/Devconfig
/usr/lib/uucp/Sysfiles
/usr/lib/uucp/Maxuuxqts
/usr/lib/uucp/Maxusscheds
/usr/spool/uucp/*
/usr/spool/locks/LCK*
/usr/spool/uucppublic/*

SEE ALSO

cron(1M), uucp(1C), uussched(1M), uustat(1C), uutry(1M), uux(1C).
NAME

uucleanup – uucp spool directory clean-up

SYNOPSIS

/usr/lib/uucp/uucleanup [ -Ctime ] [ -Wtime ] [ -Xtime ] [ -mstring ]
[ -otime ] [ -ssystem ]

DESCRIPTION

uucleanup will scan the spool directories for old files and take appropriate action to remove them in a useful way:

Inform the requestor of send/receive requests for systems that cannot be reached.

Return mail, which cannot be delivered, to the sender.

Delete or execute rnews for rnews type files (depending on where the news originated—locally or remotely).

Remove all other files.

In addition, there is provision to warn users of requests that have been waiting for a given number of days (default 1). Note that uucleanup will process as if all option times were specified to the default values unless time is specifically set.

The following options are available.

-Ctime Any C. files greater or equal to time days old will be removed with appropriate information to the requestor. (default 7 days)

-Dtime Any D. files greater or equal to time days old will be removed. An attempt will be made to deliver mail messages and execute rnews when appropriate. (default 7 days)

-Wtime Any C. files equal to time days old will cause a mail message to be sent to the requestor warning about the delay in contacting the remote. The message includes the JOBID, and in the case of mail, the mail message. The administrator may include a message line telling whom to call to check the problem (-m option). (default 1 day)

-Xtime Any X. files greater or equal to time days old will be removed. The D. files are probably not present (if they were, the X. could get executed). But if there are D. files, they will be taken care of by D. processing. (default 2 days)

-mstring This line will be included in the warning message generated by the -W option.

-otime Other files whose age is more than time days will be deleted. (default 2 days) The default line is “See your local administrator to locate the problem”.

-ssystem Execute for system spool directory only.

-xdebug_level

The -x debug level is a single digit between 0 and 9; higher numbers give more detailed debugging information. (If uucleanup was compiled with -DSMALL, no debugging output will be available.)

This program is typically started by the shell uudemon.cleanup, which should be started by cron(1M).

FILES
/usr/lib/uucp directory with commands used by uucleanup internally
/usr/spool/uucp spool directory

SEE ALSO
cron(1M), uucp(1C), uux(1C).
NAME
uucp, uulog, uname – UNIX-to-UNIX system copy

SYNOPSIS
uucp [options] source-files destination-file
uulog [options] -sysystem
uulog [options] system
uulog [options] -fsystem
uname [-l] [-c]

DESCRIPTION
uucp

uucp copies files named by the source-file arguments to the destination-file argument. A file name may be a path name on your machine, or may have the form:

    system-name!path-name

where system-name is taken from a list of system names that uucp knows about. The system-name may also be a list of names such as

    system-name!system-name!system-name!...

in which case an attempt is made to send the file via the specified route, to the destination. See WARNINGS and BUGS below for restrictions. Care should be taken to ensure that intermediate nodes in the route are willing to forward information (see WARNINGS below for restrictions).

The shell metacharacters ?, * and [...] appearing in path-name will be expanded on the appropriate system.

Path names may be one of:

1. a full path name;
2. a path name preceded by ~user where user is a login name on the specified system and is replaced by that user’s login directory;
3. a path name preceded by ~/destination where destination is appended to /usr/spool/uucppublic; (NOTE: This destination will be treated as a file name unless more than one file is being transfered by this request or the destination is already a directory. To ensure that it is a directory, follow the destination with a '/'. For example ~/dan/ as the destination will make the directory /usr/spool/uucppublic/dan if it does not exist and put the requested file(s) in that directory).
4. anything else is prefixed by the current directory.

If the result is an erroneous path name for the remote system the copy will fail. If the destination-file is a directory, the last part of the source-file name is used.

uucp preserves execute permissions across the transmission and gives 0666 read and write permissions (see chmod(2)).

The following options are interpreted by uucp:

- Do not copy local file to the spool directory for transfer to the remote machine (default).
- C Force the copy of local files to the spool directory for transfer.
- d Make all necessary directories for the file copy (default).
- f Do not make intermediate directories for the file copy.
Basic Networking Utilities

UUCP (1C)

-ggrade Grade is a single letter/number; lower ascii sequence characters will cause the job to be transmitted earlier during a particular conversation.

-j Output the job identification ASCII string on the standard output. This job identification can be used by uustat to obtain the status or terminate a job.

-m Send mail to the requester when the copy is completed.

-nuser Notify user on the remote system that a file was sent.

-r Do not start the file transfer, just queue the job.

-sfile Report status of the transfer to file. Note that the file must be a full path name.

-xdebug_level Produce debugging output on standard output. The debug_level is a number between 0 and 9; higher numbers give more detailed information. (Debugging will not be available if uucp was compiled with -DSMALL.)

uulog

uulog queries a log file of uucp or uuxqt transactions in a file /usr/spool/uucp/Log/ucico/system, or /usr/spool/uucp/Log/uuxqt/system.

The options cause uulog to print logging information:

-ssys Print information about file transfer work involving system sys.

-ffsystem Does a “tail -f” of the file transfer log for system. (You must hit BREAK to exit this function.) Other options used in conjunction with the above:

-x Look in the uuxqt log file for the given system.

-number Indicates that a “tail” command of number lines should be executed.

uuname

uuname lists the names of systems known to uucp. The -e option returns the names of systems known to cu. (The two lists are the same, unless your machine is using different Systems files for cu and uucp. See the Sysfiles file.) The -I option returns the local system name.

FILES

/usr/spool/uucp spool directories
/usr/spool/uucppublic/* public directory for receiving and sending (/usr/spool/uucppublic)
/usr/lib/uucp/* other data and program files

SEE ALSO

chmod(2), mail(1), uustat(1C), uux(1C), uuxqt(1M).

WARNINGS

The domain of remotely accessible files can (and for obvious security reasons, usually should) be severely restricted. You will very likely not be able to fetch files by path name; ask a responsible person on the remote system to send them to you. For the same reasons you will probably not be able to send files to arbitrary path names. As distributed, the remotely accessible files are those whose names begin /usr/spool/uucppublic (equivalent to -/).

All files received by uucp will be owned by uucp.

The -m option will only work sending files or receiving a single file. Receiving multiple files specified by special shell characters ? * [...] will not activate the -m option.
The forwarding of files through other systems may not be compatible with the previous version of `uucp`. If forwarding is used, all systems in the route must have the same version of `uucp`.

**BUGS**

Protected files and files that are in protected directories that are owned by the requestor can be sent by `uucp`. However, if the requestor is root, and the directory is not searchable by "other" or the file is not readable by "other", the request will fail.
NAME

uuencode, uudecode – encode/decode a binary file for transmission via mail

SYNOPSIS

uuencode [ source ] remotedest | mail sys1!sys2!..!decode
uudecode [ file ]

DESCRIPTION

uuencode and uudecode are used to send a binary file via uucp (or other) mail. This combination can be used over indirect mail links even when uuser(1C) is not available.

uuencode takes the named source file (default standard input) and produces an encoded version on the standard output. The encoding uses only printing ASCII characters, and includes the mode of the file and the remotedest for recreation on the remote system.

uudecode reads an encoded file, strips off any leading and trailing lines added by mailers, and recreates the original file with the specified mode and name.

The intent is that all mail to the user “decode” should be filtered through the uudecode program. This way the file is created automatically without human intervention. This is possible on the uucp network by either using sendmail or by making rmail be a link to Mail instead of mail. In each case, an alias must be created in a master file to get the automatic invocation of uudecode.

If these facilities are not available, the file can be sent to a user on the remote machine who can uudecode it manually.

The encode file has an ordinary text form and can be edited by any text editor to change the mode or remote name.

SEE ALSO

atob(n), uuser(1C), uucp(1C), uux(1C), mail(1), uudecode(4)

BUGS

The file is expanded by 35% (3 bytes become 4 plus control information) causing it to take longer to transmit.

The user on the remote system who is invoking uudecode (often uucp) must have write permission on the specified file.
NAME

uugetty – set terminal type, modes, speed, and line discipline

SYNOPSIS

/usr/lib/uucp/getty [-h] [-t timeout] [-r] line
    [ speed [ type [ linedisc ] ] ]
/usr/lib/uucp/getty -c file

DESCRIPTION

uugetty is identical to getty(1M) but changes have been made to support using the
line for uucico, cu, and ct; that is, the line can be used in both directions. The uugetty
will allow users to login, but if the line is free, uucico, cu, or ct can use it for dialing
out. The implementation depends on the fact that uucico, cu, and ct create lock files
when devices are used. When the "open()" returns (or the first character is read when
-r option is used), the status of the lock file indicates whether the line is being used
by uucico, cu, ct, or someone trying to login. Note that in the -r case, several
<carriage-return> characters may be required before the login message is output.
The human users will be able to handle this slight inconvenience. Uucico trying to
login will have to be told by using the following login script:

"in in in..."

where the... is whatever would normally be used for the login sequence.

An entry for an intelligent modem or direct line that has a uugetty on each end must
use the -r option. (This causes uugetty to wait to read a character before it puts out
the login message, thus preventing two ugettys from looping.) If there is a uugetty
on one end of a direct line, there must be a uugetty on the other end as well. Here is
an /etc/inittab entry using uugetty on an intelligent modem or direct line:

30:2:respawn:/usr/lib/uucp/uugetty -r -t 60 tty12 1200

FILES

/etc/gettydefs
/etc/issue

SEE ALSO

c(1C), cu(1C), getty(1M), gettydefs(4), init(1M), inittab(4), ioctl(2), login(1), tty(7),
uucico(1M).

BUGS

Ct will not work when uugetty is used with an intelligent modem such as penrill or
ventel.
NAME

uusched – the scheduler for the uucp file transport program

SYNOPSIS

/usr/lib/uucp/uusched [-x debug_level] [-u debug_level]

DESCRIPTION

uusched is the uucp file transport scheduler. It is usually started by the daemon
uudemon.hour that is started by cron(1M) from an entry in /usr/spool/cron/crontab:

39 * * * * /bin/su uucp -c"/usr/lib/uucp/uudemon.hour > /dev/null"

The two options are for debugging purposes only; -x debug_level will output debug-
ging messages from uusched and -u debug_level will be passed as -x debug_level to
uucico. The debug_level is a number between 0 and 9; higher numbers give more
detailed information.

FILES

/usr/lib/uucp/Systems
/usr/lib/uucp/Permissions
/usr/lib/uucp/Devices
/usr/spool/uucp/*
/usr/spool/locks/LCK*
/usr/spool/uucppublic/*

SEE ALSO

cron(1M), uucico(1M), uucp(1C), uustat(1C), uux(1C).
NAME

uustat – uucp status inquiry and job control

SYNOPSIS

uustat [-a]
uustat [-m]
uustat [-p]
uustat [-q]
uustat [-kjobid ]
uustat [-rjobid ]
uustat [-ssystem ] [-u user ]

DESCRIPTION

uustat will display the status of, or cancel, previously specified uucp commands, or provide general status on uucp connections to other systems. Only one of the following options can be specified with uustat per command execution:

- a
  Output all jobs in queue.
- m
  Report the status of accessibility of all machines.
- p
  Execute a "ps -flp" for all the process-ids that are in the lock files.
- q
  List the jobs queued for each machine. If a status file exists for the
  machine, its date, time and status information are reported. In addition, if
  a number appears in () next to the number of C or X files, it is the age in
  days of the oldest C./X. file for that system. The Retry field represents
  the number of hours until the next possible call. The Count is the number
  of failure attempts. NOTE: for systems with a moderate number of out-
  standing jobs, this could take 30 seconds or more of real-time to execute.

  As an example of the output produced by the -q option:

  eagle 3C 04/07-11:07 NO DEVICES AVAILABLE
  mh3bs3 2C 07/07-10:42 SUCCESSFUL

  This output tells how many command files are waiting for each system.
  Each command file may have zero or more files to be sent (zero means to
  call the system and see if work is to be done). The date and time refer to
  the previous interaction with the system followed by the status of the
  interaction.

- kjobid
  Kill the uucp request whose job identification is jobid. The killed uucp
  request must belong to the person issuing the uustat command unless one
  is the super-user.

- rjobid
  Rejuvenate jobid. The files associated with jobid are touched so that their
  modification time is set to the current time. This prevents the cleanup
demon from deleting the job until the jobs modification time reaches the
  limit imposed by the deamon.

Either or both of the following options can be specified with uustat:

- ssys
  Report the status of all uucp requests for remote system sys.
- u user
  Report the status of all uucp requests issued by user.

Output for both the -s and -u options has the following format:

  eaglen0000 4/07-11:01:03 (POLL)
eagleN1bd7 4/07-11:07 S eagle dan 522 /usr/dan/A
eagleC1bd8 4/07-11:07 S eagle dan 59 D.3b2al2ce4924
  4/07-11:07 S eagle dan rmail mike

With the above two options, the first field is the jobid of the job. This is followed by
the date/time. The next field is either an ‘S’ or ‘R’ depending on whether the job is to send or request a file. This is followed by the user-id of the user who queued the job. The next field contains the size of the file, or in the case of a remote execution (rmail – the command used for remote mail), the name of the command. When the size appears in this field, the file name is also given. This can either be the name given by the user or an internal name (e.g., D.3b2alce4924) that is created for data files associated with remote executions (rmail in this example).

When no options are given, uustat outputs the status of all uucp requests issued by the current user.

FILES

/usr/spool/uu*    spool directories

SEE ALSO

uu*cp(1C).
NAME

uuto, uupick – public UNIX-to-UNIX system file copy

SYNOPSIS

uuto [ options ] source-files destination
uupick [ -s system ]

DESCRIPTION

uuto sends source-files to destination. uuto uses the uucp(1C) facility to send files, while it allows the local system to control the file access. A source-file name is a path name on your machine. Destination has the form:

    system!user

where system is taken from a list of system names that uucp knows about (see username). User is the login name of someone on the specified system.

Two options are available:

    -p        Copy the source file into the spool directory before transmission.
    -m        Send mail to the sender when the copy is complete.

The files (or sub-trees if directories are specified) are sent to PUBDIR on system, where PUBDIR is a public directory defined in the uucp source. By default this directory is /usr/spool/uucppublic. Specifically the files are sent to

    PUBDIR/receive/user/mysystem/files.

The destined recipient is notified by mail(1) of the arrival of files.

Uupick accepts or rejects the files transmitted to the user. Specifically, uupick searches PUBDIR for files destined for the user. For each entry (file or directory) found, the following message is printed on the standard output:

    from system: [ file file-name ] [ dir dirname ]?

Uupick then reads a line from the standard input to determine the disposition of the file:

    <new-line> Go on to next entry.
    d        Delete the entry.
    m [ dir ] Move the entry to named directory dir. If dir is not specified as a complete path name (in which $HOME is legitimate), a destination relative to the current directory is assumed. If no destination is given, the default is the current directory.
    a [ dir ] Same as m except moving all the files sent from system.
    p        Print the content of the file.
    q        Stop.
    EOT (control-d) Same as q.
    !command Escape to the shell to do command.
    *        Print a command summary.

Uupick invoked with the -ssystem option will only search the PUBDIR for files sent from system.

FILES

PUBDIR /usr/spool/uucppublic public directory
SEE ALSO

mail(1), uucleanup(1M), uucp(1C), uustat(1C), uux(1C).

WARNINGS

In order to send files that begin with a dot (e.g., .profile) the files must be qualified with a dot. For example: .profile, .prof*, .profil? are correct; whereas *prof*, ?profile are incorrect.
NAME
Uutry – try to contact remote system with debugging on

SYNOPSIS
/usr/lib/uucp/Uutry [-x debug_level] [-r] system_name

DESCRIPTION
Uutry is a shell that is used to invoke uucico to call a remote site. Debugging is turned on (default is level 5); -x will override that value. The -r overrides the retry time in /usr/spool/uucp/status. The debugging output is put in file /tmp/system_name. A tail -f of the output is executed. A <DELETE> or <BREAK> will give control back to the terminal while the uucico continues to run, putting its output in /tmp/system_name.

FILES
/usr/lib/uucp/Systems
/usr/lib/uucp/Permissions
/usr/lib/uucp/Devices
/usr/lib/uucp/maxuuxqts
/usr/lib/uucp/maxuuscheds
/usr/spool/uucp/*
/usr/spool/locks/LCK*
/usr/spool/uucppublic/*
/tmp/system_name

SEE ALSO
uucico(1M), uucp(1C), uux(1C).
NAME

uux - UNIX-to-UNIX system command execution

SYNOPSIS

uux [ options ] command-string

DESCRIPTION

uux will gather zero or more files from various systems, execute a command on a
specified system and then send standard output to a file on a specified system.

NOTE: For security reasons, most installations limit the list of commands executable
on behalf of an incoming request from uux, permitting only the receipt of mail (see
mail(1)). (Remote execution permissions are defined in /usr/lib/uucp/Permissions.)

The command-string is made up of one or more arguments that look like a shell com-
mand line, except that the command and file names may be prefixed by system-name.
A null system-name is interpreted as the local system.

File names may be one of

(1) a full path name;
(2) a path name preceded by ~xxx where xxx is a login name on the specified
system and is replaced by that user's login directory;
(3) anything else is prefixed by the current directory.

As an example, the command

uux "!diff usg! /usr/dan/file1 pwba! /a4/dan/file2 > !-/dan/file.diff"

will get the file1 and file2 files from the "usg" and "pwba" machines, execute a diff(1)
command and put the results in file.diff in the local PUBDIR/dan/ directory.

Any special shell characters such as <>;| should be quoted either by quoting the
entire command-string, or quoting the special characters as individual arguments.

uux will attempt to get all files to the execution system. For files that are output files,
the file name must be escaped using parentheses. For example, the command

uux alcut -f1 b! /usr/file \(c!/usr/file\)

gets /usr/file from system "b" and sends it to system "a", performs a cut command
on that file and sends the result of the cut command to system "c".

uux will notify you if the requested command on the remote system was disallowed.
This notification can be turned off by the --n option. The response comes by remote
mail from the remote machine.

The following options are interpreted by uux:

- The standard input to uux is made the standard input to the command-
  string.

-annotate Use name as the user identification replacing the initiator user-id.
  (Notification will be returned to the user.)

-b Return whatever standard input was provided to the uux command if the
  exit status is non-zero.

-c Do not copy local file to the spool directory for transfer to the remote
  machine (default).

-C Force the copy of local files to the spool directory for transfer.

-grade Grade is a single letter/number; lower ASCII sequence characters will
  cause the job to be transmitted earlier during a particular conversation.
Basic Networking Utilities

UUX(1C)

Output the jobid ASCII string on the standard output which is the job identification. This job identification can be used by uustat to obtain the status or terminate a job.

-n Do not notify the user if the command fails.

-p Same as -: The standard input to uux is made the standard input to the command-string.

-r Do not start the file transfer, just queue the job.


-xdebug_level Produce debugging output on the standard output. The debug_level is a number between 0 and 9; higher numbers give more detailed information.

-z Send success notification to the user.

FILES

/usr/lib/uucp/spool spool directories
/usr/lib/uucp/Permissions remote execution permissions
/usr/lib/uucp/* other data and programs

SEE ALSO

cut(1), mail(1), uucp(1C), uustat(1C).

WARNINGS

Only the first command of a shell pipeline may have a system-name! All other commands are executed on the system of the first command. The use of the shell metacharacter * will probably not do what you want it to do. The shell tokens << and >> are not implemented.

The execution of commands on remote systems takes place in an execution directory known to the uucp system. All files required for the execution will be put into this directory unless they already reside on that machine. Therefore, the simple file name (without path or machine reference) must be unique within the uux request. The following command will NOT work:

```
uux "aldiff b!//usr/dan/xyz c!//usr/dan/xyz > !xyz.diff"
```

but the command

```
uux "aldiff a!/usr/dan/xyz c!/usr/dan/xyz > !xyz.diff"
```

will work. (If diff is a permitted command.)

BUGS

Protected files and files that are in protected directories that are owned by the requestor can be sent in commands using uux. However, if the requestor is root, and the directory is not searchable by "other", the request will fail.
NAME

uuxqt - execute remote command requests

SYNOPSIS

/usr/lib/uucp/uuxqt [ -s system ] [ -x debug_level ]

DESCRIPTION

uuxqt is the program that executes remote job requests from remote systems generated by the use of the uux command. (Mail uses uux for remote mail requests). uuxqt searches the spool directories looking for X. files. For each X. file, uuxqt checks to see if all the required data files are available and accessible, and file commands are permitted for the requesting system. The Permissions file is used to validate file accessibility and command execution permission.

There are two environment variables that are set before the uuxqt command is executed:

    UU_MACHINE is the machine that sent the job (the previous one).
    UU_USER is the user that sent the job.

These can be used in writing commands that remote systems can execute to provide information, auditing, or restrictions.

The -x debug_level is a single digit between 0 and 9. Higher numbers give more detailed debugging information.

FILES

/usr/lib/uucp/Permissions
/usr/lib/uucp/Maxuuxqts
/usr/spool/uucp/*
/usr/spool/locks/LCK*

SEE ALSO

mail(1), uucico(1M), uucp(1C), uustat(1C), uux(1C).
NAME
vacation – return "I am on vacation" indication

SYNOPSIS
vacation -I
vacation user

DESCRIPTION
vacation returns a message to the sender of a message telling that you are on vacation. The intended use is in a .forward file. For example, your .forward file might have:

\eric,"I vacation eric"

which would send messages to you (assuming your login name was eric) and send a message back to the sender.

vacation expects a file .vacation.msg in your home directory containing a message to be sent back to each sender. It should be an entire message (including headers). For example, it might say:

From: eric@ucbmonet.Berkeley.EDU (Eric Allman)
Subject: I am on vacation
Delivered-By-The-Graces-Of: the Vacation program

I am on vacation until July 22. If you have something urgent, please contact Joe Kalash <kalash@ucbingres.Berkeley.EDU>.

--eric

This message will only be sent once a week to each unique sender. The people who have sent you messages are kept in the files .vacation.pag and .vacation.dir in your home directory. The -I option initializes these files, and should be executed before you modify your .forward file.

If the -I flag is not specified, vacation reads the first line from the standard input for a UNIX-style "From" line to determine the sender. If this is not present, a nasty diagnostic is produced. sendmail(8) includes the "From" line automatically.

No message is sent if the initial "From" line includes the string "-REQUEST@" or if a "Precedence: bulk" or "Precedence: junk" line is included in the header.

SEE ALSO
sendmail(8)
NAME

val – validate SCCS file

SYNOPSIS

val
val [-s] [-rSID] [-mname] [-ytype] files

DESCRIPTION

val determines if the specified file is an SCCS file meeting the characteristics specified by the optional argument list. Arguments to val may appear in any order. The arguments consist of keyletter arguments, which begin with a –, and named files.

val has a special argument, –, which causes reading of the standard input until an end-of-file condition is detected. Each line read is independently processed as if it were a command line argument list.

val generates diagnostic messages on the standard output for each command line and file processed, and also returns a single 8-bit code upon exit as described below.

The keyletter arguments are defined as follows. The effects of any keyletter argument apply independently to each named file on the command line.

–s	The presence of this argument silences the diagnostic message normally generated on the standard output for any error that is detected while processing each named file on a given command line.

–rSID	The argument value SID (SCCS IDentification String) is an SCCS delta number. A check is made to determine if the SID is ambiguous (e. g., r1 is ambiguous because it physically does not exist but implies 1.1, 1.2, etc., which may exist) or invalid (e. g., r1.0 or r1.1.0 are invalid because neither case can exist as a valid delta number). If the SID is valid and not ambiguous, a check is made to determine if it actually exists.

–mname	The argument value name is compared with the s-1SCCS %M% keyword in file.

–ytype	The argument value type is compared with the SCCS %Y% keyword in file.

The 8-bit code returned by val is a disjunction of the possible errors, i. e., can be interpreted as a bit string where (moving from left to right) set bits are interpreted as follows:

bit 0 = missing file argument;
bit 1 = unknown or duplicate keyletter argument;
bit 2 = corrupted SCCS file;
bit 3 = cannot open file or file not SCCS;
bit 4 = SID is invalid or ambiguous;
bit 5 = SID does not exist;
bit 6 = %Y%, –y mismatch;
bit 7 = %M%, –m mismatch;

Note that val can process two or more files on a given command line and in turn can process multiple command lines (when reading the standard input). In these cases an aggregate code is returned – a logical OR of the codes generated for each command line and file processed.

SEE ALSO

admin(1), delta(1), get(1), help(1), prs(1).

DIAGNOSTICS

Use help(1) for explanations.
BUGS

val can process up to 50 files on a single command line. Any number above 50 will produce a core dump.
NAME
vc - version control

SYNOPSIS
vc [-a] [-t] [-cchar] [-s] [keyword=value ... keyword=value]

DESCRIPTION
The vc command copies lines from the standard input to the standard output under control of its arguments and control statements encountered in the standard input. In the process of performing the copy operation, user declared keywords may be replaced by their string value when they appear in plain text and/or control statements.

The copying of lines from the standard input to the standard output is conditional, based on tests (in control statements) of keyword values specified in control statements or as vc command arguments.

A control statement is a single line beginning with a control character, except as modified by the -t keyletter (see below). The default control character is colon (:), except as modified by the -c keyletter (see below). Input lines beginning with a backslash (\) followed by a control character are not control lines and are copied to the standard output with the backslash removed. Lines beginning with a backslash followed by a non-control character are copied in their entirety.

A keyword is composed of 9 or less alphanumerics; the first must be alphabetic. A value is any ASCII string that can be created with ed(1); a numeric value is an unsigned string of digits. Keyword values may not contain blanks or tabs.

Replacement of keywords by values is done whenever a keyword surrounded by control characters is encountered on a version control statement. The -a keyletter (see below) forces replacement of keywords in all lines of text. An uninterpreted control character may be included in a value by preceding it with \. If a literal \ is desired, then it too must be preceded by \\.

Keyletter Arguments
-a Forces replacement of keywords surrounded by control characters with their assigned value in all text lines and not just in vc statements.
-t All characters from the beginning of a line up to and including the first tab character are ignored for the purpose of detecting a control statement. If one is found, all characters up to and including the tab are discarded.
-cchar Specifies a control character to be used in place of :.
-s Silences warning messages (not error) that are normally printed on the diagnostic output.

Version Control Statements
:dcl keyword[, ..., keyword]
Used to declare keywords. All keywords must be declared.

:asg keyword=value
Used to assign values to keywords. An asg statement overrides the assignment for the corresponding keyword on the vc command line and all previous asg's for that keyword. Keywords declared, but not assigned values have null values.

:if condition
::
:end
Used to skip lines of the standard input. If the condition is true all lines between the if statement and the matching end statement are copied to the standard output. If the condition is false, all intervening lines are discarded, including control statements. Note that intervening if statements and matching end statements are recognized solely for the purpose of maintaining the proper if-end matching.

The syntax of a condition is:

<cond> ::= [ "not" ] <or>
<or> ::= <and> | <and> "||" <or>
<and> ::= <exp> | <exp> "&&" <and>
<exp> ::= "(" <or> ")" | <value> <op> <value>
<op> ::= "=" | "!=" | "<" | ">">
<value> ::= <arbitrary ASCII string> | <numeric string>

The available operators and their meanings are:

= equal
!= not equal
& and
| or
> greater than
< less than
() used for logical groupings
not may only occur immediately after the if, and
when present, inverts the value of the entire condition

The > and < operate only on unsigned integer values (e.g., : 012 > 12 is false). All other operators take strings as arguments (e.g., : 012 != 12 is true). The precedence of the operators (from highest to lowest) is:

= != < & |

Parentheses may be used to alter the order of precedence. Values must be separated from operators or parentheses by at least one blank or tab.

::text

Used for keyword replacement on lines that are copied to the standard output. The two leading control characters are removed, and keywords surrounded by control characters in text are replaced by their value before the line is copied to the output file. This action is independent of the -a keyletter.

:on
:off

Turn on or off keyword replacement on all lines.

:ctl char

Change the control character to char.

:msg message

Prints the given message on the diagnostic output.

:err message

Prints the given message followed by:

ERROR: err statement on line ... (915)
on the diagnostic output. `vc` halts execution, and returns an exit code of 1.

**SEE ALSO**
ed(1), help(1).

**DIAGNOSTICS**
Use `help(1)` for explanations.

**EXIT CODES**
0 – normal
1 – any error
NAME
vers – obtain version information

SYNOPSIS
vers [-iV] object-files

DESCRIPTION
The vers command prints out the version number of the compiler or assembler used to create the .o files, or the loader used to create a.out files.
If the object file is an archive, the file is treated as a set of individual object files.
The -i option prints the contents of the ident section. The -V option prints out the version number of the vers command.

SEE ALSO
cpp(1), a.out(4)
NAME

vgrind – grind nice listings of programs

SYNOPSIS

vgrind [ -f ] [ - ] [ -t ] [ -n ] [ -x ] [ -W ] [ -sn ] [ -h header ] [ -d file ] [ -language ]
name ...

DESCRIPTION

Vgrind formats the program sources which are arguments in a nice style using troff(1)
Comments are placed in italics, keywords in bold face, and the name of the current
function is listed down the margin of each page as it is encountered.

Vgrind runs in two basic modes, filter mode or regular mode. In filter mode vgrind
acts as a filter in a manner similar to tbl(1). The standard input is passed directly to
the standard output except for lines bracketed by the troff-like macros:

.vs - starts processing
.vE - ends processing

These lines are formatted as described above. The output from this filter can be
passed to troff for output. There need be no particular ordering with eqn(1) or tbl(1).
In regular mode vgrind accepts input files, processes them, and passes them to troff(1)
for output.

In both modes vgrind passes any lines beginning with a decimal point without
conversion.

The options are:

-f  forces filter mode
-  forces input to be taken from standard input (default if -f is specified )
-t  similar to the same option in troff causing formatted text to go to the standard output
-n  forces no keyword bolding
-x  outputs the index file in a “pretty” format. The index file itself is produced
whenever vgrind is run with a file called index in the current directory. The
index of function definitions can then be run off by giving vgrind the -x option
and the file index as argument.
-W  forces output to the (wide) Versatec printer rather than the (narrow) Varian
-s  specifies a point size to use on output (exactly the same as the argument of a
.ps)
-h  specifies a particular header to put on every output page (default is the file
name)
-d  specifies an alternate language definitions file (default is /usr/lib/vgrindefs)
-l  specifies the language to use. Currently known are PASCAL (-lp), MODEL
(-Im),C (-ic or the default), CSH (-Ish), SHELL (-Ish), RATFOR (-Ir),
MODULA2 (-Imod2), YACC (-Iyacc), ISP (-Iisp), and ICON (-II).

FILES

index  file where source for index is created
/usr/lib/tmac/tmac.vgrind  macro package
/usr/lib/vfontedpr  preprocessor
/usr/lib/vgrindefs  language descriptions
User Commands

VGRIND(1)

AUTHOR

Dave Presotto & William Joy

SEE ALSO

vlp(1), vtroff(1), vgrindefs(5)

BUGS

Vfonedpr assumes that a certain programming style is followed:
For C – function names can be preceded on a line only by spaces, tabs, or an asterisk. The parenthesized arguments must also be on the same line.
For PASCAL – function names need to appear on the same line as the keywords function or procedure.
For MODEL – function names need to appear on the same line as the keywords is beginproc.
If these conventions are not followed, the indexing and marginal function name comment mechanisms will fail.

More generally, arbitrary formatting styles for programs mostly look bad. The use of spaces to align source code fails miserably; if you plan to vgrind your program you should use tabs. This is somewhat inevitable since the font used by vgrind is variable width.
The mechanism of ctags in recognizing functions should be used here.
Filter mode does not work in documents using the –me or –ms macros. (So what use is it anyway?)
NAME

vi - screen-oriented (visual) display editor based on ex

SYNOPSIS

vi [ -t tag ] [ -r file ] [ -wn ] [ -R ] [ -x ] [ +command ] name ...
view [ -t tag ] [ -r file ] [ -wn ] [ -R ] [ -x ] [ +command ] name
vedit [ -t tag ] [ -r file ] [ -wn ] [ -R ] [ -x ] [ +command ] name

DESCRIPTION

vi (visual) is a display-oriented text editor based on an underlying line editor ex(1). It is possible to use the command mode of ex from within vi and vice-versa.

When using vi, changes you make to the file are reflected in what you see on your terminal screen. The position of the cursor on the screen indicates the position within the file.

INVOCATION

The following invocation options are interpreted by vi:

-t tag           Edit the file containing the tag and position the editor at its definition.
-r file          Recover file after an editor or system crash. If file is not specified a list of all saved files will be printed.
-wn              Set the default window size to n. This is useful when using the editor over a slow speed line.
-R               Read only mode; the readonly flag is set, preventing accidental overwriting of the file.
+command         The specified ex command is interpreted before editing begins.
-x                Encryption option; when this option is used, the file will be encrypted as it is being written and will require an encryption key to be read (see crypt(1)). Also, see the WARNING section at the end of this manual page.

The name argument indicates files to be edited.

The view invocation is the same as vi except that the readonly flag is set.

The vedit invocation is intended for beginners. The report flag is set to 1, and the showmode and novice flags are set. These defaults make it easier to get started learning the editor.

VI MODES

Command       Normal and initial mode. Other modes return to command mode upon completion. ESC (escape) is used to cancel a partial command.
Input          Entered by the following options a i A I o O c C s S R. Arbitrary text may then be entered. Input mode is normally terminated with ESC character, or abnormally with interrupt.
Last line      Reading input for : l ? or !; terminate with CR to execute, interrupt to cancel.
COMMAND SUMMARY

Sample commands

← ↓ ↑ →  arrow keys move the cursor
h j k l  same as arrow keys
itextESC  insert text abc
cwnewESC  change word to new
easESC  pluralize word
x  delete a character
dw  delete a word
dd  delete a line
3dd  ... 3 lines
u  undo previous change
ZZ  exit vi, saving changes
:q!CR  quit, discarding changes
/ textCR  search for text
^U ^D  scroll up or down
:ex cmdCR  any ex or ed command

Counts before vi commands

Numbers may be typed as a prefix to some commands. They are interpreted in one of these ways.
line/column number  z G l
scroll amount  ^D ^U
repeat effect  most of the rest

Interrupting, canceling

ESC  end insert or incomplete cmd
DEL  (delete or rubout) interrupts
^L  reprint screen if DEL scrambles it
^R  reprint screen if ^L is → key

File manipulation

:wCR  write back changes
:qCR  quit
:q!CR  quit, discard changes
:ex nameCR  edit file name
:ex!CR  reedit, discard changes
:ex + nameCR  edit, starting at end
:ex +nCR  edit starting at line n
:ex #CR  edit alternate file
  synonym for :e #

:wn nameCR  write file name
:wnl nameCR  overwrite file name
:shCR  run shell, then return
:cmdCR  run cmd, then return
:ncr  edit next file in arglist
:argscr  specify new arglist
^G  show current file and line
:ta tagCR  to tag file entry tag
^J  :ta, following word is tag In general, any ex or ed command (such as substitute or global) may be typed, preceded by a colon and followed by a CR.

Positioning within file
<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>^F</td>
<td>forward screen</td>
</tr>
<tr>
<td>^B</td>
<td>backward screen</td>
</tr>
<tr>
<td>^D</td>
<td>scroll down half screen</td>
</tr>
<tr>
<td>^U</td>
<td>scroll up half screen</td>
</tr>
<tr>
<td>G</td>
<td>go to specified line (end default)</td>
</tr>
<tr>
<td>/pat</td>
<td>next line matching pat</td>
</tr>
<tr>
<td>?pat</td>
<td>prev line matching pat</td>
</tr>
<tr>
<td>n</td>
<td>repeat last / or ?</td>
</tr>
<tr>
<td>N</td>
<td>reverse last / or ?</td>
</tr>
<tr>
<td>/pat/+n</td>
<td>nth line after pat</td>
</tr>
<tr>
<td>?pat?−n</td>
<td>nth line before pat</td>
</tr>
<tr>
<td>]</td>
<td>next section/function</td>
</tr>
<tr>
<td>]</td>
<td>previous section/function</td>
</tr>
<tr>
<td>(</td>
<td>beginning of sentence</td>
</tr>
<tr>
<td>)</td>
<td>end of sentence</td>
</tr>
<tr>
<td>{</td>
<td>beginning of paragraph</td>
</tr>
<tr>
<td>}</td>
<td>end of paragraph</td>
</tr>
<tr>
<td>%</td>
<td>find matching () { or }</td>
</tr>
</tbody>
</table>

**Adjusting the screen**

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>^L</td>
<td>clear and redraw</td>
</tr>
<tr>
<td>^R</td>
<td>retype, eliminate @ lines</td>
</tr>
<tr>
<td>zCR</td>
<td>redraw, current at window top</td>
</tr>
<tr>
<td>z−CR</td>
<td>... at bottom</td>
</tr>
<tr>
<td>z.CR</td>
<td>... at center</td>
</tr>
<tr>
<td>/pat/z−CR</td>
<td>pat line at bottom</td>
</tr>
<tr>
<td>zn.CR</td>
<td>use n line window</td>
</tr>
<tr>
<td>^E</td>
<td>scroll window down 1 line</td>
</tr>
<tr>
<td>^Y</td>
<td>scroll window up 1 line</td>
</tr>
</tbody>
</table>

**Marking and returning**

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot; &quot;</td>
<td>move cursor to previous context</td>
</tr>
<tr>
<td>&quot;</td>
<td>... at first non-white in line</td>
</tr>
<tr>
<td>mx</td>
<td>mark current position with letter x</td>
</tr>
<tr>
<td>\x</td>
<td>move cursor to mark x</td>
</tr>
<tr>
<td>\x</td>
<td>... at first non-white in line</td>
</tr>
</tbody>
</table>

**Line positioning**

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>top line on screen</td>
</tr>
<tr>
<td>L</td>
<td>last line on screen</td>
</tr>
<tr>
<td>M</td>
<td>middle line on screen</td>
</tr>
<tr>
<td>+</td>
<td>next line, at first non-white</td>
</tr>
<tr>
<td>−</td>
<td>previous line, at first non-white</td>
</tr>
<tr>
<td>CR</td>
<td>return, same as +</td>
</tr>
<tr>
<td>↓ or j</td>
<td>next line, same column</td>
</tr>
<tr>
<td>↑ or k</td>
<td>previous line, same column</td>
</tr>
</tbody>
</table>

**Character positioning**

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>^</td>
<td>first non white</td>
</tr>
<tr>
<td>0</td>
<td>beginning of line</td>
</tr>
<tr>
<td>$</td>
<td>end of line</td>
</tr>
<tr>
<td>h or →</td>
<td>forward</td>
</tr>
<tr>
<td>l or ←</td>
<td>backwards</td>
</tr>
<tr>
<td>^H</td>
<td>same as ←</td>
</tr>
</tbody>
</table>
space  same as →
f  find x forward
Fx  f backward
tx  upto x forward
Tx  back upto x
;  repeat last f F t or T
,  inverse of ;
|  to specified column
%  find matching ( ) or }

Words, sentences, paragraphs
w  word forward
b  back word
e  end of word
)  to next sentence
}  to next paragraph
(  back sentence
{  back paragraph
W  blank delimited word
B  back W
E  to end of W

Corrections during insert
^H  erase last character
^W  erase last word
erase  your erase, same as ^H
kill  your kill, erase input this line
\  quotes ^H, your erase and kill
ESC  ends insertion, back to command
DEL  interrupt, terminates insert
^D  backtab over autoindent
↑^D  kill autoindent, save for next
0^D  ... but at margin next also
^V  quote non-printing character

Insert and replace
a  append after cursor
i  insert before cursor
A  append at end of line
I  insert before first non-blank
o  open line below
O  open above
rx  replace single char with x
RtextESC  replace characters

Operators
Operators are followed by a cursor motion, and affect all text that would have been moved over. For example, since w moves over a word, dw deletes the word that would be moved over. Double the operator, e.g., dd to affect whole lines.
d  delete
c  change
y  yank lines to buffer
<  left shift
>  right shift
! filter through command
= indent for LISP

**Miscellaneous Operations**

<table>
<thead>
<tr>
<th>Letter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>change rest of line (c$)</td>
</tr>
<tr>
<td>D</td>
<td>delete rest of line (d$)</td>
</tr>
<tr>
<td>s</td>
<td>substitute chars (cl)</td>
</tr>
<tr>
<td>S</td>
<td>substitute lines (cc)</td>
</tr>
<tr>
<td>J</td>
<td>join lines</td>
</tr>
<tr>
<td>x</td>
<td>delete characters (dl)</td>
</tr>
<tr>
<td>X</td>
<td>... before cursor (dh)</td>
</tr>
<tr>
<td>Y</td>
<td>yank lines (yy)</td>
</tr>
</tbody>
</table>

**Yank and Put**

Put inserts the text most recently deleted or yanked. However, if a buffer is named, the text in that buffer is put instead.

<table>
<thead>
<tr>
<th>Letter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>put back text after cursor</td>
</tr>
<tr>
<td>P</td>
<td>put before cursor</td>
</tr>
<tr>
<td>&quot;xp</td>
<td>put from buffer x</td>
</tr>
<tr>
<td>&quot;xy</td>
<td>yank to buffer x</td>
</tr>
<tr>
<td>&quot;xd</td>
<td>delete into buffer x</td>
</tr>
</tbody>
</table>

**Undo, Redo, Retrieve**

<table>
<thead>
<tr>
<th>Letter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>undo last change</td>
</tr>
<tr>
<td>U</td>
<td>restore current line</td>
</tr>
<tr>
<td>&quot;dp</td>
<td>retrieve d’th last delete</td>
</tr>
</tbody>
</table>

**AUTHOR**

*vi* and *ex* were developed by The University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

**FILES**

/usr/lib/terminfo/* compiled terminal description database
/usr/lib/.COREterm/* subset of compiled terminal description database, supplied on hard disk

**SEE ALSO**

ed(1), edit(1), ex(1).
*User's Guide.*
*Editing Guide.*

**WARNING**

The –x option is provided with the Security Administration Utilities, which is available only in the United States.

Tampering with entries in /usr/lib/.COREterm/* or /usr/lib/terminfo/* (for example, changing or removing an entry) can affect programs such as *vi*(1) that expect the entry to be present and correct. In particular, removing the “dumb” terminal may cause unexpected problems.

**BUGS**

Software tabs using ^T work only immediately after the *autoindent.*

Left and right shifts on intelligent terminals do not make use of insert and delete character operations in the terminal.
NAME

w – who is on and what they are doing

SYNOPSIS

w [-h] [-s] [ user ]

DESCRIPTION

w prints a summary of the current activity on the system, including what each user is doing. The heading line shows the current time of day, how long the system has been up, the number of users logged into the system, and the load averages. The load average numbers give the number of jobs in the run queue averaged over 1, 5 and 15 minutes.

The fields output are: the users login name, the name of the tty the user is on, the time of day the user logged on, the number of minutes since the user last typed anything, the CPU time used by all processes and their children on that terminal, the CPU time used by the currently active processes, the name and arguments of the current process.

The -h flag suppresses the heading. The -s flag asks for a short form of output. In the short form, the tty is abbreviated, the login time and cpu times are left off, as are the arguments to commands. -I gives the long output, which is the default.

If a user name is included, the output will be restricted to that user.

FILES

/etc/utmp
/dev/kmem

SEE ALSO

who(1), finger(1), ps(1)

AUTHOR

Mark Horton

BUGS

The notion of the “current process” is muddy. The current algorithm is “the highest numbered process on the terminal that is not ignoring interrupts, or, if there is none, the highest numbered process on the terminal”. This fails, for example, in critical sections of programs like the shell and editor, or when faulty programs running in the background fork and fail to ignore interrupts. (In cases where no process can be found, w prints “-“.)

The CPU time is only an estimate, in particular, if someone leaves a background process running after logging out, the person currently on that terminal is “charged” with the time.

Background processes are not shown, even though they account for much of the load on the system.

Sometimes processes, typically those in the background, are printed with null or garbaged arguments. In these cases, the name of the command is printed in parentheses.

w does not know about the new conventions for detection of background jobs. It will sometimes find a background job instead of the right one.
NAME

wait – await completion of process

SYNOPSIS

wait [ n ]

DESCRIPTION

Wait for your background process whose process id is n and report its termination status. If n is omitted, all your shell’s currently active background processes are waited for and the return code will be zero.

The shell itself executes wait, without creating a new process.

SEE ALSO

sh(1).

CAVEAT

If you get the error message cannot fork, too many processes, try using the wait(1) command to clean up your background processes. If this doesn’t help, the system process table is probably full or you have too many active foreground processes. (There is a limit to the number of process ids associated with your login, and to the number the system can keep track of.)

BUGS

Not all the processes of a 3- or more-stage pipeline are children of the shell, and thus cannot be waited for.

If n is not an active process id, all your shell’s currently active background processes are waited for and the return code will be zero.
NAME
wall – write to all users

SYNOPSIS
/etc/wall

DESCRIPTION
wall reads its standard input until an end-of-file. It then sends this message to all
currently logged-in users preceded by:

    Broadcast Message from …

It is used to warn all users, typically prior to shutting down the system.
The sender must be super-user to override any protections the users may have
invoked (see msg(1)).

FILES
/dev/tty*

SEE ALSO
msg(1), write(1).

DIAGNOSTICS
“Cannot send to …” when the open on a user’s tty file fails.
NAME

wc – word count

SYNOPSIS

wc [ –Iwc ] [ names ]

DESCRIPTION

wc counts lines, words, and characters in the named files, or in the standard input if no names appear. It also keeps a total count for all named files. A word is a maximal string of characters delimited by spaces, tabs, or new-lines.

The options l, w, and c may be used in any combination to specify that a subset of lines, words, and characters are to be reported. The default is –Iwc.

When names are specified on the command line, they will be printed along with the counts.
NAME

what – identify SCCS files

SYNOPSIS

what [-s] files

DESCRIPTION

what searches the given files for all occurrences of the pattern that get(1) substitutes for %Z% (this is @(#) at this printing) and prints out what follows until the first ~, >, new-line, \, or null character. For example, if the C program in file f.c contains

    char ident[] = "@(#)identification information";

and f.c is compiled to yield f.o and a.out, then the command

    what f.c f.o a.out

will print

    f.c:
    identification information

    f.o:
    identification information

    a.out:
    identification information

what is intended to be used in conjunction with the command get(1), which automatically inserts identifying information, but it can also be used where the information is inserted manually. Only one option exists:

    -s       Quit after finding the first occurrence of pattern in each file.

SEE ALSO

get(1), help(1).

DIAGNOSTICS

Exit status is 0 if any matches are found, otherwise 1. Use help(1) for explanations.

BUGS

It is possible that an unintended occurrence of the pattern @(#) could be found just by chance, but this causes no harm in nearly all cases.
NAME

whatis – describe what a command is

SYNOPSIS

whatis command ...

DESCRIPTION

whatis looks up a given command and gives the header line from the manual section. You can then run the man(1) command to get more information. If the line starts 'name(section) ...' you can do 'man section name' to get the documentation for it. Try 'whatis ed' and then you should do 'man 1 ed' to get the manual.

whatis is actually just the -f option to the man(1) command.

FILES

/usr/man/whatis Data base

SEE ALSO

man(1), catman(8)
WHEREIS(1)

NAME
whereis – locate the binary, source, and manual page files for a command

SYNOPSIS
whereis [-bmsu] [-BMS directory... -f] filename ...

DESCRIPTION
whereis locates source/binary and manuals sections for specified files. The supplied names are first stripped of leading pathname components and any (single) trailing extension of the form .ext, for example, .c. Prefixes of s. resulting from use of source code control are also dealt with. whereis then attempts to locate the desired program in a list of standard places:

/usr/bin
/usr/bin
/usr/5bin
/usr/hosts
/usr/include
/usr/local
/usr/etc
/usr/lib
/usr/share/man
/usr/src
/usr/ucb

OPTIONS
-b Search only for binaries.
-m Search only for manual sections.
-s Search only for sources.
-u Search for unusual entries. A file is said to be unusual if it does not have one entry of each requested type. Thus `whereis -m -u *` asks for those files in the current directory which have no documentation.
-B Change or otherwise limit the places where whereis searches for binaries.
-M Change or otherwise limit the places where whereis searches for manual sections.
-S Change or otherwise limit the places where whereis searches for sources.
-f Terminate the last directory list and signals the start of file names, and must be used when any of the -B, -M, or -S options are used.

EXAMPLE
Find all files in /usr/bin which are not documented in /usr/share/man/man1 with source in /usr/src/cmd:

eexample% cd /usr/ucb
eexample% whereis -u -M /usr/share/man/man1-S /usr/src/cmd -f *

FILES
/usr/src/*
/usr/[doc,man]/*
/etc,/usr/[lib,bin,ucb,old,new,local]

SEE ALSO
chdir(2)

504
Stardent 1500/3000
BUGS

Since *whereis* uses *chdir*(2) to run faster, pathnames given with the -M, -S, or -B must be full; that is, they must begin with a '/'.

NAME
which – locate a program file including aliases and paths

SYNOPSIS
which [ name ] ...

DESCRIPTION
which takes a list of names and looks for the files which would be executed had these names been given as commands. Each argument is expanded if it is aliased, and searched for along the user’s path. Both aliases and path are taken from the user’s .cshrc file.

FILES
~/.cshrc source of aliases and path values

DIAGNOSTICS
A diagnostic is given for names which are aliased to more than a single word, or if an executable file with the argument name was not found in the path.

BUGS
Must be executed by a csh, since only csh’s know about aliases.
NAME

who – who is on the system

SYNOPSIS

who [-uTIHqpdbrtras] [ file ]
who am i
who am I

DESCRIPTION

who can list the user’s name, terminal line, login time, elapsed time since activity occurred on the line, and the process-ID of the command interpreter (shell) for each current UNIX system user. It examines the /etc/utmp file at login time to obtain its information. If file is given, that file (which must be in utmp[4] format) is examined. Usually, file will be /etc/wtmp, which contains a history of all the logins since the file was last created.

who with the am i or am I option identifies the invoking user.

The general format for output is:

    name [state] line time [idle] [pid] [comment] [exit]

The name, line, and time information is produced by all options except –q; the state information is produced only by –T; the idle and pid information is produced only by –u and –l; and the comment and exit information is produced only by –a. The information produced for –p, –d, and –r is explained during the discussion of each option, below.

With options, who can list logins, logoffs, reboots, and changes to the system clock, as well as other processes spawned by the init process. These options are:

-u  This option lists only those users who are currently logged in. The name is the user’s login name. The line is the name of the line as found in the directory /dev. The time is the time that the user logged in. The idle column contains the number of hours and minutes since activity last occurred on that particular line. A dot (.) indicates that the terminal has seen activity in the last minute and is therefore “current”. If more than twenty-four hours have elapsed or the line has not been used since login time, the entry is marked old. This field is useful when trying to determine whether a person is working at the terminal or not. The pid is the process-ID of the user’s shell. The comment is the comment field associated with this line as found in /etc/inittab (see inittab[4]). This can contain information about where the terminal is located, the telephone number of the dataset, type of terminal if hard-wired, etc.

-T  This option is the same as the –s option, except that the state of the terminal line is printed. The state describes whether someone else can write to that terminal. A + appears if the terminal is writable by anyone; a – appears if it is not. root can write to all lines having a + or a – in the state field. If a bad line is encountered, a ? is printed.

-l  This option lists only those lines on which the system is waiting for someone to login. The name field is LOGIN in such cases. Other fields are the same as for user entries except that the state field does not exist.

-H  This option will print column headings above the regular output.

-q  This is a quick who, displaying only the names and the number of users currently logged on. When this option is used, all other options are ignored.
-p This option lists any other process which is currently active and has been previously spawned by init. The name field is the name of the program executed by init as found in /etc/inittab. The state, line, and idle fields have no meaning. The comment field shows the id field of the line from /etc/inittab that spawned this process. See inittab(4).

-d This option displays all processes that have expired and not been respawned by init. The exit field appears for dead processes and contains the termination and exit values (as returned by wait[2]), of the dead process. This can be useful in determining why a process terminated.

-b This option indicates the time and date of the last reboot.

-r This option indicates the current run-level of the init process. In addition, it produces the process termination status, process id, and process exit status (see utmp(4)) under the idle, pid, and comment headings, respectively.

-t This option indicates the last change to the system clock (via the date[1] command) by root. See su(1).

-a This option processes /etc/utmp or the named file with all options turned on.

-s This option is the default and lists only the name, line, and time fields.

Note to the super-user: after a shutdown to the single-user state, who returns a prompt; the reason is that since /etc/utmp is updated at login time and there is no login in single-user state, who cannot report accurately on this state. who am i, however, returns the correct information.

FILES

/etc/utmp
/etc/wtmp
/etc/inittab

SEE ALSO

date(1), init(1M), login(1), msg(1), su(1M), wait(2), inittab(4), utmp(4).
NAME

whoami – print effective current user id

SYNOPSIS

whoami

DESCRIPTION

*Whoami* prints who you are. It works even if you are `su`'d, while `who am i` does not since it uses `/etc/utmp`.

FILES

/etc/passwd Name data base

SEE ALSO

who (1)
WHODO(1M)

NAME

whodo – who is doing what

SYNOPSIS

/etc/whodo

DESCRIPTION

whodo produces formatted and dated output from information in the /etc/utmp and /etc/ps_data files.

The display is headed by the date, time and machine name. For each user logged in, device name, user-id and login time is shown, followed by a list of active processes associated with the user-id. The list includes the device name, process-id, cpu minutes and seconds used, and process name.

EXAMPLE

The command:

whodo

produces a display like this:

Tue Mar 12 15:48:03 1985
bailey

tty09  mcn   8:51
tty09  28158  0:29 sh

tty52  bdr   15:23
tty52  21688  0:05 sh
tty52  22788  0:01 whodo
tty52  22017  0:03 vi
tty52  22549  0:01 sh

xt162  lee  10:20
tty08  6748  0:01 layers
xt162  6731  0:01 sh
xt163  6761  0:05 sh
tty08  6536  0:05 sh

FILES

/etc/passwd
/etc/ps_data
/etc/utmp

SEE ALSO

ps(1), who(1).
NAME
write – write to another user

SYNOPSIS
write user [ line ]

DESCRIPTION
write copies lines from your terminal to that of another user. When first called, it sends the message:

Message from yourname (tty??) [ date ]

to the person you want to talk to. When it has successfully completed the connection, it also sends two bells to your own terminal to indicate that what you are typing is being sent.

The recipient of the message should write back at this point. Communication continues until an end of file is read from the terminal, an interrupt is sent, or the recipient has executed "mesg n". At that point write writes EOT on the other terminal and exits.

If you want to write to a user who is logged in more than once, the line argument may be used to indicate which line or terminal to send to (e.g., tty00); otherwise, the first writable instance of the user found in /etc/utmp is assumed and the following message posted:

user is logged on more than one place.
You are connected to "terminal".
Other locations are:
terminal

Permission to write may be denied or granted by use of the mesg(1) command. Writing to others is normally allowed by default. Certain commands, such as pr(1) disallow messages in order to prevent interference with their output. However, if the user has super-user permissions, messages can be forced onto a write-inhibited terminal.

If the character ! is found at the beginning of a line, write calls the shell to execute the rest of the line as a command.

The following protocol is suggested for using write: when you first write to another user, wait for them to write back before starting to send. Each person should end a message with a distinctive signal (i.e., (o) for "over") so that the other person knows when to reply. The signal (oo) (for "over and out") is suggested when conversation is to be terminated.

FILES
/etc/utmp to find user
/bin/sh to execute !

SEE ALSO
mail(1), mesg(1), pr(1), sh(1), who(1).

DIAGNOSTICS
"user is not logged on" if the person you are trying to write to is not logged on.
"Permission denied" if the person you are trying to write to denies that permission (with mesg).
"Warning: cannot respond, set mesg -y" if your terminal is set to mesg n and the recipient cannot respond to you.
"Can no longer write to user" if the recipient has denied permission (mesg n) after you had started writing.
NAME
xargs – construct argument list(s) and execute command

SYNOPSIS
xargs [ flags ] [ command [ initial-arguments ] ]

DESCRIPTION
xargs combines the fixed initial-arguments with arguments read from standard input to execute the specified command one or more times. The number of arguments read for each command invocation and the manner in which they are combined are determined by the flags specified.

command, which may be a shell file, is searched for, using one's $PATH. If command is omitted, /bin/echo is used.

Arguments read in from standard input are defined to be contiguous strings of characters delimited by one or more blanks, tabs, or new-lines; empty lines are always discarded. Blanks and tabs may be embedded as part of an argument if escaped or quoted. Characters enclosed in quotes (single or double) are taken literally, and the delimiting quotes are removed. Outside of quoted strings a backslash (\) will escape the next character.

Each argument list is constructed starting with the initial-arguments, followed by some number of arguments read from standard input (Exception: see --flag). Flags --l, --l, and --n determine how arguments are selected for each command invocation. When none of these flags are coded, the initial-arguments are followed by arguments read continuously from standard input until an internal buffer is full, and then command is executed with the accumulated args. This process is repeated until there are no more args. When there are flag conflicts (e.g., --l vs. --n), the last flag has precedence. Flag values are:

--lnumber command is executed for each non-empty number lines of arguments from standard input. The last invocation of command will be with fewer lines of arguments if fewer than number remain. A line is considered to end with the first new-line unless the last character of the line is a blank or a tab; a trailing blank/tab signals continuation through the next non-empty line. If number is omitted, 1 is assumed. Option --x is forced.

--lreplstr Insert mode: command is executed for each line from standard input, taking the entire line as a single arg, inserting it in initial-arguments for each occurrence of replstr. A maximum of 5 arguments in initial-arguments may each contain one or more instances of replstr. Blanks and tabs at the beginning of each line are thrown away. Constructed arguments may not grow larger than 255 characters, and option --x is also forced. (1) is assumed for replstr if not specified.

--nnumber Execute command using as many standard input arguments as possible, up to number arguments maximum. Fewer arguments will be used if their total size is greater than size characters, and for the last invocation if there are fewer than number arguments remaining. If option --x is also coded, each number arguments must fit in the size limitation, else xargs terminates execution.

--t Trace mode: The command and each constructed argument list are echoed to file descriptor 2 just prior to their execution.

--p Prompt mode: The user is asked whether to execute command each invocation. Trace mode (--t) is turned on to print the command instance to be executed, followed by a ?... prompt. A reply of y (optionally
XARGS(1)

followed by anything) will execute the command; anything else, including just a carriage return, skips that particular invocation of command.

-x
Causes xargs to terminate if any argument list would be greater than size characters; -x is forced by the options -i and -1. When neither of the options -i, -1, or -n are coded, the total length of all arguments must be within the size limit.

-ssize
The maximum total size of each argument list is set to size characters; size must be a positive integer less than or equal to 470. If -s is not coded, 470 is taken as the default. Note that the character count for size includes one extra character for each argument and the count of characters in the command name.

-eofstr
EOFstr is taken as the logical end-of-file string. Underbar (_) is assumed for the logical EOF string if -e is not coded. The value -e with no eofstr coded turns off the logical EOF string capability (underbar is taken literally). xargs reads standard input until either end-of-file or the logical EOF string is encountered.

xargs will terminate if either it receives a return code of -1 from, or if it cannot execute, command. When command is a shell program, it should explicitly exit (see sh(1)) with an appropriate value to avoid accidentally returning with -1.

EXAMPLES

The following will move all files from directory $1 to directory $2, and echo each move command just before doing it:

    ls $1 | xargs -i -t mv $1/[] $2/[

The following will combine the output of the parenthesized commands onto one line, which is then echoed to the end of file log:

    (logname; date; echo $0 $*) | xargs >>log

The user is asked which files in the current directory are to be archived and archives them into arch (1.) one at a time, or (2.) many at a time.

1. ls | xargs -p -1 ar arch
2. ls | xargs -p -1 l xargs ar r arch

The following will execute diff(1) with successive pairs of arguments originally typed as shell arguments:

    echo $* | xargs -n2 diff

SEE ALSO

sh(1).
NAME

xstr – extract strings from C programs to implement shared strings

SYNOPSIS

xstr [-c] [-] [ file ]

DESCRIPTION

xstr maintains a file strings into which strings in component parts of a large program are hashed. These strings are replaced with references to this common area. This serves to implement shared constant strings, most useful if they are also read-only.

The command

xstr -c name

will extract the strings from the C source in name, replacing string references by expressions of the form (&xstr[number]) for some number. An appropriate declaration of xstr is prepended to the file. The resulting C text is placed in the file x.c, to then be compiled. The strings from this file are placed in the strings data base if they are not there already. Repeated strings and strings which are suffixes of existing strings do not cause changes to the data base.

After all components of a large program have been compiled a file xs.c declaring the common xstr space can be created by a command of the form

xstr

This xs.c file should then be compiled and loaded with the rest of the program. If possible, the array can be made read-only (shared) saving space and swap overhead.

xstr can also be used on a single file. A command

xstr name

creates files x.c and xs.c as before, without using or affecting any strings file in the same directory.

It may be useful to run xstr after the C preprocessor if any macro definitions yield strings or if there is conditional code which contains strings which may not, in fact, be needed. xstr reads from its standard input when the argument ‘-’ is given. An appropriate command sequence for running xstr after the C preprocessor is:

cc -E name.c | xstr -c -
cc -c x.c
mv x.o name.o

xstr does not touch the file strings unless new items are added, thus make can avoid remaking xs.o unless truly necessary.

FILES

strings     Data base of strings
x.c        Massaged C source
xs.c        C source for definition of array ‘xstr’
/tmp/xs*    Temp file when ‘xstr name’ doesn’t touch strings

SEE ALSO

mkstr(1)

BUGS

If a string is a suffix of another string in the data base, but the shorter string is seen first by xstr both strings will be placed in the data base, when just placing the longer one there will do.
NAME
yacc – yet another compiler-compiler

SYNOPSIS
yacc [-vdl] grammar

DESCRIPTION
The yacc command converts a context-free grammar into a set of tables for a simple
automaton which executes an LR(1) parsing algorithm. The grammar may be ambi-
guous; specified precedence rules are used to break ambiguities.

The output file, y.tab.c, must be compiled by the C compiler to produce a program
yyparse. This program must be loaded with the lexical analyzer program, yylex, as
well as main and yyerror, an error handling routine. These routines must be supplied
by the user; lex(1) is useful for creating lexical analyzers usable by yacc.

If the -v flag is given, the file y.output is prepared, which contains a description of
the parsing tables and a report on conflicts generated by ambiguities in the grammar.

If the -d flag is used, the file y.tab.h is generated with the #define statements that
associate the yacc-assigned “token codes” with the user-declared “token names”.
This allows source files other than y.tab.c to access the token codes.

If the -l flag is given, the code produced in y.tab.c will not contain any #line con-
structs. This should only be used after the grammar and the associated actions are
fully debugged.

Runtime debugging code is always generated in y.tab.c under conditional compila-
tion control. By default, this code is not included when y.tab.c is compiled. How-
ever, when yacc’s -t option is used, this debugging code will be compiled by default.
Independent of whether the -t option was used, the runtime debugging code is
under the control of YYDEBUG, a preprocessor symbol. If YYDEBUG has a non-zero
value, then the debugging code is included. If its value is zero, then the code will not
be included. The size and execution time of a program produced without the run-
time debugging code will be smaller and slightly faster.

FILES
y.output
y.tab.c
y.tab.h defines for token names
yacc.tmp,
 yacc.debug, yacc.acts temporary files
/usr/lib/yaccpar parser prototype for C programs

SEE ALSO
lex(1),
Programmer’s Guide.

DIAGNOSTICS
The number of reduce-reduce and shift-reduce conflicts is reported on the standard
error output; a more detailed report is found in the y.output file. Similarly, if some
rules are not reachable from the start symbol, this is also reported.

CAVEAT
Because file names are fixed, at most one yacc process can be active in a given direc-
tory at a given time.
NAME
yes – be repetitively affirmative

SYNOPSIS
yes [ expletive ]

DESCRIPTION
yes repeatedly outputs "yes", or if expletive is given, that is output repeatedly. Termination is by rubout.
NAME
intro – introduction to system maintenance procedures

DESCRIPTION
This section outlines certain procedures that will be of interest to those charged with the task of system maintenance. Included are discussions of such topics as boot procedures, recovery from crashes, file backups, etc.

SEE ALSO
Installation/Administration Guide.
NAME

accton – system accounting

SYNOPSIS

/etc/accton [ file ]

DESCRIPTION

With an argument naming an existing file, accton causes system accounting information for every process executed to be placed at the end of the file. If no argument is given, accounting is turned off.

SEE ALSO

acct(2)
arp – address resolution display and control

arp hostname
arp -a [ unix ] [ kmem ]
arp -d hostname
arp -s hostname ether_addr [ temp ] [ pub ] [ trail ]
arp -f filename

The arp program displays and modifies the Internet-to-Ethernet address translation tables used by the address resolution protocol (arp(4p)).

With no flags, the program displays the current ARP entry for hostname. The host may be specified by name or by number, using Internet dot notation. With the -a flag, the program displays all of the current ARP entries by reading the table from the file kmem (default /dev/kmem) based on the kernel file unix (default /unix).

With the -d flag, a super-user may delete an entry for the host called hostname.

The -s flag is given to create an ARP entry for the host called hostname with the Ethernet address ether_addr. The Ethernet address is given as six hex bytes separated by colons. The entry will be permanent unless the word temp is given in the command. If the word pub is given, the entry will be “published”; i.e., this system will act as an ARP server, responding to requests for hostname even though the host address is not its own. The word trail indicates that trailer encapsulations may be sent to this host.

The -f flag causes the file filename to be read and multiple entries to be set in the ARP tables. Entries in the file should be of the form

    hostname ether_addr [ temp ] [ pub ] [ trail ]

with argument meanings as given above.

SEE ALSO

inet(3N), arp(4)
NAME

boot – system boot procedure

SYNOPSIS

Autoboot:

Ensure the bottom key switch is in the horizontal position.
Turn the top key switch one position to the right.

Manual Boot:

prom X> b [<boot device>][<boot file>] [-s | -m] [rootdev=<root device>]
[swapdev=<swap device>]

eg:

prom X> b
prom X> b unix
prom X> b /usr/ken/unix
prom X> b scsi(1,5,1)unix rootdev=scsi(1,5,1)

DESCRIPTION

The front panel key switches and the PROM monitor environment variables control
the boot sequence.

The front panel key switches are located under the decorative, corrugated facade on
the front of the machine. This panel can be slid up and down to expose or conceal the
tape drive and key switches.

The upper key switch has three positions: off (vertical), on (the first position to the
right), and reset (the second, spring-loaded position to the right). The lower key
switch defines the action to be taken when the upper key is turned into the spring-
loaded position. The lower key switch has two positions: standard (the horizontal
position) and maintenance (the vertical position). For normal operations, keep the
lower switch in the standard, horizontal position. This will enable autobooting to the
run level specified in /etc/inittab. Place the lower key switch in the maintenance,
vertical position to manually boot the machine, boot UNIX in single-user mode, or to
interrupt UNIX.

The Prom Monitor retains environment variables in non-volatile memory which are
used to define the default actions taken by the boot process. These variables specify:
the boot device (disk, tape, or ethernet), the boot file (UNIX or a stand alone pro-
gram), the default root and swap disk partitions, and the name and location of the
secondary boot program. The variable settings given below specify to autoboot
UNIX from the SCSI controller 1, unit (disk) 5, partition 0.

   bootmode  =  a
   bootfile  =  unix
   path      =  scsi(1,5,0)
   rootdev   =  scsi(1,5,0)
   swapdev   =  scsi(1,5,1)
   secondary =  scsi(1,5,8)sash

To disable autoboot, unset the bootmode environment variable. Use the Prom Moni-
tor n command to display, set, and unset Prom Monitor Environment Variables.

When booting UNIX manually, additional arguments may be specified which will
override the settings of the environment Variables.
<table>
<thead>
<tr>
<th>Argument</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-s</td>
<td>Boot unix to single-user run level</td>
</tr>
<tr>
<td>-m</td>
<td>Boot unix to the run level specified in /etc/inittab</td>
</tr>
<tr>
<td>-rootdev=</td>
<td>specify the root device and partition, e.g., scsi(1,5,1)</td>
</tr>
<tr>
<td>-swapdev=</td>
<td>specify the swap device and partition, e.g., scsi(0,5,1)</td>
</tr>
</tbody>
</table>

**FILES**

scsi(1,5,0)unix -- standard UNIX kernel  
scsi(1,5,8)sash -- secondary boot program

**SEE ALSO**

intro(7), shutdown(1M), dvhtool(1M)  
*Installation/Administration Guide* appendix, *Prom Manual*

**BUGS**

It is sometimes necessary to issue a Prom reset command after a boot failure to boot successfully.
NAME

catman – create the cat files for the manual

SYNOPSIS

/etc/catman [-p] [-n] [-w] [-M path] [ sections ]

DESCRIPTION

catman creates the preformatted versions of the on-line manual from the nroff input
files. Each manual page is examined and those whose preformatted versions are
missing or out of date are recreated. If any changes are made, catman will recreate
the whatis database.

If there is one parameter not starting with a ‘-’, it is taken to be a list of manual sec-
tions to look in. For example

catman 123

will cause the updating to only happen to manual sections 1, 2, and 3.

Options:

- n  prevents creations of the whatis database.
- p  prints what would be done instead of doing it.
- w  causes only the whatis database to be created. No manual reformatting is
done.
- M  updates manual pages located in the set of directories specified by path
(/usr/man by default). Path has the form of a colon (‘:’) separated list of direc-
tory names, for example ‘/usr/local/man:/usr/man’. If the environment vari-
able ‘MANPATH’ is set, its value is used for the default path.

If the nroff source file contains only a line of the form ‘.so man/x/y.x’, a symbolic
link is made in the catx directory to the appropriate preformatted manual page.

FILES

/usr/man    default manual directory location
/usr/man/man?/*./* raw (nroff input) manual sections
/usr/man/cat?/*./* preformatted manual pages
/usr/man/bsd  manual directory location for bsd man pages
/usr/man/whatis whatis database
/usr/lib/makewhatis command script to make whatis database

SEE ALSO

man(1)
NAME
comsat – biff server

SYNOPSIS
/etc/comsat

DESCRIPTION
Comsat is the server process which receives reports of incoming mail and notifies users if they have requested this service. Comsat receives messages on a datagram port associated with the "biff" service specification (see services(5) and inetd(8)). The one line messages are of the form

```
user@mailbox-offset
```

If the user specified is logged in to the system and the associated terminal has the owner execute bit turned on (by a "biff y"), the offset is used as a seek offset into the appropriate mailbox file and the first 7 lines or 560 characters of the message are printed on the user's terminal. Lines which appear to be part of the message header other than the "From", "To", "Date", or "Subject" lines are not included in the displayed message.

FILES
/etc/utmp to find out who's logged on and on what terminals

SEE ALSO
biff(1), inetd(8)

BUGS
The message header filtering is prone to error. The density of the information presented is near the theoretical minimum.

Users should be notified of mail which arrives on other machines than the one to which they are currently logged in.

The notification should appear in a separate window so it does not mess up the screen.
**NAME**
dump – capture the state of the memory for later analysis using crash(1M)

**SYNOPSIS**
Use the Prom Monitor to copy memory to the default dump device:
prom X> sysdump

Reboot the system

Use **dd** to copy unix and system dump to cartridge tape:
```
dd if=/unix of=/dev/rmtx bs=512k
dd if=/dev/dsk/c1d5s1 of=/dev/rmtXX bs=512k count=64
```

**DESCRIPTION**
Use the procedure given above to create a system dump.

If UNIX is hung, attempt to interrupt UNIX by placing the lower key switch in the maintenance (vertical) position and then turning the upper key switch to the spring-loaded position to the right. If this fails, reset the machine and request access to the Prom Monitor. This is accomplished by:

Turning the lower key to the standard (horizontal) position. Turning the upper key to the spring loaded-position to the right. Then, within three seconds, turning the lower key to the maintenance (vertical) position. If the Prom Environment variable **bootmode** is not set and thereby disabling autoboott, only the first two steps are necessary. Note: the above procedure may require two hands.

Once the Prom Monitor has control, follow the procedure outlined in the synopsis.

Note than when coping UNIX to the tape one should use a non-rewind on close, high density tape device. Then when coping the contents of the dump device to tape, one should specify a count which describes the size of main memory.

The Prom Environment Variable **dumpdev** describes the default device which will be used by the Prom Monitor to hold the system dump. The variable **dumpdev** is normally set to the swap partition.

**WARNINGS**
The tape is by convention written in high density format and blocked in 512K byte blocks. If convention has not been followed, please indicate the procedure used on the tape label.

**FILES**
```
/dev/xxx -- high density no rewind on close cartridge tape device
/dev/xxx -- high density cartridge tape device
```

**SEE ALSO**
boot(7M), Titian Prom Manual
NAME

ftpd – DARPA Internet File Transfer Protocol server

SYNOPSIS

/etc/ftpd [-d] [-l] [-timeout]

DESCRIPTION

_ftpd_ is the DARPA Internet File Transfer Protocol server process. The server uses the TCP protocol and listens at the port specified in the "ftp" service specification; see _services_(5).

If the _-d_ option is specified, debugging information is written to the syslog.

If the _-l_ option is specified, each ftp session is logged in the syslog.

The ftp server will timeout an inactive session after 15 minutes. If the _-t_ option is specified, the inactivity timeout period will be set to _timeout_.

The ftp server currently supports the following ftp requests; case is not distinguished.

<table>
<thead>
<tr>
<th>Request</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABOR</td>
<td>abort previous command</td>
</tr>
<tr>
<td>ACCT</td>
<td>specify account (ignored)</td>
</tr>
<tr>
<td>ALLO</td>
<td>allocate storage (vacuously)</td>
</tr>
<tr>
<td>APPE</td>
<td>append to a file</td>
</tr>
<tr>
<td>CDUP</td>
<td>change to parent of current working directory</td>
</tr>
<tr>
<td>CWD</td>
<td>change working directory</td>
</tr>
<tr>
<td>DELE</td>
<td>delete a file</td>
</tr>
<tr>
<td>HELP</td>
<td>give help information</td>
</tr>
<tr>
<td>LIST</td>
<td>give list files in a directory (&quot;Is -lg&quot;)</td>
</tr>
<tr>
<td>MKD</td>
<td>make a directory</td>
</tr>
<tr>
<td>MODE</td>
<td>specify data transfer <em>mode</em></td>
</tr>
<tr>
<td>NLST</td>
<td>give name list of files in directory (&quot;Is&quot;)</td>
</tr>
<tr>
<td>NOOP</td>
<td>do nothing</td>
</tr>
<tr>
<td>PASS</td>
<td>specify password</td>
</tr>
<tr>
<td>PASV</td>
<td>prepare for server-to-server transfer</td>
</tr>
<tr>
<td>PORT</td>
<td>specify data connection port</td>
</tr>
<tr>
<td>PWD</td>
<td>print the current working directory</td>
</tr>
<tr>
<td>QUIT</td>
<td>terminate session</td>
</tr>
<tr>
<td>RETR</td>
<td>retrieve a file</td>
</tr>
<tr>
<td>RMD</td>
<td>remove a directory</td>
</tr>
<tr>
<td>RNFR</td>
<td>specify rename-from file name</td>
</tr>
<tr>
<td>RNTO</td>
<td>specify rename-to file name</td>
</tr>
<tr>
<td>STOR</td>
<td>store a file</td>
</tr>
<tr>
<td>STOU</td>
<td>store a file with a unique name</td>
</tr>
<tr>
<td>STRU</td>
<td>specify data transfer <em>structure</em></td>
</tr>
<tr>
<td>TYPE</td>
<td>specify data transfer <em>type</em></td>
</tr>
<tr>
<td>USER</td>
<td>specify user name</td>
</tr>
<tr>
<td>XCUP</td>
<td>change to parent of current working directory</td>
</tr>
<tr>
<td>XCWD</td>
<td>change working directory</td>
</tr>
<tr>
<td>XMKD</td>
<td>make a directory</td>
</tr>
<tr>
<td>XFPWD</td>
<td>print the current working directory</td>
</tr>
<tr>
<td>XRMD</td>
<td>remove a directory</td>
</tr>
</tbody>
</table>

The remaining ftp requests specified in Internet RFC 959 are recognized, but not implemented.
The ftp server will abort an active file transfer only when the ABOR command is preceded by a Telnet "Interrupt Process" (IP) signal and a Telnet "Synch" signal in the command Telnet stream, as described in Internet RFC 959.

Ftpd interprets file names according to the "globbing" conventions used by csh(1). This allows users to utilize the metacharacters "+?[]~".

Ftpd authenticates users according to three rules:

1) The user name must be in the password data base, /etc/passwd, and not have a null password. In this case a password must be provided by the client before any file operations may be performed.
2) The user name must not appear in the file /etc/ftpusers.
3) The user must have a standard shell returned by getusershell(3).
4) If the user name is "anonymous" or "ftp", an anonymous ftp account must be present in the password file (user "ftp"). In this case the user is allowed to log in by specifying any password (by convention this is given as the client host's name).

In the last case, ftpd takes special measures to restrict the client's access privileges. The server performs a chroot(2) command to the home directory of the "ftp" user. In order that system security is not breached, it is recommended that the "ftp" subtree be constructed with care; the following rules are recommended.

~ftp

Make the home directory owned by "ftp" and unwritable by anyone.

~ftp/bin

Make this directory owned by the super-user and unwritable by anyone. The program ls(1) must be present to support the list commands. This program should have mode 111.

~ftp/etc

Make this directory owned by the super-user and unwritable by anyone. The files passwd(5) and group(5) must be present for the ls command to work properly. These files should be mode 444.

~ftp/pub

Make this directory mode 777 and owned by "ftp". Users should then place files which are to be accessible via the anonymous account in this directory.

~ftp/dev

Make this directory owned by the super-user and unwritable by anyone. Then mknod(1) for tcp in this directory:

```
  mknod ~ftp/dev/tcp c 14 0
cchmod 666 ~ftp/dev/tcp
```

SEE ALSO

ftp(1C), getusershell(3), syslogd(8)

BUGS

The anonymous account is inherently dangerous and should avoided when possible.

The server must run as the super-user to create sockets with privileged port numbers. It maintains an effective user id of the logged in user, reverting to the super-user only when binding addresses to sockets. The possible security holes have been extensively scrutinized, but are possibly incomplete.
NAME

gated – gateway routing daemon

SYNOPSIS

gated [-t [i][e][r][p][u][R][H]][logfile]

DESCRIPTION

gated is a routing daemon that handles multiple routing protocols and replaces
routed(8), egpup(8), and any routing daemon that speaks the HELLO routing protocol.
gated currently handles the RIP, EGP, and HELLO routing protocols. The gated process
can be configured to perform all routing protocols or any combination of the three.
The configuration for gated is by default stored in the file /etc/gated.conf and can be
changed at compile time in the file defsh.

COMMAND LINE TRACING OPTIONS

gated can be invoked with a number of tracing flags and/or a log file. Tracing flags
may also be specified in the configuration file with the traceflags clause. gated forks
and detaches itself from the controlling terminal unless tracing flags are specified
without specifying a log file, in which case all tracing output is sent to the controlling
terminal. The valid tracing flags are as follows:

-t If used alone, log all error messages, route changes and EGP packets sent
and received. Using this flag alone turns on the i, e, r, and p trace flags
automatically. When used with another flag, the -t flag has no effect and only
the accompanying flags are recognized. Note that when using other flags,
the -t flag must be used with them.
i Log all internal errors and interior routing errors.
e Log all external errors due to EGP, exterior routing errors, and EGP state
changes.
r Log all routing changes.
p Trace all EGP packets sent and received.
u Log all routing updates sent.
R Trace all RIP packets received.
H Trace all HELLO packets received.

The gated process always logs fatal errors. If no log file is specified and no tracing
flags are set, all messages are sent to /dev/null.

SIGNAL PROCESSING

gated catches a number of signals and performs specific actions. Currently gated does
special processing with the SIGHUP and SIGINT signals.

When a SIGHUP is sent to the gated process and gated is invoked with trace flags and a
log file, tracing is toggled off and the log file is closed. At this point the log file may
be moved or removed. The next SIGHUP to gated will toggle the tracing on. gated
reads the configuration file and sets the tracing flags to those specified with the
traceflags clause. If no traceflags clause is specified tracing is resumed using the trace
flags specified on the command line. The log file specified in the command line is
created if necessary and the trace output is sent to that file. The trace output is
appended to an already existing log file. This is useful for having rotating log files
like those of the syslog daemon.

Sending gated a SIGINT will cause a memory dump to be scheduled within the next
sixty seconds. The memory dump will be written to the file /usr/tmp/gated_dump.
gated will finish processing pending routing updates before performing the memory
dump. The memory dump contains a snapshot of the current gated status, including

Stardent 1500/3000 11
the interface configurations, EGP neighbor status and the routing tables. If the file 
/usr/tmp/gated_dump already exists, the memory dump will be appended to the exist-
ing file.

**CONFIGURATION FILE OPTIONS FOR CONTROLLING TRACING OUTPUT**

traceflags [traceflag] [traceflag] ...  
The clause tells the *gated* process what level of tracing output is desired. This option 
is read during *gated* initialization and whenever *gated* receives a SIGHUP. This option 
is overridden at initialization time if tracing flags are specified on the command line. 
The valid tracing flags are as follows:

- **internal** Log all internal errors and interior routing errors.
- **external** Log all external errors due to EGP, exterior routing errors, and EGP state 
  changes.
- **route** Log all routing changes.
- **egp** Trace all EGP packets sent and received.
- **update** Log all routing updates sent.
- **rip** Trace all RIP packets received.
- **hello** Trace all HELLO packets received.
- **general** A combination of **internal**, **external**, **route**, and **egp**.
- **all** Enable all of the above tracing flags.

If more than one *traceflags* clause is used, the tracing flags accumulate.

**CONFIGURATION FILE OPTIONS FOR HANDLING ROUTING PROTOCOLS**

In this section, the numerous configuration options are explained. Each time the 
*gated* process is started, it reads the file */etc/gated.conf* to obtain its instructions on how 
routing will be managed with respect to each protocol. The configuration options are 
as follows:

- **RIP (yes | no | supplier | pointopoint | quiet | gateway *)**

  This tells the *gated* process how to perform the RIP routing protocol. Only one of the 
  above RIP arguments is allowed after the keyword RIP. If more than one is specified, 
  only the first one is recognized. A list of the arguments to the RIP clause follows:

  - **yes** Perform the RIP protocol. Process all incoming RIP packets and supply RIP 
    information every thirty seconds only if there are two or more network 
    interfaces.
  - **no** Do not perform the RIP protocol. Do not perform RIP.
  - **supplier** Perform the RIP protocol. Process all incoming RIP packets and force the 
    supplying of RIP information every thirty seconds no matter how many net-
    work interfaces are present.
  - **pointopoint** Perform the RIP protocol. Process all incoming RIP packets and force the 
    supplying of RIP information every thirty seconds no matter how many net-
    work interfaces are present. When this argument is specified, RIP information 
    will not be sent out in a broadcast packet. The RIP information will be 
    sent directly to the gateways listed in the *sourceripgateways* option described 
    below.
quiet  Process all incoming RIP packets, but do not supply any RIP information no matter how many network interfaces are present.

gateway #  Process all incoming RIP packets, supply RIP information every thirty seconds, and announce the default route (0.0.0.0) with a metric of #. The metric should be specified in a value that represents a RIP hopcount. With this option set, all other default routes coming from other RIP gateways will be ignored. The default route is only announced when actively peering with at least one EGP neighbor and therefore should only be used when EGP is used.

If no RIP clause is specified, RIP will not be performed.

HELLO [yes | no | supplier | pointopoint | quiet | gateway #]

This tells the gated process how to perform the HELLO routing protocol. The arguments parallel the RIP arguments, but do have some minor differences. Only one of the above HELLO arguments is allowed after the keyword HELLO. If more than one is specified, only the first one is recognized. A list of the arguments to the HELLO clause follows:

yes  Perform the HELLO protocol. Process all incoming HELLO packets and supply HELLO information every fifteen seconds only if there are two or more network interfaces.

no   Do not perform the HELLO protocol. Do not perform HELLO.

supplier  Perform the HELLO protocol. Process all incoming HELLO packets and force the supplying of HELLO information every fifteen seconds no matter how many network interfaces are present.

pointopoint  Perform the HELLO protocol. Process all incoming HELLO packets and force the supplying of HELLO information every fifteen seconds no matter how many network interfaces are present. When this argument is specified, HELLO information will not be sent out in a broadcast packet. The HELLO information will be sent directly to the gateways listed in the sourcehellogateways option described below.

quiet  Process all incoming HELLO packets, but do not supply any HELLO information despite the number of network interfaces present.

gateway #  Process all incoming HELLO packets, supply HELLO information every fifteen seconds, and announce the default route (0.0.0.0) with a time delay of #. The time delay should be specified in milliseconds. The default route is only announced when actively peering with at least one of EGP neighbor, therefore should only be used when running EGP.

If no HELLO clause is specified, HELLO will not be performed.

EGP [yes | no]

This clause allows the processing of EGP by gated to be turned on or off.

no   Do not perform any EGP processing.

yes  Perform all EGP operations.

Please note that by default, EGP processing will take place. Therefore, if no EGP clause is specified, all EGP operations will take place.
autonomoussystem 

If performing the EGP protocol, this clause must be used to specify the autonomous system number (H). If not specified, gated will exit and give a fatal error message.

egpmaxacquire 

If performing the EGP protocol, this clause specifies the number of EGP peers with whom gated will be performing EGP. This number must be greater than zero and less than or equal to the number of EGP neighbors specified or gated will exit.

egpneighbor gateway 

If performing the EGP protocol, this clause specifies with whom gated will be performing EGP. gateway can be either a symbolic name in /etc/hosts or an IP host address in Internet dot notation (a.b.c.d) notation. Dot notation is recommended to avoid confusion. Each EGP neighbor will be acquired in the order listed in the configuration file.

CONFIGURATION FILE OPTIONS FOR HANDLING ROUTING INFORMATION

The following configuration file options tell gated how to deal with both incoming and outgoing routing information.

trustedripgateways gateway [gateway] [gateway] ..... 

trustedheloigateways gateway [gateway] [gateway] ..... 

When these clauses are specified, gated will only listen to RIP or HELLO information, respectively from these RIP or HELLO gateways. gateway can be either a symbolic name from /etc/hosts or an IP host address in dot notation (a.b.c.d). Again, dot notation is recommended to eliminate confusion. Please note that the propagation of routing information is not restricted by this clause.

sourceripgateways gateway [gateway] [gateway] ..... 

sourceheloigateways gateway [gateway] [gateway] ..... 

gated will send RIP or HELLO information directly to the gateways specified. If point-to-point is specified in the RIP or HELLO clauses mentioned above, gated will only send RIP or HELLO information to the specified gateways. gated will NOT send out any information using the broadcast address. If point-to-point is not specified in those clauses and gated is supplying RIP or HELLO information, gated will send information to the specified gateways as well as broadcasting it using a broadcast address.

noripoutinterface intfaddr [intfaddr] [intfaddr] ..... 

nohelooutinterface intfaddr [intfaddr] [intfaddr] ..... 

noripfrominterface intfaddr [intfaddr] [intfaddr] [intfaddr] ..... 

nohelofrominterface intfaddr [intfaddr] [intfaddr] [intfaddr] ..... 

The above clauses turn protocols on and off on a per interface basis. no(rip|hello)frominterface means that no RIP or HELLO information will be accepted coming into the listed interfaces from another gateway. no(rip|hello)outinterface means that no RIP or HELLO knowledge will be sent out of the listed interfaces. intfaddr should be in dot notation (a.b.c.d).

passiveinterfaces intfaddr [intfaddr] [intfaddr] ..... 

In order to dynamically determine if an interface is properly functioning, gated will time out an interface when no RIP, HELLO, or EGP packets are being received on that particular interface. PSN interfaces send a RIP or HELLO packet to themselves to determine if the interface is properly functioning as the delay between EGP packets may be longer than the interface timeout. Interfaces that have timed out automatically have their routes re-installed when routing information is again received over
the interface. The above clause stops *gated* from timing out the listed interfaces. The interfaces listed will always be considered up and working. If *gated* is not a RIP or HELLO supplier, all interfaces will not be aged and the *passive interfaces* automatically applies to all interfaces.

```
interfacemetric intfaddr metric#
```

This feature allows the specification of an interface metric for the listed interface. On systems that support interface metrics, this clause will override the kernel’s metric. On systems that do not have support for an interface metric, this feature allows the specification of one. The interface metric is added to the true metric of each route that comes in via routing information from the listed interface. The interface metric is also added to the true metric of any information sent out via the listed interface. This clause is required for each interface on which an interface metric is desired.

```
reconstmetric intfaddr metric#
```

This is a first attempt to throw hooks for fallback routing into *gated*. If the above clause is used, the metrics of the routes contained in any RIP information coming into the listed interface will be set to the specified metric#. Metric reconstitution should not be used lightly, since it could be a major contributor in the formation of routing loops. USE THIS WITH EXTREME CAUTION. Any route that has a metric of infinity will not be reconstituted and left as infinity.

```
fixedmetric intfaddr proto {rip | hello} metric#
```

This is another attempt to throw hooks for fallback routing into *gated*. If the above clause is used, all routing information sent out the specified interface will have a metric of *metric*#. For RIP, specify the metric as a RIP hopcount from 0 to infinity. For HELLO, specify the metric as a HELLO delay in milliseconds from 0 to infinity. Any route that has a metric of infinity will be left as infinity. Fixed metrics should also be USED WITH EXTREME CAUTION!

```
donotlisten net intf addr [addr] ... proto {rip | hello}
donotlistenhost host intf addr [addr] ... proto {rip | hello}
```

This clause reads as follows: keyword *donotlisten* followed by a network number, which should be in dot notation followed by keyword *intf*. Then a list of interfaces in dot notation precede the keyword *proto*, followed by *rip* or *hello*.

This means that any information regarding *net* coming in via the specified protocols AND from the specified interfaces will be ignored. The keyword *all* may be used after the keyword *intf* to specify all interfaces on the machine. For example:

```
donotlisten 10.0.0.0 intf 128.84.253.200 proto rip
```

means that any RIP information about net 10.0.0.0 coming in via interface 128.84.253.200 will be ignored. One clause is required for each net on which this restriction is desired.

```
donotlisten 26.0.0.0 intf all proto rip hello
```

means that any RIP and HELLO information about net 26.0.0.0 coming in via any interface will be ignored.

*donotlistenhost* can be described the same way as above except that a host address is provided instead of a network address. Restrictions of the nature described above are applied to the specified host route learned of by the specified routing protocol.

```
listen net gateway addr [addr] ... proto {rip | hello}
listenhost host gateway addr [addr] ... proto {rip | hello}
```
This clause reads as follows: keyword *listen* followed by a network number which should be in dot notation followed by keyword *gateway*. Then a list of gateways in dot notation should precede the keyword *proto*, followed by *rip* or *hello*.

This means to only listen to information about network net by the specified protocol(s) only from the listed gateways. For example:

```
listen 128.84.0.0 gateway 128.84.253.3 proto hello
```

means that any HELLO information about net 128.84 coming in via gateway 128.84.253.3 will be accepted. Any other information about 128.84 from any other gateway will be rejected. One clause is necessary for each net to be restricted.

```
listenablest 26.0.0.15 gateway 128.84.253.3 proto rip
```

means that any information about host 26.0.0.15 must come via RIP and from gateway 128.84.253.3. All other information regarding this host will be ignored.

```
announce net intf addr [addr] ... proto type [egpmetric #]
announcehost host intf addr ... proto type [egpmetric #]
noannounce net intf addr [addr] ... proto type [egpmetric #]
noannouncehost host intf addr ... proto type [egpmetric #]
```

These clauses allow restriction of the networks and hosts announced and by which protocol. The *announce(host)* and *noannounce(host)* clauses may not be used together on the same interface. With the *announce(host)* clause, *gated* will only announce the nets or hosts that have an associated *announce(host)* clause with the appropriate protocol. With the *noannounce(host)* clause, *gated* will announce everything, EXCEPT those nets or hosts that have an associated *noannounce(host)* clause. This allows a choice of announcing only what is on the announce list or everything except those nets on the noannounce list on a per interface basis.

The arguments are the same as in the *donotlisten* clause except *egp* may be specified in the *proto* field. *type* can either be *rip*, *hello*, *egp*, or any combination of the three. When *egp* is specified in the *proto* field, an *egp* metric must be specified. This is the metric at which *gated* will announce the listed net via EGP.

Please note that these are not static route entries. These restrictions will only apply if the net or host is learned via one of the routing protocols. If a restricted network suddenly becomes unreachable and goes away, announcement of this net will stop until it is learned again.

Currently, only one *announce(host)* or *noannounce(host)* may be specified per network or host. It is not possible to announce a network or host via HELLO out one interface and via RIP out another.

Some examples:

```
announce 128.84 intf all proto rip hello egp egpmetric 0
announce 10.0.0.0 intf all proto rip
announce 0.0.0.0 intf 128.84.253.200 proto rip
announce 35.0.0.0 intf all proto rip egp egpmetric 3
```

With only these four *announce* clauses in the configuration file, this *gated* process will only announce these four nets. It will announce 128.84.0.0 via RIP and HELLO to all interfaces and announce it via EGP with a metric of 0. Net 10.0.0.0 will be announced via RIP to all interfaces. Net 0.0.0.0 (default) will be announced by RIP out interface 128.84.253.200 only. Net 35.0.0.0 will be announced via RIP to all interfaces and announced via EGP with a metric of 3. These are the only nets that will be broadcast
by this gateway. Once the first announce clause is specified, only the nets with announce clauses will be broadcast; this includes local subnets. Once an announce(host) or noannounce(host) has an all specified after an intf, that clause is applied globally and the option of having per interface restrictions is lost. If no routing announcement restrictions are desired, announce clauses should not be used. All information learned will then be propagated out. Please note that this has no affect on the information to which gated listens. Any net that does not have an announce clause is still added to the kernel routing tables, but it is not announced via any of the routing protocols. To stop nets from being added to the kernel the donotlisten clause may be used.

announce 128.84 intf 128.59.2.1 proto rip

noannounce 128.84 intf 128.59.1.1 proto rip

The above clauses mean that on interface 128.59.2.1, only information about 128.84.0.0 will be announced via RIP, but on interface 128.59.1.1, all information will be announced, except 128.84.0.0 via RIP.

noannounce 128.84 intf all proto rip hello egp egpmetric 0

noannounce 10.0.0.0 intf all proto hello

These clauses mean that except for the two specified nets, all nets will be propagated. Specifically, net 128.84.0.0 will not be announced on any interface via any protocols. Knowledge of 128.84.0.0 is not sent anywhere. Net 10.0.0.0 will not be announced via HELLO to any interface. This also implies that net 10.0.0.0 will be announced to every interface via RIP. This net will also be broadcast via EGP with a metric specified in the defaultegpmetric clause.

defaultegpmetric #

This is a default EGP metric to use when there are no routing restrictions. Normally, with no routing restrictions, gated announces all networks learned via HELLO or RIP via EGP with this specified default EGP metric. If this clause is not used, the default EGP metric is set to 255, which would make any EGP advertised route of this nature be ignored. When there are no routing restrictions, any network with a direct interface is announced via EGP with a metric of 0. Note that this does not include subnets, but only the non-subnetted network.

defaultrouter gateway gateway proto [active | passive]

This initial default gateway is installed in the kernel routing tables during startup and initialization. If EGP is being used, as soon as an update is received from an EGP neighbor, this default route is deleted. If EGP is not being used, but RIP or HELLO are, and active is specified, this default route will be deleted and overwritten by default learned from any other RIP or HELLO gateway. If no default route is learned via RIP or HELLO, this default route will be maintained in the kernel routine tables. If passive is specified, the default route will be added to the kernel and will NOT be overwritten by any other default route broadcast. The default route will NOT be propagated when the passive option is used.

gateway should be an address in dot notation. proto should be either rip, egp, or hello. The proto field initializes the protocol by which the route was learned. Although in this case it is unused, but the field is remains for consistency.

net netaddr gateway addr metric hopcnt [rip | egp | hello]

host hostaddr gateway addr metric hopcnt [rip | egp | hello]
The following clauses install a static route to net netaddr or host hostaddr through gateway addr at a metric of hopcnt learned via either RIP, HELLO, or EGP. As usual, dot notation is recommended for the addresses. This route will be installed in the kernel's routing table and will never be affected by any other gateway's RIP or HELLO announcements. The protocol by which it was learned is important if the route is to be announced via EGP. If the protocol is rip or hello and there are no routing restrictions, then this route will be announced by EGP with a metric of defaultegpmetric. If the protocol is egp and there are no routing restrictions, then this route will be announced by EGP with a metric of hopcnt.

egpnetreachable net [net] [net].....

This option was left in as a soft restriction. It cannot be used when the announce or noannounce clauses are used. Normally, with no restrictions, gated announces all routes learned from RIP and HELLO via EGP. The egpnetreachable clause restricts EGP announcement to those nets listed in the clause. The metric used for the HELLO and RIP learned routes is the value given in the defaultegpmetric clause. If this clause does not specify a value, the value is set to 255. With the egpnetreachable clause, individual unique EGP metrics may not be set for each net. The defaultegpmetric is used for all networks except those that are directly connected, which use a metric of 0.

martiannets net [net] [net]...

This clause appends to gated's list of martian networks. martian networks are those known to be invalid and should be ignored. When gated hears about one of these networks through any means, it will immediately ignore it. If external tracing is enabled, a message will be printed to the trace log. Multiple occurrences of the martiannets clause accumulate.

A initial list of martian networks is coded into gated in the include file rt_control.h. This list contains 127.0.0.0, 128.0.0.0, 191.253.0.0, 192.0.0.0, 223.255.255.0, and 224.0.0.0.

NOTES ON CONFIGURATION OPTIONS

* gated stores its Process ID in the file /etc/gated.pid. *

If EGP is being used when supplying the default route (via RIP gateway or HELLO gateway and all EGP neighbors are lost, the default route will not be advertised until at least one EGP neighbor is regained.

If routing restrictions are used, gated logs all invalid networks using syslog at log level LOG_WARNING and facility LOG_DAEMON. The facility may be changed at compile time by use of the LOG_FACILITY define.

With the complexity of the current network topology and with many back door paths to networks, the use of routing restrictions is recommended. With the current routing strategies, it is easy for illegal or invalid networks to penetrate into the ARPAnet Core or the NSFNet backbone. Using routing restrictions does take a little more maintenance time and routing restrictions are not the long term answer, but for now, in order to be good internet players, we must use them.

GATED INTERNAL METRICS

* gated stores all metrics internally as a HELLO time delay ranging from 0 to 30000. Received RIP metrics are translated to and from these internal time delays with the use of the following translation tables:
### Time Delay RIP metric

<table>
<thead>
<tr>
<th>Time Delay</th>
<th>RIP metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>148</td>
</tr>
<tr>
<td>3</td>
<td>219</td>
</tr>
<tr>
<td>4</td>
<td>325</td>
</tr>
<tr>
<td>5</td>
<td>481</td>
</tr>
<tr>
<td>6</td>
<td>713</td>
</tr>
<tr>
<td>7</td>
<td>1057</td>
</tr>
<tr>
<td>8</td>
<td>1567</td>
</tr>
<tr>
<td>9</td>
<td>2322</td>
</tr>
<tr>
<td>10</td>
<td>3440</td>
</tr>
<tr>
<td>11</td>
<td>5097</td>
</tr>
<tr>
<td>12</td>
<td>7552</td>
</tr>
<tr>
<td>13</td>
<td>11190</td>
</tr>
<tr>
<td>14</td>
<td>16579</td>
</tr>
<tr>
<td>15</td>
<td>24564</td>
</tr>
<tr>
<td>16</td>
<td>30000</td>
</tr>
</tbody>
</table>

### NOTES ON IMPLEMENTATION SPECIFICS

*gate* stores all metrics internally in milliseconds. The RIP metric is mapped to a millisecond based metric and processed. This will preserve the granularity of the HELLO protocol time delay.

In the *gate* configuration file, all references to POINT-TO-POINT interfaces must use the DESTINATION address. This is the only change made to the configuration file syntax from earlier versions, which used the source address of the PTP link. Otherwise, old configuration files should be compatible.

All protocols have a two minute hold down. When a routing update indicates that the route in use is being deleted, *gate* will not delete the route for two minutes.

Changes can be made to the interfaces and *gate* will notice them without having to restart the process. If the, netmask, subnetmask, broadcast address, or interface metric are changed, the interface should be marked down with *ifconfig(8C)*, then marked up at least thirty seconds later. Flag changes do not require the interface to be brought down and back up.
RIP propagates and listens to host routes. This was done to handle FTP links more consistently. This version also supports the RIP_TRACE commands.

Subnet interfaces are supported. Subnet information will only be propagated on interfaces to other subnets of the same network. For example, if there is a gateway between two class B networks, the subnet routes for each respective class B net are not propagated into the other class B net. Just the class B network number is propagated.

`gated` listens to host and network REDIRECTs and tries to take an action on the REDIRECT for its own internal tables that parallels the kernel’s action. In this way, the redirect routine in `gated` parallels the Berkeley kernel redirect routine as closely as possible. Unlike the Berkeley kernel, `gated` times out routes learned via a REDIRECT after six minutes. The route is then deleted from the kernel routing tables. This helps keep the routing tables more consistent. Any route that was learned via a REDIRECT is NOT announced by any routing protocol.

The `gated` EGP code verifies that all nets sent and received are valid class A, B, or C networks per the EGP specification. Information about networks that do not meet these criteria is not propagated. If an EGP update packet contains information about a network that is not either class A, B or C, the update is considered to be in error and is ignored. Only the information about the specific network will be ignored if `gated` is compiled with the EGP_IGNORE_BAD define specified. This option should be used with caution.

**FILES**

/`etc/gated.conf` The configuration file.
/`etc/gated.pid` The process ID is stored here.
/`usr/tmp/gated_dump` The memory dump file.
/`etc/gated` The GATED process itself.

**SEE ALSO**

routed(8C)
RFC827 (EGP formal specs.), RFC891 (HELLO formal specs.), RFC911 (EGP under UNIX)
The conf directory in the distribution package for sample configuration files.

**AUTHORS**

Mark Fedor (1986-1987)
fedor@nisc.nyser.net

Jeffrey C. Honig
Cornell Theory Center
265 Olin Hall
Cornell University
Ithaca, NY 14853-5201
607/255-8686
jch@tcgould.techn.cornell.edu
{ucbvax,unet,decvax)!tcgould.techn.cornell.edu|jch

**CREDITS**

This program was derived from Paul Kirton’s EGP for UNIX, UC at Berkeley’s routed(8C), and HELLO routines by Mike Petry at the University of Maryland.

Special thanks to Craig Partridge, craig@nnsn.nsf.net, for linting, profiling and performance enhancements.
Also, thanks to all the BETA-TESTERS and the NSFNET backbone crew who put up with occasional nasty conditions brought on by the development of this program.
NAME
halt – stop the processor

SYNOPSIS
/etc/halt [-n] [-q]

DESCRIPTION
halt writes out cached disk information and then enters the PROM monitor. The machine does not reboot, even if the auto-reboot switch is set.

halt without the -n or -q option is equivalent to
  # telinit 0

The -q and -n options cause a quick halt, no graceful shutdown is attempted.
With either the -n or -q option specified, halt is equivalent to
  # /etc/uadmin 2 0

SEE ALSO
init(1M), reboot(8), shutdown(8), telinit(1M), uadmin(1M)
NAME

ifconfig – configure network interface parameters

SYNOPSIS

/etc/ifconfig interface [ address_family ] [ address [ dest_address ] ] [ parameters ]
/etc/ifconfig interface [ protocol_family ]

DESCRIPTION

Ifconfig is used to assign an address to a network interface and/or configure network
interface parameters. Ifconfig must be used at boot time to define the network
address of each interface present on a machine; it may also be used at a later time to
redefine an interface’s address or other operating parameters. The interface parameter
is a string of the form “name unit”, e.g. “en0”.

An interface may receive transmissions in differing protocols, each of which may
require separate naming schemes. If the address family of the specified address is not
“inet”, it is necessary to specify address_family, which changes the interpretation of
the remaining parameters. The “inet” address family may be optionally specified,
since it is the default. Currently, “ns” is the only other option for address_family.

For the DARPA-Internet family, the address is either a host name present in the host
name data base, hosts(5), or a DARPA Internet address expressed in the Internet stand-
dard “dot notation”. For the Xerox Network Systems(tm) family, addresses are
net:a.b.c.d.e.f, where net is the assigned network number (in decimal), and each of the
six bytes of the host number, a through f, are specified in hexadecimal. The host
number may be omitted on 10Mb/s Ethernet interfaces, which use the hardware
physical address, and on interfaces other than the first.

The following parameters may be set with ifconfig:

up Mark an interface “up”. This may be used to enable an interface
after an “ifconfig down.” It happens automatically when setting the
first address on an interface. If the interface was reset when previ-
ously marked down, the hardware will be re-initialized.

down Mark an interface “down”. When an interface is marked “down”,
the system will not attempt to transmit messages through that in-
terface. If possible, the interface will be reset to disable reception as
well. This action does not automatically disable routes using the
interface.

trailers Request the use of a “trailer” link level encapsulation when sending
(default). If a network interface supports trailers, the system will,
when possible, encapsulate outgoing messages in a manner which
minimizes the number of memory to memory copy operations
performed by the receiver. On networks that support the Address
Resolution Protocol (see arp(4P); currently, only 10 Mb/s Ethernet),
this flag indicates that the system should request that other systems
use trailers when sending to this host. Similarly, trailer encapsu-
lations will be sent to other hosts that have made such requests.
Currently used by Internet protocols only.

-trailers Disable the use of a “trailer” link level encapsulation.

arp Enable the use of the Address Resolution Protocol in mapping
between network level addresses and link level addresses (default).
This is currently implemented for mapping between DARPA Inter-
net addresses and 10Mb/s Ethernet addresses.
-arp

Disable the use of the Address Resolution Protocol.

metric \( n \)

Set the routing metric of the interface to \( n \), default 0. The routing metric is used by the routing protocol \((routed(8c))\). Higher metrics have the effect of making a route less favorable; metrics are counted as addition hops to the destination network or host.

debug

Enable driver dependent debugging code; usually, this turns on extra console error logging.

-debug

Disable driver dependent debugging code.

netmask \( mask \)

(Inet only) Specify how much of the address to reserve for subdividing networks into sub-networks. The mask includes the network part of the local address and the subnet part, which is taken from the host field of the address. The mask can be specified as a single hexadecimal number with a leading \( 0x \), with a dot-notation Internet address, or with a pseudo-network name listed in the network table \( networks(5) \). The mask contains 1's for the bit positions in the 32-bit address which are to be used for the network and subnet parts, and 0's for the host part. The mask should contain at least the standard network portion, and the subnet field should be contiguous with the network portion.

dstaddr

Specify the address of the correspondent on the other end of a point to point link.

broadcast

(Inet only) Specify the address to use to represent broadcasts to the network. The default broadcast address is the address with a host part of all 1's.

ipdst

(NS only) This is used to specify an Internet host who is willing to receive ip packets encapsulating NS packets bound for a remote network. In this case, an apparent point to point link is constructed, and the address specified will be taken as the NS address and network of the destinee.

Ifconfig displays the current configuration for a network interface when no optional parameters are supplied. If a protocol family is specified, Ifconfig will report only the details specific to that protocol family.

Only the super-user may modify the configuration of a network interface.

**DIAGNOSTICS**

Messages indicating the specified interface does not exit, the requested address is unknown, or the user is not privileged and tried to alter an interface’s configuration.

**SEE ALSO**

netstat(1), intro(4N)
NAME

ined - internet "super-server"

SYNOPSIS

/etc/inetd [-d] [ configuration file ]

DESCRIPTION

ined should be run at boot time by /etc/rc2.d/S30inetd. It then listens for connections on
certain internet sockets. When a connection is found on one of its sockets, it decides
what service the socket corresponds to, and invokes a program to service the request.
After the program is finished, it continues to listen on the socket (except in some
cases which will be described below). Essentially, inetd allows running one daemon
to invoke several others, reducing load on the system.

Upon execution, inetd reads its configuration information from a configuration file
which, by default, is /etc/inetd.conf. There must be an entry for each field of the
configuration file, with entries for each field separated by a tab or a space. Com-
ments are denoted by a "#" at the beginning of a line. There must be an entry for
each field, except as noted below. The fields of the configuration file are as follows:

- service name
- socket type
- protocol
- wait/nowait
- user
- server program
- server program arguments

The service name entry is the name of a valid service in the file /etc/services. For "in-
ternal" services (discussed below), the service name must be the official name of the ser-
vice (that is, the first entry in /etc/services). For RPC services, the value of the service
name field consists of the RPC service name, followed by a slash (/) and either a ver-
nion number or a range of version numbers (for example, mountd/1).

The socket type should be one of "stream", "dgram", "raw", "rdm", or "seqpacket",
depending on whether the socket is a stream, datagram, raw, reliably delivered mes-
sage, or sequenced packet socket.

The protocol must be a valid protocol as given in /etc/protocols. Examples might be
"tcp" or "udp". For RPC services, the field consists of the string "rpc" followed by a
slash (/) and the name of the protocol (for example, rpc/udp).

The wait/nowait entry is applicable to datagram sockets only (other sockets should
have a "nowait" entry in this space). If a datagram server connects to its peer, free-
ing the socket so inetd can received further messages on the socket, it is said to be a
"multi-threaded" server, and should use the "nowait" entry. For datagram servers
which process all incoming datagrams on a socket and eventually time out, the
server is said to be "single-threaded" and should use a "wait" entry. "Comsat"
("biff") and "talk" are both examples of the latter type of datagram server. Tftp is
an exception; it is a datagram server that establishes pseudo-connections. It must be
listed as "wait" in order to avoid a race; the server reads the first packet, creates a
new socket, and then forks and exits to allow inetd to check for new service requests
to spawn new servers.

The user entry should contain the user name of the user as whom the server should
run. This allows for servers to be given less permission than root. The server program
entry should contain the pathname of the program which is to be executed by inetd
when a request is found on its socket. If inetd provides this service internally, this
entry should be "internal".
The arguments to the server program should be just as they normally are, starting with argv[0], which is the name of the program. If the service is provided internally, the word “internal” should take the place of this entry.

**Inetd** provides several “trivial” services internally by use of routines within itself. These services are “echo”, “discard”, “chargen” (character generator), “daytime” (human readable time), and “time” (machine readable time, in the form of the number of seconds since midnight, January 1, 1900). All of these services are tcp based. For details of these services, consult the appropriate RFC from the Network Information Center.

**Inetd** rereads its configuration file when it receives a hangup signal, SIGHUP. Services may be added, deleted or modified when the configuration file is reread.

**FILES**

/etc/inetd.conf
/etc/services
/etc/protocols

**SEE ALSO**

ftpd(8C), rlogind(8C), rshd(8C), telnetd(8C), tftp(8C)
NAME

makeindex – index manpages allowing ‘man very_long_name’ to work

SYNOPSIS

/usr/lib/makeindex file ...

DESCRIPTION

makeindex builds a database of man page file names and man page names. The man
command uses the data base to find the file name for a particular man page.

For every file in the argument list, makeindex writes one or more lines on stdout con-
taining the file name, and a list of manpage names. Man page Index files are created
by:

for i in 1 2 3 4 5 6 7 8
    cd /usr/man/man$i
    /usr/lib/makeindex * > ../cat$i/Index
done

for i in 1 2 3
    cd /usr/man/bsd/man$i
    /usr/lib/makeindex * > ../cat$i/Index
done

FILE FORMAT

To allow makeindex to locate the name, the source file format of a man-page MUST
be consistent with the following description.

The FIRST occurrence of a line that begins with .SH must be the line that defines the
NAME of a function, that is:

.SH NAME

The lines that follow the name will all be scanned to locate possible entries for the
index file. The format of these lines can be any one of the following:

    function \- oneline description
    function1, function2, function_N \- oneline description
    string: strcat, strcmp, strncmp \- string functions

The next line that begins with the characters .SH stops the scanning for possible
index entries, thus allowing multiple lines having the same format to create multiple
entries.

An index entry, listing the file name and the function name will be created for each of
the names listed above. If the manpage named ‘samplepage’ contained exactly the
entries shown in the example above, then an ‘Index’ file would contain the following
entries:

    samplepage:function
    samplepage:function1
    samplepage:function2
    samplepage:function_N
    samplepage:strcat
    samplepage:strcmp
    samplepage:string
    samplepage:strncmp
In other words, names may be either colon delimited or comma delimited in a one-line description that contains a `\` preceding the actual description of the functions. Everything must be in that single-line description to allow indexing to function correctly.

**WHAT TO AVOID**

The following layout will *not* function correctly:

```
.SH NAME
function1, function2, function3  <newline character>
\- description on this following line.
.SH SYNOPSIS
```

No entries will be created with the above layout.

```
.SH NAME
function1, function2, function3  <newline character>
function4, function5 \- description of these functions
.SH SYNOPSIS
```

This will result in only function4 and function5 being indexed because there is no backslash-hyphen used.

**WHAT IS PERMITTED**

The following layout is permitted:

```
.SH NAME
  generalname_if_any: func1, func2, func3, func4, func5 \-
  description on this next line
.SH SYNOPSIS
```

In other words, the elements that cause an entry to be created for a line are the function names (comma or colon delimited), and the backslash-hyphen. Any lines not containing these elements will be ignored.

**SEE ALSO**

catman(8), man(1)
NAME

named – Internet domain name server

SYNOPSIS

named [-d debuglevel ] [-p port# ] [(-b) bootfile ]

DESCRIPTION

named is the Internet domain name server. See RFC883 for more information on the
Internet name-domain system. Without any arguments, named will read the default
boot file /etc/named.boot, read any initial data and listen for queries.
Options are:

-d  Print debugging information. A number after the “d” determines the level of
messages printed.

-p  Use a different port number. The default is the standard port number as listed
in /etc/services.

-b  Use an alternate boot file. This is optional and allows you to specify a file with
a leading dash.

Any additional argument is taken as the name of the boot file. The boot file contains
information about where the name server is to get its initial data. If multiple boot
files are specified, only the last is used. Lines in the boot file cannot be continued on
subsequent lines. The following is a small example:

; ;     boot file for name server
; directory /usr/local/domain
;
;   type       domain     source host/file    backup file

  cache .          root.cache
  primary Berkeley.EDU      berkeley.edu.zone
  primary 32.128.IN-ADDR.ARPA  uchosts.rev
  secondary CC.Berkeley.EDU 128.32.137.8 128.32.137.3 cc.zone.bak
  secondary 6.32.128.IN-ADDR.ARPA 128.32.137.8 128.32.137.3 cc.rev.bak
  primary 0.0.127.IN-ADDR.ARPA  localhost.rev
  forwarders 10.0.0.78 10.2.0.78

; slave

The “directory” line causes the server to change its working directory to the directory specified. This can be important for the correct processing of $INCLUDE files in primary zone files.

The “cache” line specifies that data in “root.cache” is to be placed in the backup cache. Its main use is to specify data such as locations of root domain servers. This cache is not used during normal operation, but is used as “hints” to find the current root servers. The file “root.cache” is in the same format as “berkeley.edu.zone”. There can be more than one “cache” file specified. The cache files are processed in such a way as to preserve the time-to-live’s of data dumped out. Data for the root nameservers is kept artificially valid if necessary.

The first “primary” line states that the file “berkeley.edu.zone” contains authoritative data for the “Berkeley.EDU” zone. The file “berkeley.edu.zone” contains data in the master file format described in RFC883. All domain names are relative to the origin, in this case, “Berkeley.EDU” (see below for a more detailed description). The
second "primary" line states that the file "ucbhosts.rev" contains authoritative data for the domain "32.128.IN-ADDR.ARPA," which is used to translate addresses in network 128.32 to hostnames. Each master file should begin with an SOA record for the zone (see below).

The first "secondary" line specifies that all authoritative data under "CC.Berkeley.EDU" is to be transferred from the name server at 128.32.137.8. If the transfer fails it will try 128.32.137.3 and continue trying the addresses, up to 10, listed on this line. The secondary copy is also authoritative for the specified domain. The first non-dotted-quad address on this line will be taken as a filename in which to backup the transferred zone. The name server will load the zone from this backup file if it exists when it boots, providing a complete copy even if the master servers are unreachable. Whenever a new copy of the domain is received by automatic zone transfer from one of the master servers, this file will be updated. The second "secondary" line states that the address-to-hostname mapping for the subnet 128.32.136 should be obtained from the same list of master servers as the previous zone.

The "forwarders" line specifies the addresses of sitewide servers that will accept recursive queries from other servers. If the boot file specifies one or more forwarding servers, then the server will send all queries for data not in the cache to the forwarding first. Each forwarder will be asked in turn until an answer is returned or the list is exhausted. If no answer is forthcoming from a forwarder, the server will continue as it would have without the forwarders line unless it is in "slave" mode. The forwarding facility is useful to cause a large sitewide cache to be generated on a master, and to reduce traffic over links to outside servers. It can also be used to allow servers to run that do not have access directly to the Internet, but wish to act as though they do.

The "slave" line (shown commented out) is used to put the server in slave mode. In this mode, the server will only make queries to forwarders. This option is normally used on machine that wish to run a server but for physical or administrative reasons cannot be given access to the Internet, but have access to a host that does have access.

The "sortlist" line can be used to indicate networks that are to be preferred over other, unlisted networks. Queries for host addresses from hosts on the same network as the server will receive responses with local network addresses listed first, then addresses on the sort list, then other addresses. This line is only acted on at initial startup. When reloading the nameserver with a SIGHUP, this line will be ignored.

The master file consists of control information and a list of resource records for objects in the zone of the forms:

```$INCLUDE <filename> <opt_domain>
$ORIGIN <domain>
<domain> <opt_ttl> <opt_class> <type> <resource_record_data>
```

where domain is "." for root, "@" for the current origin, or a standard domain name. If domain is a standard domain name that does not end with "." the current origin is appended to the domain. Domain names ending with "." are unmodified. The opt_domain field is used to define an origin for the data in an included file. It is equivalent to placing a $ORIGIN statement before the first line of the included file. The field is optional. Neither the opt_domain field nor $ORIGIN statements in the included file modify the current origin for this file. The opt_ttl field is an optional integer number for the time-to-live field. It defaults to zero, meaning the minimum value specified in the SOA record for the zone. The opt_class field is the object address type; currently only one type is supported, IN, for objects connected to the DARPA Internet. The type field contains one of the following tokens; the data expected in the resource_record_data field is in parentheses.
A a host address (dotted quad)
NS an authoritative name server (domain)
MX a mail exchanger (domain)
CNAME the canonical name for an alias (domain)
SOA marks the start of a zone of authority (domain of originating host, domain address of maintainer, a serial number and the following parameters in seconds: refresh, retry, expire and minimum TTL (see RFC883))
MB a mailbox domain name (domain)
MG a mail group member (domain)
MR a mail rename domain name (domain)
NULL a null resource record (no format or data)
WKS a well know service description (not implemented yet)
PTR a domain name pointer (domain)
HINFO host information (cpu_type OS_type)
MINfo mailbox or mail list information (request_domain error_domain)

Resource records normally end at the end of a line, but may be continued across lines between opening and closing parentheses. Comments are introduced by semicolons and continue to the end of the line.

Each master zone file should begin with an SOA record for the zone. An example SOA record is as follows:

```plaintext
@   IN   SOA   ucbvax.Berkeley.EDU. rwh.ucbvax.Berkeley.EDU. ( 2.89 ; serial 10800 ; refresh 3600 ; retry 3600000 ; expire 86400 ) ; minimum
```

The SOA lists a serial number, which should be changed each time the master file is changed. Secondary servers check the serial number at intervals specified by the refresh time in seconds; if the serial number changes, a zone transfer will be done to load the new data. If a master server cannot be contacted when a refresh is due, the retry time specifies the interval at which refreshes should be attempted until successful. If a master server cannot be contacted within the interval given by the expire time, all data from the zone is discarded by secondary servers. The minimum value is the time-to-live used by records in the file with no explicit time-to-live value.

### NOTES

The boot file directives "domain" and "suffixes" have been obsoleted by a more useful resolver based implementation of suffixing for partially qualified domain names. The prior mechanisms could fail under a number of situations, especially when then local nameserver did not have complete information.

The following signals have the specified effect when sent to the server process using the `kill(1)` command.

**SIGHUP**

Causes server to read named.boot and reload database.
SIGINT
Dumps current data base and cache to /usr/tmp/named_dump.db

SIGIO
Dumps statistics data into /usr/tmp/named.stats if the server is compiled
-DSTATS. Statistics data is appended to the file.

SIGSYS
Dumps the profiling data in /usr/tmp if the server is compiled with profiling
(server forks, chdirs and exits).

SIGTERM
Dumps the primary and secondary database files. Used to save modified data
on shutdown if the server is compiled with dynamic updating enabled.

SIGUSR1
Turns on debugging; each SIGUSR1 increments debug level. (SIGEMT on older
systems without SIGUSR1)

SIGUSR2
Turns off debugging completely. (SIGFPE on older systems without SIGUSR2)

FILES
/etc/named.boot name server configuration boot file
/etc/named.pid the process id
/usr/tmp/named.run debug output
/usr/tmp/named_dump.db dump of the name server database
/usr/tmp/named.stats nameserver statistics data

SEE ALSO
kill(1), gethostname(3N), signal(3c), resolver(3), resolver(5), hostname(7), RFC882,
RFC883, RFC973, RFC974, Name Server Operations Guide for BIND
NAME

ping – send ICMP ECHO_REQUEST packets to network hosts

SYNOPSIS

/etc/ping [ -r ] [ -v ] host [ packetsize ] [ count ]

DESCRIPTION

The DARPA Internet is a large and complex aggregation of network hardware, connected together by gateways. Tracking a single-point hardware or software failure can often be difficult. ping utilizes the ICMP protocol’s mandatory ECHO_REQUEST datagram to elicit an ICMP ECHO_RESPONSE from a host or gateway. ECHO_REQUEST datagrams (“pings”) have an IP and ICMP header, followed by a struct timeval, and then an arbitrary number of “pad” bytes used to fill out the packet. Default datagram length is 64 bytes, but this may be changed using the command-line option. Other options are:

- r Bypass the normal routing tables and send directly to a host on an attached network. If the host is not on a directly-attached network, an error is returned. This option can be used to ping a local host through an interface that has no route through it (e.g., after the interface was dropped by routed(8C)).

- v Verbose output. ICMP packets other than ECHO RESPONSE that are received are listed.

When using ping for fault isolation, it should first be run on the local host, to verify that the local network interface is up and running. Then, hosts and gateways further and further away should be “pinged”. ping sends one datagram per second, and prints one line of output for every ECHO_RESPONSE returned. No output is produced if there is no response. If an optional count is given, only that number of requests is sent. Round-trip times and packet loss statistics are computed. When all responses have been received or the program times out (with a count specified), or if the program is terminated with a SIGINT, a brief summary is displayed.

This program is intended for use in network testing, measurement and management. It should be used primarily for manual fault isolation. Because of the load it could impose on the network, it is unwise to use ping during normal operations or from automated scripts.

AUTHOR

Mike Muuss
NAME
reboot - reboot /unix

SYNOPSIS
/etc/reboot [-n] [-q]

DESCRIPTION
reboot writes out cached disk information and then requests the PROM monitor to
reboot /unix. reboot without the -q or -n option is equivalent to
   # telinit 0
The -q and -n options cause a quick reboot, no graceful shutdown is attempted. With either
the -q or -n option specified, reboot is equivalent to
   # /etc/uadmin 2 2

SEE ALSO
halt(8), init(1M), shutdown(8), telinit(8), uadmin(1M)
NAME
rlogind – remote login server

SYNOPSIS
/etc/rlogind [ -d ]

DESCRIPTION
rlogind is the server for the rlogin(1C) program. The server provides a remote login
facility with authentication based on privileged port numbers from trusted hosts.

rlogind listens for service requests at the port indicated in the "login" service
specification; see services(5). When a service request is received the following protocol
is initiated:

1) The server checks the client’s source port. If the port is not in the range 0-1023,
   the server aborts the connection.

2) The server checks the client’s source address and requests the corresponding
   host name (see gethostbyaddr(3N), hosts(5) and named(8)). If the hostname cannot
   be determined, the dot-notation representation of the host address is used.

Once the source port and address have been checked, rlogind allocates a pseudo ter-
mital (see pty(4)), and manipulates file descriptors so that the slave half of the
pseudo terminal becomes the stdin, stdout, and stderr for a login process. The login
process is an instance of the login(1) program, invoked with the -r option. The login
process then proceeds with the authentication process as described in rshd(8C), but if
automatic authentication fails, it reprompts the user to login as one finds on a stan-
dard terminal line.

The parent of the login process manipulates the master side of the pseudo terminal,
operating as an intermediary between the login process and the client instance of the
rlogin program. In normal operation, the packet protocol described in pty(4) is
invoked to provide ^S/^Q type facilities and propagate interrupt signals to the
remote programs. The login process propagates the client terminal’s baud rate and
terminal type, as found in the environment variable, “TERM”; see environ(7). The
screen or window size of the terminal is requested from the client, and window size
changes from the client are propagated to the pseudo terminal.

DIAGNOSTICS
All diagnostic messages are returned on the connection associated with the stderr,
after which any network connections are closed. An error is indicated by a leading
byte with a value of 1.

"Try again."
A fork by the server failed.

"/bin/sh: ...
"The user’s login shell could not be started.

BUGS
The authentication procedure used here assumes the integrity of each client machine
and the connecting medium. This is insecure, but is useful in an “open” environ-
ment.

A facility to allow all data exchanges to be encrypted should be present.
A more extensible protocol should be used.
NAME
route - manually manipulate the routing tables

SYNOPSIS
/etc/route [-f] [-n] [command args]

DESCRIPTION
Route is a program used to manually manipulate the network routing tables. It normally is not needed, as the system routing table management daemon, routed(8C), should tend to this task.

Route accepts two commands: add, to add a route, and delete, to delete a route.

All commands have the following syntax:
/etc/route command [ net | host ] destination gateway [ metric ]

where destination is the destination host or network, gateway is the next-hop gateway to which packets should be addressed, and metric is a count indicating the number of hops to the destination. The metric is required for add commands; it must be zero if the destination is on a directly-attached network, and nonzero if the route utilizes one or more gateways. If adding a route with metric 0, the gateway given is the address of this host on the common network, indicating the interface to be used for transmission. Routes to a particular host are distinguished from those to a network by interpreting the Internet address associated with destination. The optional keywords net and host force the destination to be interpreted as a network or a host, respectively. Otherwise, if the destination has a "local address part" of INADDR_ANY, or if the destination is the symbolic name of a network, then the route is assumed to be to a network; otherwise, it is presumed to be a route to a host. If the route is to a destination connected via a gateway, the metric should be greater than 0. All symbolic names specified for a destination or gateway are looked up first as a host name using gethostbyname(3N). If this lookup fails, getnetbyname(3N) is then used to interpret the name as that of a network.

Route uses a raw socket and the SIOCADDRT and SIOCDELRT ioctl's to do its work. As such, only the super-user may modify the routing tables.

If the -f option is specified, route will "flush" the routing tables of all gateway entries. If this is used in conjunction with one of the commands described above, the tables are flushed prior to the command's application.

The -n option prevents attempts to print host and network names symbolically when reporting actions.

DIAGNOSTICS
"add [ host | network ] %s: gateway %s flags %x"
The specified route is being added to the tables. The values printed are from the routing table entry supplied in the ioctl call. If the gateway address used was not the primary address of the gateway (the first one returned by gethostbyname), the gateway address is printed numerically as well as symbolically.

"delete [ host | network ] %s: gateway %s flags %x"
As above, but when deleting an entry.

"%s %s done"
When the -f flag is specified, each routing table entry deleted is indicated with a message of this form.

"Network is unreachable"
An attempt to add a route failed because the gateway listed was not on a directly-connected network. The next-hop gateway must be given.
"not in table"
A delete operation was attempted for an entry which wasn’t present in the tables.

"routing table overflow"
An add operation was attempted, but the system was low on resources and was unable to allocate memory to create the new entry.

SEE ALSO
intro(4N), routed(8C)
NAME
routed — network routing daemon

SYNOPSIS
/etc/routed [-d] [-g] [-s] [-q] [-t] logfile

DESCRIPTION
Routed is invoked at boot time to manage the network routing tables. The routing
daemon uses a variant of the Xerox NS Routing Information Protocol in maintaining
up to date kernel routing table entries. It used a generalized protocol capable of use
with multiple address types, but is currently used only for Internet routing within a
cluster of networks.

In normal operation routed listens on the udph(4P) socket for the route service (see ser-
vices(5)) for routing information packets. If the host is an internetwork router, it
periodically supplies copies of its routing tables to any directly connected hosts and
networks.

When routed is started, it uses the SIOCGIFCONF ioctl to find those directly con-
ected interfaces configured into the system and marked "up" (the software loop-
back interface is ignored). If multiple interfaces are present, it is assumed that the
host will forward packets between networks. Routed then transmits a request packet
on each interface (using a broadcast packet if the interface supports it) and enters a
loop, listening for request and response packets from other hosts.

When a request packet is received, routed formulates a reply based on the information
maintained in its internal tables. The response packet generated contains a list of
known routes, each marked with a "hop count" metric (a count of 16, or greater, is
considered "infinite"). The metric associated with each route returned provides a
metric relative to the sender.

Response packets received by routed are used to update the routing tables if one of the
following conditions is satisfied:

1. No routing table entry exists for the destination network or host, and the metric
   indicates the destination is "reachable" (i.e. the hop count is not infinite).

2. The source host of the packet is the same as the router in the existing routing
table entry. That is, updated information is being received from the very inter-
   network router through which packets for the destination are being routed.

3. The existing entry in the routing table has not been updated for some time
   (defined to be 90 seconds) and the route is at least as cost effective as the
current route.

4. The new route describes a shorter route to the destination than the one
currently stored in the routing tables; the metric of the new route is compared
against the one stored in the table to decide this.

When an update is applied, routed records the change in its internal tables and
updates the kernel routing table. The change is reflected in the next response packet
sent.

In addition to processing incoming packets, routed also periodically checks the rout-
ing table entries. If an entry has not been updated for 3 minutes, the entry's metric is
set to infinity and marked for deletion. Deletions are delayed an additional 60
seconds to insure the invalidation is propagated throughout the local internet.

Hosts acting as internetwork routers gratuitously supply their routing tables every
30 seconds to all directly connected hosts and networks. The response is sent to the
broadcast address on nets capable of that function, to the destination address on
point-to-point links, and to the router's own address on other networks. The normal
routing tables are bypassed when sending gratuitous responses. The reception of
responses on each network is used to determine that the network and interface are
functioning correctly. If no response is received on an interface, another route may
be chosen to route around the interface, or the route may be dropped if no alternative
is available.

Routed supports several options:

-d Enable additional debugging information to be logged, such as bad packets
received.

-g This flag is used on internetwork routers to offer a route to the “default” desti-
nation. This is typically used on a gateway to the Internet, or on a gateway that
uses another routing protocol whose routes are not reported to other local
routers.

-s Supplying this option forces routed to supply routing information whether it is
acting as an internetwork router or not. This is the default if multiple network
interfaces are present, or if a point-to-point link is in use.

-q This is the opposite of the -s option.

-t If the -t option is specified, all packets sent or received are printed on the stan-
standard output. In addition, routed will not divorce itself from the controlling ter-
minal so that interrupts from the keyboard will kill the process.

Any other argument supplied is interpreted as the name of file in which routed’s
actions should be logged. This log contains information about any changes to the
routing tables and, if not tracing all packets, a history of recent messages sent and
received which are related to the changed route.

In addition to the facilities described above, routed supports the notion of “distant”
passive and active gateways. When routed is started up, it reads the file /etc/gateways to
find gateways which may not be located using only information from the SIGIFCONF ioctl.
Gateways specified in this manner should be marked passive if they are
not expected to exchange routing information, while gateways marked active
should be willing to exchange routing information (i.e. they should have a routed
process running on the machine). Passive gateways are maintained in the routing
tables forever and information regarding their existence is included in any routing
information transmitted. Active gateways are treated equally to network interfaces.
Routing information is distributed to the gateway and if no routing information is
received for a period of the time, the associated route is deleted. External gateways
are also passive, but are not placed in the kernel routing table nor are they included
in routing updates. The function of external entries is to inform routed that another
routing process will install such a route, and that alternate routes to that destination
should not be installed. Such entries are only required when both routers may learn
of routes to the same destination.

The /etc/gateways is comprised of a series of lines, each in the following format:
< net | host > name1 gateway name2 metric value < passive | active | external >
The net or host keyword indicates if the route is to a network or specific host.
Name1 is the name of the destination network or host. This may be a symbolic name
located in /etc/networks or /etc/hosts (or, if started after named(8), known to the name
server), or an Internet address specified in “dot” notation; see inet(3N).
Name2 is the name or address of the gateway to which messages should be for-
warded.
Value is a metric indicating the hop count to the destination host or network.

One of the keywords passive, active or external indicates if the gateway should be treated as passive or active (as described above), or whether the gateway is external to the scope of the routed protocol.

Internetwork routers that are directly attached to the Arpanet or Milnet should use the Exterior Gateway Protocol (EGP) to gather routing information rather then using a static routing table of passive gateways. EGP is required in order to provide routes for local networks to the rest of the Internet system. Sites needing assistance with such configurations should contact the Computer Systems Research Group at Berkeley.

FILES

/etc/gateways  for distant gateways

SEE ALSO

"Internet Transport Protocols", XSID 028112, Xerox System Integration Standard.

BUGS

The kernel's routing tables may not correspond to those of routed when redirects change or add routes. The only remedy for this is to place the routing process in the kernel.

Routed should incorporate other routing protocols, such as Xerox NS (XNSrouted(8C)) and EGP. Using separate processes for each requires configuration options to avoid redundant or competing routes.

Routed should listen to intelligent interfaces, such as an IMP, and to error protocols, such as ICMP, to gather more information. It does not always detect unidirectional failures in network interfaces (e.g., when the output side fails).
**NAME**

rshd – remote shell server

**SYNOPSIS**

/etc/rshd

**DESCRIPTION**

*rshd* is the server for the *rcmd* *(3X)* routine and, consequently, for the *rsh*(1C) program. The server provides remote execution facilities with authentication based on privileged port numbers from trusted hosts.

*rshd* listens for service requests at the port indicated in the "cmd" service specification; see *services*(5). When a service request is received the following protocol is initiated:

1) The server checks the client’s source port. If the port is not in the range 0-1023, the server aborts the connection.

2) The server reads characters from the socket up to a null (\'\0\') byte. The resultant string is interpreted as an ASCII number, base 10.

3) If the number received in step 1 is non-zero, it is interpreted as the port number of a secondary stream to be used for the std**err**. A second connection is then created to the specified port on the client’s machine. The source port of this second connection is also in the range 0-1023.

4) The server checks the client’s source address and requests the corresponding host name (see *gethostbyaddr*(3N), *hosts*(5) and *named*(8)). If the hostname cannot be determined, the dot-notation representation of the host address is used.

5) A null terminated user name of at most 16 characters is retrieved on the initial socket. This user name is interpreted as the user identity on the client’s machine.

6) A null terminated user name of at most 16 characters is retrieved on the initial socket. This user name is interpreted as a user identity to use on the server’s machine.

7) A null terminated command to be passed to a shell is retrieved on the initial socket. The length of the command is limited by the upper bound on the size of the system’s argument list.

8) *rshd* then validates the user according to the following steps. The local (server-end) user name is looked up in the password file and a *chdir* is performed to the user’s home directory. If either the lookup or *chdir* fail, the connection is terminated. If the user is not the super-user, (user id 0), the file /etc/*hosts.equiv* is consulted for a list of hosts considered "equivalent". If the client’s host name is present in this file, the authentication is considered successful. If the lookup fails, or the user is the super-user, then the file .rhosts in the home directory of the remote user is checked for the machine name and identity of the user on the client’s machine. If this lookup fails, the connection is terminated.

9) A null byte is returned on the initial socket and the command line is passed to the normal login shell of the user. The shell inherits the network connections established by *rshd*.

**DIAGNOSTICS**

Except for the last one listed below, all diagnostic messages are returned on the initial socket, after which any network connections are closed. An error is indicated by a leading byte with a value of 1 (0 is returned in step 9 above upon successful completion of all the steps prior to the execution of the login shell).
"locuser too long"
The name of the user on the client's machine is longer than 16 characters.

"remuser too long"
The name of the user on the remote machine is longer than 16 characters.

"command too long"
The command line passed exceeds the size of the argument list (as configured into the system).

"Login incorrect."
No password file entry for the user name existed.

"No remote directory."
The chdir command to the home directory failed.

"Permission denied."
The authentication procedure described above failed.

"Can't make pipe."
The pipe needed for the stderr, wasn't created.

"Try again."
A fork by the server failed.

"<shellname>:

The user's login shell could not be started. This message is returned on the connection associated with the stderr, and is not preceded by a flag byte.

SEE ALSO
rsh(1C)

BUGS
The authentication procedure used here assumes the integrity of each client machine and the connecting medium. This is insecure, but is useful in an "open" environment.

A facility to allow all data exchanges to be encrypted should be present.

A more extensible protocol should be used.
NAME
rwhod – system status server

SYNOPSIS
/etc/rwhod

DESCRIPTION
rwhod is the server which maintains the database used by the rwho(1C) and runtime(1C) programs. Its operation is predicated on the ability to broadcast messages on a network.

rwhod operates as both a producer and consumer of status information. As a producer of information it periodically queries the state of the system and constructs status messages which are broadcast on a network. As a consumer of information, it listens for other rwhod servers' status messages, validating them, then recording them in a collection of files located in the directory /usr/spool/rwho.

The server transmits and receives messages at the port indicated in the "rwho" service specification; see services(5). The messages sent and received, are of the form:

```c
struct outmp {
    char  out_line[8]; /* tty name */
    char  out_name[8]; /* user id */
    long  out_time;/* time on */
};

struct whod {
    char  wd vers;
    char  wd_type;
    char  wd_fill[2];
    int   wd_sendtime;
    int   wd_recvtime;
    char  wd_hostname[32];
    int   wd_loadav[3];
    int   wd_boottime;
    struct whoent {
        struct outmp we utmp;
        int   we idle;
    } wd we[1024 / sizeof(struct whoent)];
};
```

All fields are converted to network byte order prior to transmission. The load averages are as calculated by the w(1) program, and represent load averages over the 5, 10, and 15 minute intervals prior to a server's transmission; they are multiplied by 100 for representation in an integer. The host name included is that returned by the gethostname(2) system call, with any trailing domain name omitted. The array at the end of the message contains information about the users logged in to the sending machine. This information includes the contents of the utmp(5) entry for each non-idle terminal line and a value indicating the time in seconds since a character was last received on the terminal line.

Messages received by the rwho server are discarded unless they originated at an rwho server's port. In addition, if the host's name, as specified in the message, contains any unprintable ASCII characters, the message is discarded. Valid messages received by rwhod are placed in files named whod.hostname in the directory /usr/spool/rwho. These files contain only the most recent message, in the format described above.
Status messages are generated approximately once every 3 minutes.

**SEE ALSO**

rwho(1C), ruptime(1C)

**BUGS**

There should be a way to relay status information between networks. Status information should be sent only upon request rather than continuously. People often interpret the server dying or network communication failures as a machine going down.
NAME
sendmail – send mail over the internet

SYNOPSIS
/usr/lib/sendmail [ flags ] [ address ... ]
newaliases
mailq [ –v ]

DESCRIPTION
Sendmail sends a message to one or more recipients, routing the message over whatever networks are necessary. Sendmail does internetwork forwarding as necessary to deliver the message to the correct place.

Sendmail is not intended as a user interface routine; other programs provide user-friendly front ends; sendmail is used only to deliver pre-formatted messages.

With no flags, sendmail reads its standard input up to an end-of-file or a line consisting only of a single dot and sends a copy of the message found there to all of the addresses listed. It determines the network(s) to use based on the syntax and contents of the addresses.

Local addresses are looked up in a file and aliased appropriately. Aliasing can be prevented by preceding the address with a backslash. Normally the sender is not included in any alias expansions, e.g., if ‘john’ sends to ‘group’, and ‘group’ includes ‘john’ in the expansion, then the letter will not be delivered to ‘john’.

Flags are:

–ba Go into ARPANET mode. All input lines must end with a CR-LF, and all messages will be generated with a CR-LF at the end. Also, the “From:” and “Sender:” fields are examined for the name of the sender.

–bd Run as a daemon. This requires Berkeley IPC. Sendmail will fork and run in background listening on socket 25 for incoming SMTP connections. This is normally run from /etc/rc.

–bi Initialize the alias database.

–bm Deliver mail in the usual way (default).

–bp Print a listing of the queue.

–bs Use the SMTP protocol as described in RFC821 on standard input and output. This flag implies all the operations of the –ba flag that are compatible with SMTP.

–bt Run in address test mode. This mode reads addresses and shows the steps in parsing; it is used for debugging configuration tables.

–bv Verify names only – do not try to collect or deliver a message. Verify mode is normally used for validating users or mailing lists.

–bz Create the configuration freeze file.

–Cfile Use alternate configuration file. Sendmail refuses to run as root if an alternate configuration file is specified. The frozen configuration file is bypassed.

–dx Set debugging value to X.

–fullname Set the full name of the sender.
-f
Sets the name of the "from" person (i.e., the sender of the mail).
-f can only be used by "trusted" users (normally root, daemon,
and netword) or if the person you are trying to become is the same
as the person you are.

-hN
Set the hop count to N. The hop count is incremented every
time the mail is processed. When it reaches a limit, the mail is
returned with an error message, the victim of an aliasing loop. If
not specified, "Received:" lines in the message are counted.

-n
Don't do aliasing.

-ox value
Set option x to the specified value. Options are described below.

-q[time]
Processed saved messages in the queue at given intervals. If time
is omitted, process the queue once. Time is given as a tagged
number, with 's' being seconds, 'm' being minutes, 'h' being
hours, 'd' being days, and 'w' being weeks. For example,
"-q1h30m" or "-q90m" would both set the timeout to one hour
thirty minutes. If time is specified, sendmail will run in back-
ground. This option can be used safely with -bd.

-xname
An alternate and obsolete form of the -x flag.

-t
Read message for recipients. To:, Cc:, and Bcc: lines will be
scanned for recipient addresses. The Bcc: line will be deleted
before transmission. Any addresses in the argument list will be
suppressed, that is, they will not receive copies even if listed in
the message header.

-v
Go into verbose mode. Alias expansions will be announced, etc.

There are also a number of processing options that may be set. Normally these will
only be used by a system administrator. Options may be set either on the command
line using the -o flag or in the configuration file. The options are:

Af
Use alternate alias file.

C
On mailers that are considered "expensive" to connect to, don't
initiate immediate connection. This requires queueing.

Dx
Set the delivery mode to x. Delivery modes are 'I' for interactive
(synchronous) delivery, 'B' for background (asynchronous)
delivery, and 'Q' for queue only - i.e., actual delivery is done the
next time the queue is run.

D
Try to automatically rebuild the alias database if necessary.

Ex
Set error processing to mode x. Valid modes are 'm' to mail back
the error message, 'w' to "write" back the error message (or mail
it back if the sender is not logged in), 'p' to print the errors on the
terminal (default), 'q' to throw away error messages (only exit
status is returned), and 'e' to do special processing for the Berk-
Net. If the text of the messages is not mailed back by modes 'm' or
'w' and if the sender is local to this machine, a copy of the mes-
sage is appended to the file "dead.letter" in the sender's home
directory.

Fmode
The mode to use when creating temporary files.

f
Save UNIX-style From lines at the front of messages.
The default group id to use when calling mailers.

The SMTP help file.

Do not take dots on a line by themselves as a message terminator.

The log level.

Send to "me" (the sender) also if I am in an alias expansion.

If set, this message may have old style headers. If not set, this message is guaranteed to have new style headers (i.e., commas instead of spaces between addresses). If set, an adaptive algorithm is used that will correctly determine the header format in most cases.

Select the directory in which to queue messages.

The timeout on reads; if none is set, sendmail will wait forever for a mailer. This option violates the word (if not the intent) of the SMTP specification, show the timeout should probably be fairly large.

Save statistics in the named file.

Always instantiate the queue file, even under circumstances where it is not strictly necessary. This provides safety against system crashes during delivery.

Set the timeout on undelivered messages in the queue to the specified time. After delivery has failed (e.g., because of a host being down) for this amount of time, failed messages will be returned to the sender. The default is three days.

Set the name of the time zone.

Set the default user id for mailers.

In aliases, the first character of a name may be a vertical bar to cause interpretation of the rest of the name as a command to pipe the mail to. It may be necessary to quote the name to keep sendmail from suppressing the blanks from between arguments. For example, a common alias is:

`msgs: "l /usr/ucb/mgs -s"

Aliases may also have the syntax "include:filename" to ask sendmail to read the named file for a list of recipients. For example, an alias such as:

`poets: "include:/usr/local/lib/poets.list"

would read /usr/local/lib/poets.list for the list of addresses making up the group.

Sendmail returns an exit status describing what it did. The codes are defined in

<sysexits.h>

<table>
<thead>
<tr>
<th>Exit Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EX_OK</td>
<td>Successful completion on all addresses.</td>
</tr>
<tr>
<td>EX_NOUSER</td>
<td>User name not recognized.</td>
</tr>
<tr>
<td>EX_UNAVAILABLE</td>
<td>Catchall meaning necessary resources were not available.</td>
</tr>
<tr>
<td>EX_SYNTAX</td>
<td>Syntax error in address.</td>
</tr>
<tr>
<td>EX_SOFTWARE</td>
<td>Internal software error, including bad arguments.</td>
</tr>
<tr>
<td>EX_OSERR</td>
<td>Temporary operating system error, such as &quot;cannot fork&quot;.</td>
</tr>
<tr>
<td>EX_NOHOST</td>
<td>Host name not recognized.</td>
</tr>
<tr>
<td>EX_TEMPFAIL</td>
<td>Message could not be sent immediately, but was queued.</td>
</tr>
</tbody>
</table>
If invoked as *newaliases*, *sendmail* will rebuild the alias database. If invoked as *mailq*, *sendmail* will print the contents of the mail queue.

**FILES**

Except for `/usr/lib/sendmail.cf`, these pathnames are all specified in `/usr/lib/sendmail.cf`. Thus, these values are only approximations.

- `/usr/lib/aliases` raw data for alias names
- `/usr/lib/aliases.pag` data base of alias names
- `/usr/lib/aliases.dir` configuration file
- `/usr/lib/sendmail.fc` frozen configuration
- `/usr/lib/sendmail.hf` help file
- `/usr/lib/sendmail.st` collected statistics
- `/usr/spool/mqueue/*` temp files

**SEE ALSO**

mail(1), mailx(1), rmail(1), syslog(3), aliases(4), sendmail.cf(4), mailaddr(5)

[System Administrator's Guide](#)
NAME
syslogd – log systems messages

SYNOPSIS
/etc/syslogd [-f configfile] [-m markinterval] [-d ]

DESCRIPTION
Syslogd reads and logs messages into a set of files described by the configuration file
/etc/syslog.conf. Each message is one line. A message can contain a priority code, marked
by a number in angle braces at the beginning of the line. Priorities are defined in <sys/syslog.h>. Syslogd reads from the UNIX domain socket /dev/log, from
an Internet domain socket specified in /etc/services, and from the special device
/dev/klog (to read kernel messages).

Syslogd configures when it starts up and whenever it receives a hangup signal. Lines
in the configuration file have a selector to determine the message priorities to which
the line applies and an action. The action field are separated from the selector by one
or more tabs.

Selectors are semicolon separated lists of priority specifiers. Each priority has a facility
describing the part of the system that generated the message, a dot, and a level
indicating the severity of the message. Symbolic names may be used. An asterisk
selects all facilities. All messages of the specified level or higher (greater severity) are
selected. More than one facility may be selected using commas to separate them. For example:

*.emerg;mail,daemon.crit

Selects all facilities at the emerg level and the mail and daemon facilities at the crit level.

Known facilities and levels recognized by syslogd are those listed in syslog(3) without
the leading "LOG_.". The additional facility "mark" has a message at priority
LOG_INFO sent to it every 20 minutes (this may be changed with the -m flag). The
"mark" facility is not enabled by a facility field containing an asterisk. The level
"none" may be used to disable a particular facility. For example,

*.debug;mail.none

Sends all messages except mail messages to the selected file.

The second part of each line describes where the message is to be logged if this line is
selected. There are four forms:

• A filename (beginning with a leading slash). The file will be opened in append
mode.

• A hostname preceded by an at sign ("@"). Selected messages are forwarded to
the syslogd on the named host.

• A comma separated list of users. Selected messages are written to those users if
they are logged in.

• An asterisk. Selected messages are written to all logged-in users.

Blank lines and lines beginning with '#' are ignored.

For example, the configuration file:

kern,mark.debug /dev/console
*.notice,mail.info /usr/spool/adm/syslog
*.crit /usr/adm/critical
kern.err @ucbarpa
*.emerg *
*.alert eric,kridle
*alert;auth.warning    ralph
logs all kernel messages and 20 minute marks onto the system console, all notice (or higher) level messages and all mail system messages except debug messages into the file /usr/spool/adm/syslog, and all critical messages into /usr/adm/critical; kernel messages of error severity or higher are forwarded to ucbarp. All users will be informed of any emergency messages, the users “eric” and “kridle” will be informed of any alert messages, and the user “ralph” will be informed of any alert message, or any warning message (or higher) from the authorization system.

The flags are:

- `f` Specify an alternate configuration file.
- `m` Select the number of minutes between mark messages.
- `d` Turn on debugging.

Syslogd creates the file /etc/syslog.pid, if possible, containing a single line with its process id. This can be used to kill or reconfigure syslogd.

To bring syslogd down, it should be sent a terminate signal (e.g. kill `cat /etc/syslog.pid`).

**FILES**

/etc/syslog.conf  the configuration file
/etc/syslog.pid   the process id
/dev/log         Name of the UNIX domain datagram log socket
/dev/klog        The kernel log device

**SEE ALSO**

logger(1), syslog(3)
NAME
telnetd – DARPA TELNET protocol server

SYNOPSIS
/etc/telnetd

DESCRIPTION
telnetd is a server which supports the DARPA standard TELNET virtual terminal protocol. telnetd is invoked by the internet server (see inetd(8)), normally for requests to connect to the TELNET port as indicated by the /etc/services file (see services(5)).
telnetd operates by allocating a pseudo-terminal device (see pty(4)) for a client, then creating a login process which has the slave side of the pseudo-terminal as stdin, stdout, and stderr. telnetd manipulates the master side of the pseudo-terminal, implementing the TELNET protocol and passing characters between the remote client and the login process.

When a TELNET session is started up, telnetd sends TELNET options to the client side indicating a willingness to do remote echo of characters, to suppress go ahead, and to receive terminal type information from the remote client. If the remote client is willing, the remote terminal type is propagated in the environment of the created login process. The pseudo-terminal allocated to the client is configured to operate in “cooked” mode, and with XTABS and CRMOD enabled (see tty(4)).
telnetd is willing to do: echo, binary, suppress go ahead, and timing mark. telnetd is willing to have the remote client do: binary, terminal type, and suppress go ahead.

SEE ALSO
telnet(1C)

BUGS
Some TELNET commands are only partially implemented.
The TELNET protocol allows for the exchange of the number of lines and columns on the user’s terminal, but telnetd doesn’t make use of them.

Because of bugs in the original 4.2 BSD telnet(1C), telnetd performs some dubious protocol exchanges to try to discover if the remote client is, in fact, a 4.2 BSD telnet(1C).

Binary mode has no common interpretation except between similar operating systems (Unix in this case).
The terminal type name received from the remote client is converted to lower case.
The packet interface to the pseudo-terminal (see pty(4)) should be used for more intelligent flushing of input and output queues.
telnetd never sends TELNET go ahead commands.
NAME
zic – time zone compiler

SYNOPSIS
zic [ -v ] [ -d directory ] [ -l localtime ] [ -L leapsecondfilename ] [ -s ] [ filename ... ]

DESCRIPTION
Zic reads text from the file(s) named on the command line and creates the time conversion information files specified in this input. If a filename is –, the standard input is read.

These options are available:
-\(d\) directory
  Create time conversion information files in the named directory rather than in the standard directory named below.

-\(l\) timezone
  Use the given time zone as local time. Zic will act as if the file contained a link line of the form

  Link timezone localtime

-\(L\) leapsecondfilename
  Read leap second information from the file with the given name. If this option is not used, leap second information is read from a file named "leapseconds".

-\(v\)
  Complain if a year that appears in a data file is outside the range of years representable by time(2) values.

-\(s\)
  Limit time values stored in output files to values that are the same whether they’re taken to be signed or unsigned. You can use this option to generate SVVS-compatible files.

Input lines are made up of fields. Fields are separated from one another by any number of white space characters. Leading and trailing white space on input lines is ignored. An unquoted sharp character (#) in the input introduces a comment which extends to the end of the line the sharp character appears on. White space characters and sharp characters may be enclosed in double quotes (") if they’re to be used as part of a field. Any line that is blank (after comment stripping) is ignored. Non-blank lines are expected to be of one of three types: rule lines, zone lines, and link lines.

A rule line has the form

Rule NAME FROM TO TYPE IN ON AT SAVE LETTER/S

For example:

Rule USA 1969 1973 – Apr lastSun 2:00 1:00 D

The fields that make up a rule line are:

NAME  Gives the (arbitrary) name of the set of rules this rule is part of.
FROM  Gives the first year in which the rule applies. The word minimum (or an abbreviation) means the minimum year with a representable time value. The word maximum (or an abbreviation) means the maximum year with a representable time value.
TO  Gives the final year in which the rule applies. In addition to minimum and maximum (as above), the word only (or an abbreviation) may be used to repeat the value of the FROM field.

TYPE  Gives the type of year in which the rule applies. If TYPE is – then the rule applies in all years between FROM and TO inclusive; if TYPE is uspres, the rule applies in U.S. Presidential election years; if TYPE is nonpres, the rule applies in years other than U.S. Presidential election years. If TYPE is something else, then zic executes the command

\[
\text{yearistype year type}
\]

to check the type of a year: an exit status of zero is taken to mean that the year is of the given type; an exit status of one is taken to mean that the year is not of the given type.

IN  Names the month in which the rule takes effect. Month names may be abbreviated.

ON  Gives the day on which the rule takes effect. Recognized forms include:

- 5 the fifth of the month
- lastSun the last Sunday in the month
- lastMon the last Monday in the month
- Sun>=8 first Sunday on or after the eighth
- Sun<=25 last Sunday on or before the 25th

Names of days of the week may be abbreviated or spelled out in full. Note that there must be no spaces within the ON field.

AT  Gives the time of day at which the rule takes effect. Recognized forms include:

- 2 time in hours
- 2:00 time in hours and minutes
- 15:00 24-hour format time (for times after noon)
- 1:28:14 time in hours, minutes, and seconds

Any of these forms may be followed by the letter w if the given time is local “wall clock” time or s if the given time is local “standard” time; in the absence of w or s, wall clock time is assumed.

SAVE  Gives the amount of time to be added to local standard time when the rule is in effect. This field has the same format as the AT field (although, of course, the w and s suffixes are not used).

LETTER/S  Gives the “variable part” (for example, the “S” or “D” in “EST” or “EDT”) of time zone abbreviations to be used when this rule is in effect. If this field is –, the variable part is null.

A zone line has the form

\[
\text{Zone NAME GMTOFF RULES/SAVE FORMAT [UNTIL]}
\]

For example:

\[
\text{Zone Australia/South–west 9:30 Aus CST 1987 Mar 15 2:00}
\]

The fields that make up a zone line are:
NAME The name of the time zone. This is the name used in creating the time conversion information file for the zone.

GMTOFF
The amount of time to add to GMT to get standard time in this zone. This field has the same format as the AT and SAVE fields of rule lines; begin the field with a minus sign if time must be subtracted from GMT.

RULES/SAVE
The name of the rule(s) that apply in the time zone or, alternately, an amount of time to add to local standard time. If this field is – then standard time always applies in the time zone.

FORMAT
The format for time zone abbreviations in this time zone. The pair of characters %s is used to show where the “variable part” of the time zone abbreviation goes.

UNTIL
The time at which the GMT offset or the rule(s) change for a location. It is specified as a year, a month, a day, and a time of day. If this is specified, the time zone information is generated from the given GMT offset and rule change until the time specified.

The next line must be a “continuation” line; this has the same form as a zone line except that the string “Zone” and the name are omitted, as the continuation line will place information starting at the time specified as the UNTIL field in the previous line in the file used by the previous line. Continuation lines may contain an UNTIL field, just as zone lines do, indicating that the next line is a further continuation.

A link line has the form

Link LINK-FROM LINK-TO

For example:

Link US/Eastern EST5EDT

The LINK-FROM field should appear as the NAME field in some zone line; the LINK-TO field is used as an alternate name for that zone.

Except for continuation lines, lines may appear in any order in the input.

Lines in the file that describes leap seconds have the following form:

Leap YEAR MONTH DAY HH:MM:SS CORR R/S

For example:

Leap 1974 Dec 31 23:59:60 + S

The YEAR, MONTH, DAY, and HH:MM:SS fields tell when the leap second happened. The CORR field should be “+” if a second was added or “-” if a second was skipped. The R/S field should be (an abbreviation of) “Stationary” if the leap second time given by the other fields should be interpreted as GMT or (an abbreviation of) “Rolling” if the leap second time given by the other fields should be interpreted as local wall clock time.
NOTE

For areas with more than two types of local time, you may need to use local standard time in the AT field of the earliest transition time's rule to ensure that the earliest transition time recorded in the compiled file is correct.

FILES

/etc/zoneinfo standard directory used for created files leapseconds default leap second information file

SEE ALSO

cctime(3), zdump(8)