SEQUENT

DYNIX/ptx Reference Manual
Section 1, A-L

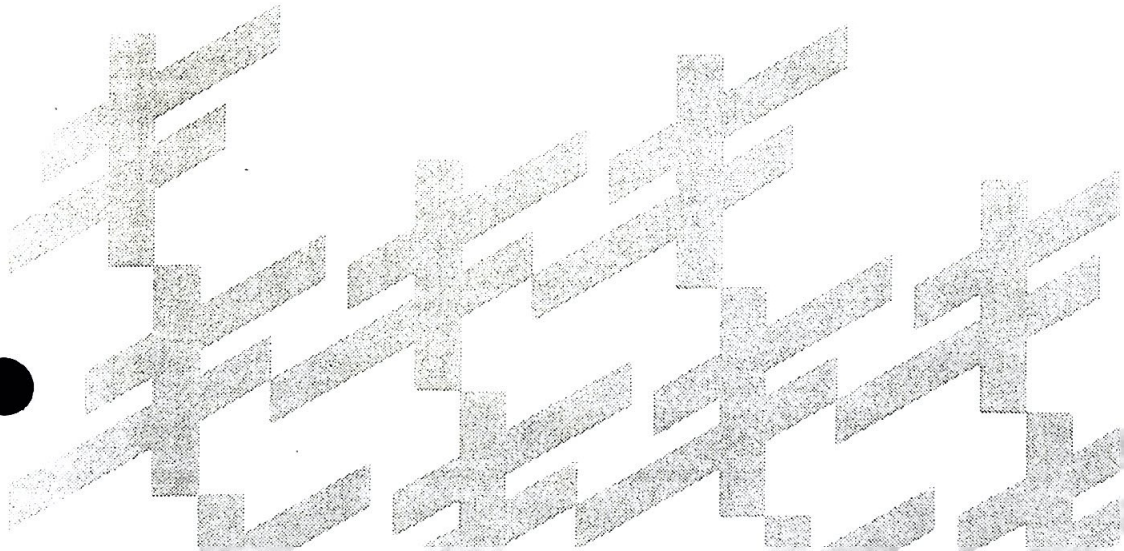
1003-49230-01

THIS SHEET LIFTER
MUST BE PLACED ON
TOP OF CONTENTS

GRAPHIC PRODUCTS
Chicago, Illinois 60609

DYNIX/ptx®
Reference Manual
Section 1, A-L

1003-49230-01
Sequent Computer Systems, Inc.



ACT is a trademark of Micro-Term.
Ann Arbor is a trademark of Ann Arbor Terminals.
Beehive is a trademark of Beehive International.
Concept is a trademark of Human Designed Systems.
Dataphone is a registered trademark of AT&T.
DEC is a trademark of Digital Equipment.
Diablo is a registered trademark of Xerox.
Digital is a trademark of Digital Equipment.
DOCUMENTER'S WORKBENCH is a trademark of AT&T.
DYNIX and DYNIX/ptx are registered trademarks of Sequent Computer Systems, Inc.
HP is a trademark of Hewlett-Packard.
i386 is a trademark of Intel Corp.
LSI is a trademark of Lear Siegler.
Micro-term and MIME are trademarks of Micro-Term.
MULTIBUS is a trademark of Intel Corp.
PDP is a trademark of Digital Equipment.
Sequent is a trademark of Sequent Computer Systems, Inc.
Symmetry is a registered trademark of Sequent Computer Systems, Inc.
Tektronix and Tektronix 4010 are registered trademarks of Tektronix.
Teleray is a trademark of Research.
Teletype is a registered trademark of AT&T.
TeleVideo is a registered trademark of TeleVideo Systems.
UNIX is a registered trademark of AT&T.
VAX is a trademark of Digital Equipment.
Versatec is a registered trademark of Versatec.
VT100 is a trademark of Digital Equipment.

Copyright © 1989, 1990 by Sequent Computer Systems, Inc. All rights reserved. This document may not be copied or reproduced in any form without permission from Sequent Computer Systems, Inc. Information in this document is subject to change without notice.

Printed in the United States of America.

About This Manual

Overview

The *DYNIX/ptx Reference Manual* is for programmers working in the DYNIX/ptx operating system environment. This manual describes the programming features of the operating system and assumes that the audience is familiar with the operating system. It does not provide a general overview of the operating system or implementation details of the system.

Organization

The *DYNIX/ptx Reference Manual* is contained in five binders organized as follows:

- Section 1 A-F **Commands**
- Section 1 G-Z **Commands**
- Section 2 **System Calls**
  Section 2SEQ **Sequent System Calls**
  Section 3 **Subroutines**
    - Section 3C **C Library Routines**
    - Section 3M **Mathematical Library Routines**
    - Section 3N **Networking Support Utilities**
    - Section 3PPS **Sequent Parallel Programming Library**
    - Section 3S **Standard I/O Library Routines**
    - Section 3SEQ **Sequent I/O Library Routines**
    - Section 3X **Special Routines**
- Section 4 **File Formats**
  Section 5 **Miscellaneous Facilities**
- Section 7 **Special Files**
  Section 8 **Stand-alone Utilities**

Each binder also contains the following items:

- **Table of Contents** that alphabetically lists the contents of the entire manual set
- **Permuted Index** that lists each programming feature in the entire manual set

The permuted index is a list of keywords, given alphabetically in the second of three columns, together with the context in which each keyword is found. The following shows four separate entries illustrating four ways that the **cat** command can be looked up in the permuted index. Note that you can find the **cat** command by looking up "cat," "concatenate," "files," or "print."

```
                    cat: concatenate and print files.....cat(1)
               cat: concatenate and print files..... cat(1)
concatenate and print files.....cat(1)
     concatenate and print files..... cat(1)
```

Keywords are either topical keywords or the names of manual entries themselves. Entries are identified with their section numbers shown in parentheses. The numbers are important because there is duplication of names among the sections. The right column lists the name of the manual page on which each keyword may be found. The left column contains useful information about the keyword.

The following paragraphs briefly describe the other sections of the manual.

Section 1: Commands describes operating system commands and commands that support C and other programming languages.

Section 2: System Calls describes the access to the services provided by the operating system kernel, including the C language interface.

Section 2SEQ: Sequent System Calls describes the access to the Sequent services provided by the operating system kernel, including the C language interface.

Section 3: Subroutines describes the available subroutines. Their binary versions reside in various system libraries in the directories */lib* and */usr/lib*. The introduction to Section 3 describes these libraries and the files in which they are stored.

Section 4: File Formats documents the structure of particular kinds of files; for example, the format of the output of the link editor is given in *a.out*(4). Excluded are files used by only one command (for example, the assembler's intermediate files). In general, the C language structures corresponding to these formats can be found in the directories */usr/include* and */usr/include/sys*.

Section 5: Miscellaneous Facilities contains a variety of descriptions including descriptions of character sets, macro packages, and so forth.

Section 7: Special Files describes various special files that refer to specific hardware peripherals and operating system device drivers.

Section 8: Stand-alone Utilities describes stand-alone procedures.

References to manual pages with numbers other than those above mean that the utility is contained in the appropriate section of another manual. Each section consists of independent entries which are alphabetized, with the exception of the introductions that begin some sections. Section 3 is divided into subsections in alphabetical order by suffix. Some entries may describe several routines, commands, and so forth. In such cases, the entry appears only once, alphabetized under its *primary* name, the name that appears at the upper corners of each manual page.

Each manual page can be viewed online with the **man**(1) command. For example, enter **man shell** to view the **shell** manual page. Enter **man man** for more information about the **man** command.

Notational Conventions

Manual pages are based on a common format. The main headings are described here—not all of them appear in all cases:

- **NAME** gives the name(s) of the entry and briefly states its purpose. This entry is used to create the Permuted Index and the Table of Contents.
- **SYNOPSIS** summarizes the usage of the program being described. Conventions vary in some sections but are generally as follows:

- **Boldface** strings are literals and are to be typed just as they appear.
- *Italic* strings usually represent substitutable argument prototypes and program names found elsewhere in the manual.
- Square brackets [ ] around an argument prototype indicate that the argument is optional. When an argument prototype is given as *name* or *file*, it refers to a *filename*.
- Ellipses ... are used to show that the previous argument prototype may be repeated.
- A final convention is used by the commands themselves. An argument beginning with a minus –, plus +, or equal sign = is often taken to be some sort of flag argument even if it appears in a position where a file name could appear. Therefore, it is unwise to have files whose names begin with –, +, or =.

- **CONFIGURATION ENTRY** shows how a driver is called out in a system configuration file.
- **DESCRIPTION** describes the entry, including its purpose and how it is used.
- **FIELDS** describe system file or record entries.
- **EXAMPLES** and **BINARY CONFIGURATION** give examples of usage.
- **FILES** list files that are part of a program, created by a program, or related to a program.
- **SEE ALSO** points to related information in other manual pages.
- **DIAGNOSTICS** discuss the diagnostic (error) messages that may be produced. Messages that are self-explanatory are not listed.
- **WARNINGS** contains warnings against improper user actions.
- **NOTES** give hints about the use of the utility.
- **BUGS** describe known bugs in the software.

Contents

1 Commands

intro(1) ..... introduction to maintenance commands and application programs
300(1): 300, 300s ..... handle special functions of DASI 300 and 300s terminals
4014(1) ..... paginator for the Tektronix 4014 terminal
450(1) ..... handle special functions of the DASI 450 terminal
accept(1M): accept, reject ..... allow or prevent LP requests
acct(1M): acctdisk, acctdusg, accton, acctwtmp ..... overview of accounting and miscellaneous accounting commands
acctcms(1M) ..... command summary from per-process accounting records
acctcom(1) ..... search and print process accounting files
acctcon(1M): acctcon1, acctcon2 ..... connect-time accounting
acctmerg(1M) ..... merge or add total accounting files
acctprc(1M): acctprc1, acctprc2 ..... process accounting
acctsh(1M): chargefee, ckpacct, dodisk, lastlogin, monacct, nulladm, prctmp, prdaily, prtacct, shutacct, startup, turnacct ..... shell procedures for accounting
addcmdopt(1M) ..... add a command option to a menu
addformopt(1M) ..... add a form/SELECT window option to a menu
addmenuopt(1M) ..... add a menu option to a menu
admin(1) ..... create and administer SCCS files
apropos(1) ..... locate commands by keyword lookup
ar(1) ..... see *C and Language Tools Binder*
as(1) ..... see *C and Language Tools Binder*
asa(1) ..... interpret ASA carriage control characters
assist(1) ..... assistance using operating system commands
astgen(1) ..... program for generating/modifying ASSIST menus or command forms
at(1): at, batch ..... execute commands at a later time
awk(1) ..... pattern scanning and processing language
banner(1) ..... make posters
basename(1): basename, dirname ..... deliver portions of pathnames
bc(1) ..... arbitrary-precision arithmetic language
bcheckrc(1M) ..... system initialization procedures
bdflush(1M) ..... update daemon
bdiff(1) ..... big diff
bfs(1) ..... big file scanner
bomverify(1M) ..... perform an installation consistency check based on the BOM
bootflags(1M) ..... display/change system boot flags
bp(1M) ..... binary patcher
cal(1) ..... print calendar

calendar(1) ..... reminder service
captoinfo(1M) ..... convert a termcap description into a terminfo description
cat(1) ..... concatenate and print files
cb(1) ..... see *C and Language Tools Binder*
cc(1) ..... see *C and Language Tools Binder*
cd(1) ..... change working directory
cdc(1) ..... change the delta commentary of an SCCS delta
cflow(1) ..... see *C and Language Tools Binder*
checkeq(1) ..... check documents for correct eqn syntax.
checkfsys(1M) ..... check a filesystem
chmod(1) ..... change mode
chown(1): chown, chgrp ..... change owner or group
chroot(1M) ..... change root directory for a command
chrtbl(1M) ..... generate character classification and conversion tables
ci(1) ..... check in RCS revisions
ckbupscd(1M) ..... check filesystem backup schedule
clri(1M) ..... clear inode
cmp(1) ..... compare two files
co(1) ..... check out RCS revisions
col(1) ..... filter reverse line-feeds
comb(1) ..... combine SCCS deltas
comm(1) ..... select or reject lines common to two sorted files
compress(1): compress,
uncompress, zcat ..... compress and expand data
config(1M) ..... build system configuration files
coproc(1F): cocreate, cosend,
cocheck, coreceive, codestroy ..... communicate to a process
cp(1): cp, ln, mv ..... copy, link or move files
cpp(1) ..... see *C and Language Tools Binder*
cpio(1) ..... copy file archives in and out
cprs(1) ..... see *C and Language Tools Binder*
cpset(1) ..... copy file and set mode, user, and group IDs
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
ct(1C) ..... spawn getty to a remote terminal
ctags(1) ..... create a tags file
ctrace(1) ..... see *C and Language Tools Binder*
cu(1C) ..... call another UNIX system
cut(1) ..... cut out selected fields of each line of a file
cxref(1) ..... see *C and Language Tools Binder*
date(1) ..... print and set the date
dbx(1) ..... see *C and Language Tools Binder*
dc(1) ..... desk calculator
dcheck(1M) ..... filesystem directory consistency check
dd(1M) ..... convert and copy a file
delta(1) ..... make a delta (change) to an SCCS file

deroff(1) ..... remove nroff, tbl, and eqn constructs
devbuild(1M): devbuild,
devdestroy ..... build and destroy virtual devices
devnm(1M) ..... device name
df(1M) ..... report number of free disk blocks and inodes
diff(1) ..... differential file comparison
diff3(1) ..... 3-way differential file comparison
dircmp(1) ..... directory comparison
dis(1) ..... see *C and Language Tools Binder*
diskusg(1M) ..... generate disk accounting data by user ID
dltsectbl(1M) ..... refresh security table
dmesg(1M) ..... collect system diagnostic messages to form error log
du(1M) ..... summarize disk usage
dump(1) ..... see *C and Language Tools Binder*
dump(1M) ..... incremental filesystem dump
dumpconf(1M) ..... dump kernel auto-configuration device table
dumpfs(1M) ..... dump filesystem information
ec(1M) ..... forms compiler
echo(1) ..... echo arguments
echo(1F) ..... put string on virtual output
ed(1): ed, red ..... text editor
edc(1M) ..... run an externally described command
edit(1) ..... text editor (variant of ex for casual users)
edquota(1M) ..... edit user quotas
efl(1) ..... Extended Fortran Language
egrep(1) ..... search a file for a pattern using full regular expressions
enable(1): enable, disable ..... enable/disable LP printers
env(1) ..... set environment for command execution
ex(1) ..... text editor
expr(1) ..... evaluate arguments as an expression
factor(1) ..... obtain the prime factors of a number
ff(1M) ..... list filenames and statistics for a filesystem
fgrep(1) ..... search a file for a character string
file(1) ..... determine file type
find(1) ..... find files
format(1M) ..... format disks or perform other defect management functions
fsck(1M) ..... filesystem consistency check and interactive repair
fsirand(1M) ..... install random inode generation numbers
fsplit(1) ..... split f77, ratfor, or efl files
fsstat(1M) ..... report filesystem status
fstyp(1M) ..... determine filesystem identifier
fuser(1M) ..... identify processes using a file or file structure
fwtmp(1M): fwtmp, wtmpfix ..... manipulate connect accounting records
gdev(1G): hpd, erase, hardcopy,
tekset, td ..... graphical device routines and filters
ged(1G) ..... graphical editor
gencc(1M) ..... create a front-end to the cc command
get(1) ..... get a version of an SCCS file
getfrm(1F) ..... returns the current frame number

getitems(1F) ..... returns a list of the currently marked menu items
getopt(1) ..... parse command options
getopts(1): getopts, getoptcvt ..... parse command options
getty(1M) ..... set terminal type, modes, speed, and line discipline
glossary(1) ..... definitions of common operating system terms and symbols
graph(1G) ..... draw a graph
graphics(1G) ..... access graphical and numerical commands
greek(1) ..... select terminal filter
grep(1) ..... search a file for a pattern
groups(1) ..... show group memberships
gutil(1G) ..... graphical utilities
help(1) ..... operating system Help Facility
helpadm(1M) ..... change the Help Facility database
hp(1) ..... handle special functions of Hewlett-Packard terminals
hpio(1) ..... Hewlett-Packard 2645A terminal tape file archiver
icheck(1M) ..... filesystem storage consistency check
id(1M) ..... print user and group IDs and names
ident(1) ..... identify files
indicator(1F) ..... display application specific alarms and/or the working indicator
infocmp(1M) ..... compare or print out terminfo descriptions
init(1M): init, telinit ..... process control initialization
install(1M) ..... install commands
installboot(1M) ..... installs a boot program on a disk
ipcrm(1) ..... remove a message queue, semaphore set, or shared memory ID
ipcs(1) ..... report inter-process communication facilities status
isismgr(1M) ..... ISIS file manager, updates text files in ISIS file format
join(1) ..... relational database operator
kill(1) ..... terminate a process
killall(1M) ..... kill all active processes
ksh(1): ksh, rksh ..... the KornShell, a standard/restricted command and programming language
labelit(1M) ..... provide labels for filesystems
ld(1) ..... see *C and Language Tools Binder*
lex(1) ..... generate programs for simple lexical tasks
line(1) ..... read one line
link(1M): link, unlink ..... link and unlink files and directories
lint(1) ..... see *C and Language Tools Binder*
list(1) ..... see *C and Language Tools Binder*
locate(1) ..... use keywords to identify an operating system command
login(1) ..... sign on
logname(1) ..... get login name
lorder(1) ..... see *C and Language Tools Binder*
lp(1): lp, cancel ..... send/cancel requests to the LP print service
lpadmin(1M) ..... configure the LP print service
lpfilter(1M) ..... administer filters used with the LP print service
lpforms(1M) ..... administer forms used with the LP print service

lpsched(1M): lpsched, lpshut,
lpmove ..... start/stop the LP print service and move requests
lpstat(1) ..... print information about the status of the LP print service
lpusers(1M) ..... set printing queue priorities
ls(1) ..... list contents of directory
m4(1) ..... macro processor
machid(1): i386, ns32000, pdp11,
u3b, u3b2, u3b5, vax ..... get processor type truth value
mail(1): mail, rmail ..... send mail to users or read mail
mailbug(1) ..... submit a bug report or enhancement request
mailx(1) ..... interactive message processing system
make(1) ..... maintain, update, and regenerate groups of programs
makealiases(1M): makealiases,
makewhatis ..... generate man page databases
makedev(1M) ..... make system special files
makekey(1) ..... generate encryption key
man(1) ..... display on-line man pages
mcs(1) ..... manipulate the object file comment section
menu(1M) ..... menu processor
menumsg(1M) ..... route user message to menu message queue
merge(1) ..... three-way file merge
mesg(1) ..... permit or deny messages
message(1F) ..... puts its arguments on message line
mkdir(1) ..... make directories
mkfs(1M) ..... construct a filesystem
mknod(1M) ..... build special file
mkpwdbm(1M) ..... make a database for fast password lookup
mkvtoc(1M) ..... populate VTOC on disks
monitor(1M) ..... monitor system activity
more(1) ..... view a file one full screen at a time
mount(1M): mount, umount ..... mount and unmount filesystems
mountall(1M): mountall,
umountall ..... mount, unmount multiple filesystems
mt(1) ..... magnetic tape manipulating program
mvdir(1M) ..... move a directory
nawk(1) ..... pattern scanning and processing language
ncheck(1M) ..... generate pathnames from i-numbers
newform(1) ..... change the format of a text file
newfs(1M) ..... construct a new file system
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) ..... see *C and Language Tools Binder*
noage(1M): noage, nopff, noswap,
onproc ..... run a command with priority aging, PFF, swapping, or processor migration disabled
nohup(1) ..... run a command immune to hangups and quits
od(1) ..... octal dump
online(1M): online, offline ..... bring processors online and offline

pack(1): pack, pcat, unpack ..... compress and expand files
passmgmt(1M) ..... password files management
passwd(1) ..... change login password and password attributes
paste(1) ..... merge same lines of several files or subsequent lines of one file
pathconv(1F) ..... search Interpreter criteria for filename
pdbx(1) ..... see *C and Language Tools Binder*
pg(1) ..... file perusal filter for CRTs
pmap(1M) ..... display or alter parameters of virtual-to-physical mapping devices
pr(1) ..... print files
prof(1) ..... display profile data
prompt(1M) ..... prompt program
prs(1) ..... print an SCCS file
prtvtoc(1M) ..... print the VTOC of a block device
ps(1) ..... report process status
pwck(1M): pwck, grpck ..... password/group file checkers
pwconv(1M) ..... installs and updates /etc/shadow with information from /etc/passwd
pwd(1) ..... working directory name
pwdhd(1M) ..... daemon to oversee password database creation
pwfix(1M) ..... remove incorrectly formatted password entries
pwunconv(1M) ..... converts from a two to a one password file scheme
quota(1) ..... display disk usage and limits
quotacheck(1M) ..... filesystem quota consistency checker
quotaon(1M): quotaon, quotaoff ..... turn filesystem quotas on and off
ratfor(1) ..... rational Fortran dialect
rc0(1M) ..... run commands performed to stop the operating system
rc2(1M) ..... run commands performed for multi-user environment
rcs(1) ..... change RCS file attributes
rcsdiff(1) ..... compare RCS revisions
rcsintro(1) ..... introduction to RCS commands
rcsmerge(1) ..... merge RCS revisions
readfile(1F): readfile, longline ..... reads file and gets longest line
regcmp(1) ..... regular expression compile
regex(1F) ..... match patterns against a string
reinit(1F) ..... run an initialization file
repquota(1M) ..... summarize quotas for a filesystem
reset(1F) ..... reset the current form field to its default values
restore(1M) ..... incremental filesystem restore
rlog(1) ..... print log messages and other information about RCS files
rm(1): rm, rmdir ..... remove files or directories
rmdel(1) ..... remove a delta from an SCCS file
rmvopt(1M) ..... remove a menu option
run(1F) ..... run an executable
runacct(1M) ..... run daily accounting
sact(1) ..... print current SCCS file editing activity
sag(1G) ..... system activity graph
sar(1) ..... system activity reporter
sar(1M): sa1, sa2, sadc ..... system activity report package

savecore(1M) ..... save a core dump of the operating system
sccsdiff(1) ..... compare two versions of an SCCS file
sccstorcs(1) ..... build RCS file from SCCS file
sdiff(1) ..... side-by-side difference program
sed(1) ..... stream editor
set(1F): set, unset ..... set and unset environment variables in core or in files
setcolor(1F) ..... redefine or create a color
setmnt(1M) ..... establish mount table
setpgrp(1) ..... run command in its own process group
setup(1) ..... initialize system for first user
sh(1): sh, rsh ..... shell, the standard/restricted command programming language
shell(1F) ..... run a command using shell
showcfg(1M) ..... show the configuration of the machine
shutdown(1M) ..... shut down system, change system state
size(1) ..... see *C and Language Tools Binder*
sleep(1) ..... suspend execution for an interval
sort(1) ..... sort and/or merge files
spell(1): spell, hashmake, spellin, hashcheck ..... find spelling errors
spline(1G) ..... interpolate smooth curve
split(1) ..... split a file into pieces
starter(1) ..... information about the operating system for beginning users
stat(1G) ..... statistical network useful with graphical commands
strace(1M) ..... print STREAMS trace messages
strclean(1M) ..... STREAMS error logger cleanup program
strerr(1M) ..... STREAMS error logger daemon
strip(1) ..... see *C and Language Tools Binder*
stty(1) ..... set the options for a terminal
su(1M) ..... become superuser or another user
sum(1) ..... print checksum and block count of a file
swap(1M) ..... swap administrative interface
sweepon(1M): sweepon, sweepoff ..... enable/disable patrol seeks on DCC disks
sync(1M) ..... update the superblock
sysadm(1) ..... menu interface to do system administration
tabs(1) ..... set tabs on a terminal
tail(1) ..... deliver the last part of a file
tar(1) ..... tape file archiver
tasktbl(1M) ..... create security table in shared memory
tee(1) ..... pipe fitting
test(1) ..... condition evaluation command
tfipc(1M) ..... interprocess communication reporter
tfmenush(1M) ..... assign menu system as user login shell
tfnotify(1M) ..... send completion message
tic(1M) ..... terminfo compiler
time(1): time, ptime ..... time a command
timex(1) ..... time a command; report process data and system activity
toc(1G): dtoc, ttoc, vtoc ..... graphical table of contents routines
touch(1) ..... update access and modification times of a file

tplot(1G) ..... graphics filters
tput(1) ..... initialize a terminal or query terminfo database
tr(1) ..... translate characters
true(1): true, false ..... provide truth values
tsort(1) ..... topological sort
tty(1) ..... get the name of the terminal
ttydevname(1) ..... get the device associated with a terminal or the terminal associated with a device
tunefs(1M) ..... tune up an existing filesystem
uadmin(1M) ..... administrative control
umask(1) ..... set file-creation mode mask
uname(1) ..... print name of current system
unget(1) ..... undo a previous get of an SCCS file
uniq(1) ..... report repeated lines in a file
units(1) ..... conversion program
uptime(1) ..... show how long system has been up
usage(1) ..... retrieve a command description and usage examples
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): uucp, uulog, uuname ..... UNIX-to-UNIX system copy
uugetty(1M) ..... set terminal type, modes, speed, and line discipline
uusched(1M) ..... the scheduler for the uucp file transport program
uustat(1C) ..... uucp status inquiry and job control
uuto(1C): uuto, uupick ..... public UNIX-to-UNIX system file copy
uutry(1M) ..... try to contact remote system with debugging on
uux(1C) ..... UNIX-to-UNIX system command execution
uuxqt(1M) ..... execute remote command requests
val(1) ..... validate SCCS file
vc(1) ..... version control
vi(1) ..... screen-oriented (visual) display editor based on ex
vmtune(1M) ..... tune system-wide virtual-memory parameters
volcopy(1M) ..... make literal copy of filesystem
vsig(1F) ..... synchronize a co-process with the controlling FMLI object
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
who(1) ..... who is on the system
whodo(1M) ..... who is doing what
write(1) ..... write to another user
xargs(1) ..... construct argument list(s) and execute command
yacc(1) ..... yet another compiler-compiler

2 System Calls

intro(2) ..... introduction to system calls and error numbers
access(2) ..... determine accessibility of a file
acct(2) ..... enable or disable process accounting
alarm(2) ..... set a process alarm clock
brk(2): brk, sbrk ..... change data segment space allocation
chdir(2) ..... change working directory
chmod(2) ..... change mode of file
chown(2) ..... change owner and group of a file
chroot(2) ..... change root directory
close(2) ..... close a file descriptor
creat(2) ..... create a new file or rewrite an existing one
dup(2) ..... duplicate an open file descriptor
dup2(2) ..... duplicate an open file descriptor
exec(2): execl, execv, execle, execve, execlp, execvp ..... execute a file
exit(2): exit, _exit ..... terminate process
fcntl(2) ..... file control
fork(2) ..... create a new process
getdents(2) ..... read directory entries and put in a file system independent format
getgroups(2) ..... get supplementary group IDs
getmsg(2) ..... get next message off a stream
getpid(2): getpid, getpgrp, getppid ..... get process, process group, and parent process IDs
getuid(2): getuid, geteuid, getgid, getegid ..... get real user, effective user, real group, and effective group IDs
ioctl(2) ..... control device
kill(2) ..... send a signal to a process or a group of processes
link(2) ..... link to a file
lseek(2) ..... move read/write file pointer
mkdir(2) ..... make a directory
mknod(2): mknod ..... make a directory, or a special or ordinary file
mount(2) ..... mount a filesystem
msgctl(2) ..... message control operations
msgget(2) ..... get message queue
msgop(2) ..... message operations
nice(2) ..... change priority of a process
open(2) ..... open for reading or writing
pathconf(2): pathconf, fpathconf ..... get configurable pathname variables
pause(2) ..... suspend process until signal
pipe(2) ..... create an interprocess channel
plock(2) ..... lock process, text, or data in memory
poll(2) ..... STREAMS input/output multiplexing
profil(2) ..... execution time profile
ptrace(2): ptrace, mptrace ..... process trace facility
putmsg(2) ..... send a message on a stream
read(2) ..... read from file
rename(2) ..... change the name of a file
rmdir(2) ..... remove a directory

semctl(2) ..... semaphore control operations
semget(2) ..... get set of semaphores
semop(2) ..... semaphore operations
setpgid(2) ..... manipulate signal sets
setpgrp(2) ..... set process group ID
setsid(2) ..... create session and set process group ID
setuid(2): setuid, setgid ..... set user and group IDs
shmctl(2) ..... shared memory control operations
shmget(2) ..... get shared memory segment identifier
shmop(2): shmop, shmat, shmdt ..... shared memory operations
sigaction(2) ..... examine and change signal actions
sigemptyset(2): sigemptyset,
sigfillset, sigaddset, sigdelset,
sigismember ..... manipulate signal sets
signal(2) ..... specify what to do upon receipt of a signal
sigpending(2) ..... examine pending signals
sigprocmask(2) ..... examine and change blocked signals
sigset(2): sigset, sighold, sigrelse,
sigignore, sigpause ..... signal management
sigsuspend(2) ..... wait for a signal
stat(2): stat, lstat, fstat ..... get file status
statfs(2): statfs, fstatfs ..... get filesystem information
stime(2) ..... set time
stty(2): stty, gtty ..... set and get terminal state (defunct)
sync(2) ..... update superblock
syscall(2) ..... indirect system call
sysconf(2) ..... get configurable system variables
sysi86(2) ..... machine specific functions
tcgetattr(2): tcgetattr, tcsetattr ..... get/set terminal state
tcgetpgrp(2) ..... get foreground process group ID
tcsetpgrp(2) ..... set foreground process group ID
time(2) ..... get time
times(2) ..... get process and child process times
uadmin(2) ..... administrative control
ulimit(2) ..... get and set user limits
umask(2) ..... set and get file creation mask
umount(2) ..... unmount a filesystem
uname(2) ..... get name of current operating system
unlink(2) ..... remove directory entry
ustat(2) ..... get filesystem statistics
utime(2) ..... set file access and modification times
vfork(2) ..... spawn new process in a virtual memory efficient way
wait(2): wait, waitpid ..... wait for a child process to stop or terminate
write(2) ..... write on a file

2SEQ Sequent System Calls

intro(2SEQ) ..... introduction to Sequent system calls
bdflush(2SEQ) ..... update daemon
fvhangup(2SEQ) ..... virtually hangup the specified controlling terminal
get_process_stats(2SEQ) ..... get resource utilization information
getdtablemax(2SEQ) ..... get maximum in-use descriptor table index
getdtablesize(2SEQ) ..... get descriptor table size
getpagesize(2SEQ) ..... get system page size
mmap(2SEQ) ..... map an open file into the processs address space
munmap(2SEQ) ..... unmap part of the processs address space
proc_ctl(2SEQ) ..... control over various process attributes
quotactl(2SEQ) ..... manipulate disk quotas
readlink(2SEQ) ..... read value of a symbolic link
readv(2SEQ) ..... read vector from file
setdtablesize(2SEQ) ..... set descriptor table size
setgroups(2SEQ) ..... set group access list
sigcontext(2SEQ) ..... get signal context
sigstack(2SEQ) ..... set and/or get signal stack context
symlink(2SEQ) ..... make symbolic link to a file
tmp_affinity(2SEQ) ..... bind a process to a processor
tmp_ctl(2SEQ) ..... multi-processor control
truncate(2SEQ): truncate,
ftruncate ..... truncate a file to a specified length
vm_ctl(2SEQ) ..... examine and change virtual memory tuning parameters
writev(2SEQ) ..... write vector on a file

3 Subroutines

intro(3) ..... introduction to functions and libraries

3C C Library Routines

a64l(3C): a64l, l64a ..... convert between long integer and base-64 ASCII string
abort(3C) ..... generate an IOT fault
abs(3C) ..... return integer absolute value
bsearch(3C) ..... binary search a sorted table
clock(3C) ..... report CPU time used
conv(3C): toupper, tolower,
_toupper, _tolower, toascii ..... translate characters
crypt(3C): crypt, setkey, encrypt ..... generate hashing encryption
ctime(3C): ctime, localtime,
gmtime, asctime, cftime, ascftime,
mktime, tzset ..... convert date and time to string
ctype(3C): isdigit, isxdigit,
islower, isupper, isalpha, isalnum,
isspace, iscntrl, ispunct, isprint,
isgraph, isascii, tolower, toupper,
toascci, _tolower, _toupper,
setchrclass ..... character handling

dev_to_ttyname(3C): dev_to_ttyname, ttyname_to_dev ..... convert between major/minor numbers and tty names
dial(3C) ..... establish an out-going terminal line connection
drand48(3C): drand48, erand48, lrand48, nrand48, mrand48, jrand48, srand48, seed48, lcong48 ..... generate uniformly distributed pseudorandom numbers
ecvt(3C): ecvt, fcvt, gcvt ..... convert floating-point number to string
end(3C): end, etext, edata ..... last locations in program
fpgetround(3C): fpgetround, fpsetround, fpgetmask, fpsetmask, fpgetsticky, fpsetsticky ..... IEEE floating-point environment control
frexp(3C): frexp, ldexp, modf ..... manipulate parts of floating-point numbers
ftw(3C) ..... walk a file tree
getcwd(3C) ..... get pathname of current working directory
getenv(3C) ..... return value for environment name
getgrent(3C): getgrent, getgrgid, getgrnam, setgrent, endgrent, fgetgrent ..... get group file entry
getlogin(3C) ..... get login name
getopt(3C) ..... get option letter from argument vector
getpass(3C) ..... read a password
getpw(3C) ..... get name from UID
getpwent(3C): getpwent, getpwuid, getpwnam, setpwent, endpwent, fgetpwent ..... get password file entry
getut(3C): getutent, getutid, getutline, pututline, setutent, endutent, utmpname, getutpid ..... access utmp file entry
hsearch(3C): hsearch, hcreate, hdestroy ..... manage hash search tables
isnan(3C): isnand, isnanf ..... test for floating-point NaN (Not-A-Number)
l3tol(3C): l3tol, ltol3 ..... convert between 3-byte integers and long integers
lockf(3C) ..... record locking on files
lsearch(3C): lsearch, lfind ..... linear search and update
malloc(3C): malloc, free, realloc, calloc ..... main memory allocator
memory(3C): memccpy, memchr, memcmp, memcpy, memset ..... memory operations
mkfifo(3C) ..... make a FIFO special file
mktemp(3C) ..... make a unique filename
monitor(3C) ..... prepare execution profile
nlist(3C) ..... get entries from name list
perror(3C): perror, errno, sys_errlist, sys_nerr ..... system error messages
putenv(3C) ..... change or add value to environment
putpwent(3C) ..... write password file entry
qsort(3C) ..... quicker sort
rand(3C): rand, srand ..... simple random-number generator
setjmp(3C): setjmp, longjmp ..... non-local goto
setlocale(3C) ..... set programs location

sigsetjmp(3C): sigsetjmp,
siglongjmp ..... non-local jumps
sleep(3C) ..... suspend execution for interval
ssignal(3C): ssignal, gsignal ..... software signals
stdipc(3C) ..... standard interprocess communication package
strftime(3C) ..... format time information
string(3C): strcat, strdup, strncat,
strcmp, strncmp, strcpy, strncpy,
strlen, strchr, strrchr, strpbrk,
strspn, strcspn, strtok ..... string operations
strstr(3C) ..... find first occurrence of character
strtod(3C): strtod, atof ..... convert string to double-precision number
strtol(3C): strtol, atol, atoi ..... convert string to integer
swab(3C) ..... swap bytes
tsearch(3C): tsearch, tfind,
tdelete, twalk ..... manage binary search trees
ttyname(3C): ttyname, isatty ..... find name of a terminal
ttyslot(3C) ..... find the slot in the utmp file of the current user

3M Mathematical Library Routines

bessel(3M): j0, j1, jn, y0, y1, yn ..... Bessel functions
erf(3M): erf, erfc ..... error function and complementary error function
exp(3M): exp, log, log10, pow, sqrt .... exponential, logarithm, power, square root functions
floor(3M): floor, ceil, fmod, fabs ..... floor, ceiling, remainder, absolute value functions
gamma(3M) ..... log gamma function
hypot(3M) ..... Euclidean distance function
matherr(3M) ..... error-handling function
sinh(3M): sinh, cosh, tanh ..... hyperbolic functions
trig(3M): sin, cos, tan, asin, acos,
atan, atan2 ..... trigonometric functions

3N Networking Support Utilities

t_accept(3N) ..... accept a connect request
t_alloc(3N) ..... allocate a library structure
t_bind(3N) ..... bind an address to a transport endpoint
t_close(3N) ..... close a transport endpoint
t_connect(3N) ..... establish a connection with another transport user
t_error(3N) ..... produce error message
t_free(3N) ..... free a library structure
t_getinfo(3N) ..... get protocol-specific service information
t_getstate(3N) ..... get the current state
t_listen(3N) ..... listen for a connect request
t_look(3N) ..... look at the current event on a transport endpoint
t_open(3N) ..... establish a transport endpoint
t_optmgmt(3N) ..... manage options for a transport endpoint
t_rcv(3N) ..... receive data or expedited data sent over a connection
t_rcvconnect(3N) ..... receive the confirmation from a connect request
t_rcvdis(3N) ..... retrieve information from disconnect
t_rcvrel(3N) ..... acknowledge receipt of an orderly release indication

t_rcvudata(3N) ..... receive a data unit
t_rcvuderr(3N) ..... receive a unit data error indication
t_snd(3N) ..... send data or expedited data over a connection
t_snddis(3N) ..... send user-initiated disconnect request
t_sndrel(3N) ..... initiate an orderly release
t_sndudata(3N) ..... send a data unit
t_sync(3N) ..... synchronize transport library
t_unbind(3N) ..... disable a transport endpoint

3PPS Sequent Parallel Programming Library

intro(3PPS) ..... introduction to Parallel Programming Library
brk(3PPS): brk, sbrk ..... change private data segment size
cpus_online(3PPS) ..... returns the number of CPUs on-line
m_fork(3PPS) ..... execute a subprogram in parallel
m_get_myid(3PPS) ..... return process identification
m_get_numprocs(3PPS) ..... return number of child processes
m_kill_procs(3PPS) ..... kill child processes
m_lock(3PPS): m_lock, m_unlock ..... lock, unlock locks
m_next(3PPS) ..... increment global counter
m_park_procs(3PPS): m_park_procs,
m_rele_procs ..... suspend and resume child process execution
m_set_procs(3PPS) ..... set number of child processes
m_single(3PPS): m_single,
m_multi ..... mark single-process code section
m_sync(3PPS) ..... check in at barrier
s_lock(3PPS): s_init_lock, s_lock,
s_clock, s_unlock ..... initialize, lock, unlock locks
s_wait_barrier(3PPS): s_init_barrier,
s_wait_barrier ..... initialize barrier, wait at barrier
shbrk(3PPS): shbrk, shsbrk ..... change shared data segment size
shmalloc(3PPS): shmalloc,
shrealloc, shfree ..... shared memory allocator

3S Standard I/O Library Routines

ctermid(3S) ..... generate filename for terminal
cuserid(3S) ..... get character login name of the user
fclose(3S): fclose, fflush ..... close or flush a stream
ferror(3S): ferror, feof, clearerr,
fileno ..... stream status inquiries
fopen(3S): fopen, freopen, fdopen ..... open a stream
fread(3S): fread, fwrite ..... binary input/output
fseek(3S): fseek, rewind, ftell ..... reposition a file pointer in a stream
getc(3S): getc, getchar, fgetc, getw .... get character or word from a stream
gets(3S): gets, fgets ..... get a string from a stream
popen(3S): popen, pclose ..... initiate pipe to/from a process
printf(3S): printf, fprintf, sprintf ..... print formatted output

putc(3S): putc, putchar, fputc,
putw ..... put character or word on a stream
puts(3S): puts, fputs ..... put a string on a stream
scanf(3S): scanf, fscanf, sscanf ..... convert formatted input
setbuf(3S): setbuf, setvbuf ..... assign buffering to a stream
stdio(3S) ..... standard buffered input/output package
system(3S) ..... issue a shell command
tmpfile(3S) ..... create a temporary file
tmpnam(3S): tmpnam, tempnam ..... create a name for a temporary file
ungetc(3S) ..... push character back into input stream
vprintf(3S): vprintf, vfprintf,
vsprintf ..... print formatted output of a varargs argument list

3SEQ Sequent I/O Library Routines

dis386(3SEQ) ..... disassemble Series 386 instructions
getgeombyname(3SEQ) ..... get disk geometry by name
getmntent(3SEQ): addmntent,
endmntent, getmntent,
hasmntopt, setmntent ..... get file system descriptor file entry
getpseudotty(3SEQ) ..... get pseudo terminals
getscsiinfo(3SEQ): getscsiinfo,
getscsimatch ..... get scsiinfo structure
getusclk(3SEQ): getusclk,
usclk_init ..... get/init microsecond clock
getzdinfo(3SEQ): getzdinfo,
getzdinfobyname ..... get zdinfo structure
initgroups(3SEQ) ..... initialize group access list
nblocks(3SEQ) ..... calculate number of blocks and indirect blocks
offline_all(3SEQ) ..... take all but one active processor off line
read_constab(3SEQ) ..... read constab entries into a structure
valloc(3SEQ) ..... aligned memory allocator

3X Special Routines

assert(3X) ..... verify program assertion
crypt(3X) ..... password and file encryption functions
curses(3X) ..... terminal screen handling and optimization package
directory(3X): opendir, readdir,
telldir, seekdir, rewinddir, closedir .... directory operations
getspent(3X): getspent, getspnam,
setspent, endspent, fgetspent,
lckpwdf, ulckpwdf ..... get shadow password file entry
ldahread(3X) ..... read the archive header of a member of an archive file
ldclose(3X): ldclose, ldaclose ..... close a common object file
ldfhread(3X) ..... read the file header of a common object file
ldgetname(3X) ..... retrieve symbol name for common object file symbol table entry

ldlread(3X): ldlread, ldlinit, ldlitem .. manipulate line number entries of a common object file function
ldlseek(3X): ldlseek, ldnlseek..... seek to line number entries of a section of a common object file
ldohseek(3X) ..... seek to the optional file header of a common object file
ldopen(3X): ldopen, ldaopen ..... open a common object file for reading
ldrseek(3X): ldrseek, ldnrseek ..... seek to relocation entries of a section of a common object file
ldshread(3X): ldshread, ldnshread .... read an indexednamed section header of a common object file
ldsseek(3X): ldsseek, ldnsseek ..... seek to an indexednamed section of a common object file
ldtbindex(3X) ..... compute the index of a symbol table entry of a common object file
ldtbread(3X) ..... read an indexed symbol table entry of a common object file
ldtbseek(3X) ..... seek to the symbol table of a common object file
logname(3X) ..... return login name of user
malloc(3X): malloc, free, realloc, calloc, mallopt, mallinfo ..... fast main memory allocator
ndbm(3X): dbm_open, dbm_close, dbm_fetch, dbm_store, dbm_delete, dbm_firstkey, dbm_nextkey, dbm_error, dbm_clearerr, dmb_blkflush ..... database subroutines
plot(3X) ..... graphics interface subroutines
putspent(3X) ..... write shadow password file entry
regcmp(3X): regcmp, regex ..... compile and execute regular expression
sputl(3X): sputl, sgetl ..... access long integer data in a machine-independent fashion
utadduser(3X): ut_add_user, ut_delete_user ..... add/delete users in utmp file

4 File Formats

intro(4) ..... introduction to file formats
a.out(4) ..... common assembler and link editor output
acct(4) ..... per-process accounting file format
addusr_def(4) ..... add user defaults
alias(4) ..... alias file for FACE
ar(4) ..... see *C and Language Tools Binder*
backup_codes(4) ..... backup cycle code definitions
backup_codes1(4) ..... backup subcode definitions
backup_dev(4) ..... backup devices
backup_info(4) ..... backup set definitions
cftime(4) ..... language specific strings
checklist(4) ..... list of filesystems processed by fsck and ncheck
constab(4) ..... constructed devices table
core(4) ..... format of core image file
cpio(4) ..... format of cpio archive
cronlog(4) ..... cron history file
crontabs(4) ..... clock daemon file
devtab(4) ..... virtual device table

dir(4) ..... format of directories
dirent(4) ..... file system independent directory entry
diskinfo(4) ..... disk device characteristic database
dump(4): dump, dumpdates ..... incremental dump format
env(4): .environ, .pref, .variables ..... FACE files
errorlog(4) ..... internal error log file
errormsg(4) ..... error message text file
filehdr(4) ..... file header for common object files
fs(4): fs, inode ..... format of file system volume
fspec(4) ..... format specification in text files
fstab(4) ..... file-system-table
gettydefs(4) ..... speed and terminal settings used by getty
gps(4): gps ..... graphical primitive string, format of graphical files
group(4) ..... group file
helplog(4) ..... monitor help actions
init.d(4) ..... system level transition files directory
inittab(4) ..... script for the init process
issue(4) ..... issue identification file
ldfcn(4) ..... common object file access routines
limits(4) ..... file header for implementation-specific constants
linenum(4) ..... line number entries in a common object file
loginlog(4) ..... log of failed login attempts
lptypes(4) ..... table of printer types supported by the menu system
mnttab(4) ..... mounted filesystem table
mod_config(4) ..... format of a module configuration file
motd(4) ..... message-of-the-day file
ott(4) ..... files that hold object architecture information
passwd(4) ..... password file
plot(4) ..... graphics interface
product(4) ..... installed products file
profile(4) ..... setting up an environment at login time
proto(4) ..... prototype job file for at
rc(4) ..... reboot and shutdown command scripts
rcsfile(4) ..... format of RCS file
reloc(4) ..... relocation information for a common object file
sccsfile(4) ..... format of SCCS file
scnhdr(4) ..... section header for a common object file
scr_dump(4) ..... format of curses screen image file
shadow(4) ..... restricted password file
shells(4) ..... shell table
shutdown(4) ..... system shutdown information file
spellhist(4) ..... unmatched spelling words file
sulog(4) ..... log of substitute user events
syms(4) ..... common object file symbol table format
sys_config(4) ..... format of a system configuration file
syskeymap(4) ..... identify keys to internal menu system functions
task(4) ..... global security table
term(4) ..... format of compiled term file
termcap(4) ..... terminal capability data base
terminfo(4) ..... terminal capability database

timezone(4) ..... set default system time zone
unistd(4) ..... file header for symbolic constants
utmp(4): utmp, wtmp ..... utmp and wtmp entry formats
uucp(4) ..... networking support files

5 Miscellaneous

intro(5) ..... introduction to miscellany
ascii(5) ..... map of ASCII character set
environ(5) ..... user environment
fcntl(5) ..... file control options
math(5) ..... math functions and constants
prof(5) ..... profile within a function
regexp(5) ..... regular expression compile and match routines
stat(5) ..... data returned by stat system call
term(5) ..... conventional names for terminals
types(5) ..... primitive system data types
values(5) ..... machine-dependent values
varargs(5) ..... handle variable argument list

7 Special Files

intro(7) ..... introduction to special files
clone(7) ..... open any minor device on a STREAMS driver
cmpt(7) ..... compatibility partitioning driver
cn(7) ..... pseudoconsole driver
consmem(7): consmem, sm, ss ..... console memory drivers
console(7) ..... console interface
kmmem(7): kMBmem, kMWmem ..... MULITBUS memory
ld0(7) ..... STREAMS tty line discipline
log(7) ..... interface to STREAMS error logging and event tracing
lp(7) ..... Systech parallel line printer interface
lpmod(7) ..... parallel printer canonical processing module
mbad(7) ..... MULITBUS adapter
mem(7): mem, kmem ..... core memory
mtio(7) ..... UNIX magnetic tape interface
null(7) ..... the null file
pmap(7) ..... map physical addresses into processs virtual address space
sa(7) ..... devices administered by System Administration
sced(7) ..... SCSI/Ethernet/Diagnostics board
sd(7) ..... SCSI disk controller
sp(7) ..... STREAMS pipe device
ssm(7) ..... System Services Module
st(7) ..... Systech MTI-800/1600/1650 terminal multiplexor driver
streamio(7) ..... STREAMS ioctl commands
streampty(7) ..... STREAMS pseudo terminal driver
streamtty(7) ..... Generic STREAMS tty driver interface
termio(7) ..... general terminal interface
tg(7) ..... SCSI 1/2-inch magnetic tape interface

timod(7) ..... Transport Interface cooperating STREAMS module
tirdwr(7) ..... Transport Interface read/write interface STREAMS module
tm(7) ..... SCSI tape controller
ts(7) ..... SCSI tape controller
tty(7) ..... controlling terminal (gentty) interface
usclk(7) ..... 32-bit microsecond counter
vtoc(7) ..... VTOC partitioning driver
wd(7) ..... SSM SCSI disk controller
wp(7) ..... SSM parallel line printer interface
xs(7) ..... Xylogics 781 terminal multiplexor driver
xt(7) ..... Xylogics 472 MULITBUS magnetic tape interface
zd(7) ..... dual-channel disk controller

8 Stand-Alone Utilities

intro(8) ..... introduction to system standalone utilities
ccsformat(8) ..... standalone embedded SCSI disk formatter
boot(8) ..... run a standalone program or bring up the operating system
copy(8) ..... perform a standalone data transfer
dump(8) ..... standalone memory dump
prtvtoc(8) ..... print the VTOC of a standalone disk device
zdformat(8) ..... standalone disk formatter for dual-channel disk controller

Permuted Index

st: Systech MTI-800/1600/	1650 terminal multiplexor driver.....st(7)
hpio: Hewlett-Packard	2645A terminal tape file archiver.....hpio(1)
of DASI 300 and 300s terminals	300, 300s: handle special functions.....300(1)
handle special functions of DASI	300 and 300s terminals.....300(1)
DASI 300 and 300s terminals 300,	300s: handle special functions of.....300(1)
special functions of DASI 300 and	300s terminals.....300(1)
usclk:	32-bit microsecond counter.....usclk(7)
dis386: disassemble Series	386 instructions.....dis386(3SEQ)
DASI 450 terminal	450: handle special functions of the.....450(1)
handle special functions of the DASI	450 terminal.....450(1)
xt: Xylogics	472 MULITBUS magnetic tape interface.....xt(7)
between long integer and base-	64 ASCII string.....a64l(3C)
xs: Xylogics	781 terminal multiplexor driver.....xs(7)
integer and base-64 ASCII string	a64l, l64a: convert between long.....a64l(3C)
	abort: generate an IOT fault.....abort(3C)
	abs: return integer absolute value.....abs(3C)
abs: return integer	absolute value.....abs(3C)
fabs: floor, ceiling, remainder,	absolute value functions.....floor(3M)
t_accept:	accept a connect request.....t_accept(3N)
requests	accept, reject: allow or prevent LP.....accept(1M)
utime: set file	access and modification times.....utime(2)
touch: update	access and modification times of a file.....touch(1)
file	access: determine accessibility of a.....access(2)
initgroups: initialize group	access list.....initgroups(3SEQ)
setgroups: set group	access list.....setgroups(2SEQ)
machine-independent/ sputl, sgetl:	access long integer data in a.....sputl(3X)
ldfcn: common object file	access routines.....ldfcn(4)
endutent, utmpname, getutpid:	access utmp file entry.....getut(3C)
access: determine	accessibility of a file.....access(2)
acct: enable or disable process	accounting.....acct(2)
acctcon1, acctcon2: connect-time	accounting.....acctcon(1M)
acctprc1, acctprc2: process	accounting.....acctprc(1M)
turnacct: shell procedures for	accounting.....acctsh(1M)
runacct: run daily	accounting.....runacct(1M)
/accton, acctwtmp: overview of	accounting and miscellaneous/.....acct(1M)
of accounting and miscellaneous	accounting commands.....acct(1M)
diskusg: generate disk	accounting data by user ID.....diskusg(1M)
acct: per-process	accounting file format.....acct(4)
acctcom: search and print process	accounting files.....acctcom(1)
acctmerg: merge or add total	accounting files.....acctmerg(1M)
command summary from per-process	accounting records.....acctcms(1M)
fwtmp, wtmpfix: manipulate connect	accounting records.....fwtmp(1M)
accounting	acct: enable or disable process.....acct(2)
format	acct: per-process accounting file.....acct(4)
per-process accounting records	acctcms: command summary from.....acctcms(1M)
accounting files	acctcom: search and print process.....acctcom(1)
accounting	acctcon1, acctcon2: connect-time.....acctcon(1M)

acctcon1, acctcon2: connect-time accounting ..... acctcon(1M)
acctwtmp: overview of accounting/ acctdisk, acctdusg, accton, ... acct(1M)
of accounting and/ acctdisk, acctdusg, accton, acctwtmp: overview ..... acct(1M)
accounting files acctmerg: merge or add total ..... acctmerg(1M)
acctdisk, acctdusg, accton, acctwtmp: overview of accounting and/ ..... acct(1M)
accounting acctprc1, acctprc2: process ..... acctprc(1M)
acctprc1, acctprc2: process accounting ..... acctprc(1M)
acctdisk, acctdusg, accton, acctwtmp: overview of accounting and/ ..... acct(1M)
release indication t_rcvrel: acknowledge receipt of an orderly ..... t_rcvrel(3N)
trig: sin, cos, tan, asin, acos, atan, atan2: trigonometric/ ..... trig(3M)
helplog: monitor help actions ..... helplog(4)
sigaction: examine and change signal actions ..... sigaction(2)
killall: kill all active processes ..... killall(1M)
offline_all: take all but one active processor off line ..... offline_all(3SEQ)
mbad: MULITBUS adapter ..... mbad(7)
addcmdopt: add a command option to a menu ..... addcmdopt(1M)
menu addformopt: add a form/SELECT window option to a ..... addformopt(1M)
addmenuopt: add a menu option to a menu ..... addmenuopt(1M)
acctmerg: merge or add total accounting files ..... acctmerg(1M)
addusr.def: add user defaults ..... addusr_def(4)
putenv: change or add value to environment ..... putenv(3C)
menu addcmdopt: add a command option to a ..... addcmdopt(1M)
ut_add_user, ut_delete_user: add/delete users in utmp file ..... utadduser(3X)
option to a menu addformopt: add a form/SELECT window ... addformopt(1M)
menu addmenuopt: add a menu option to a ..... addmenuopt(1M)
hasmntopt, setmntent: get file/ addmntent, endmntent, getmntent, ... getmntent(3SEQ)
map an open file into the processs address space ..... mmap(2SEQ)
munmap: unmap part of the processs address space ..... munmap(2SEQ)
addresses into processs virtual address space ..... pmap(7)
t_bind: bind an address to a transport endpoint ..... t_bind(3N)
pmap: map physical addresses into processs virtual address space ..... pmap(7)
addusr.def: add user defaults ..... addusr_def(4)
files admin: create and administer SCCS ..... admin(1)
print service lpfilter: administer filters used with the LP ..... lpfilter(1M)
print service lpforms: administer forms used with the LP ..... lpforms(1M)
admin: create and administer SCCS files ..... admin(1)
SA: devices administered by System ..... sa(7)
SA: devices administered by System Administration ..... sa(7)
sysadm: menu interface to do system administration ..... sysadm(1)
uadmin: administrative control ..... uadmin(1M)
uadmin: administrative control ..... uadmin(2)
swap: swap administrative interface ..... swap(1M)
/usr/adm/loginlog: log of failed login attempts ..... loginlog(4)
/onproc: run a command with priority aging, PFF, swapping, or processor/ ..... noage(1M)
alarm: set a process alarm clock ..... alarm(2)
alarm: set a process alarm clock ..... alarm(2)
/display application specific alarms and/or the working indicator ..... indicator(1F)
pathalias: alias file for FACE ..... alias(4)
valloc: aligned memory allocator ..... valloc(3SEQ)
t_alloc: allocate a library structure ..... t_alloc(3N)
brk, sbrk: change data segment space allocation ..... brk(2)
free, realloc, calloc: main memory allocator ..... malloc(3C)

mallopt, mallinfo: fast main memory allocator.....malloc(3X)
shrealloc, shfree: shared memory allocator.....shmalloc(3PPS)
valloc: aligned memory allocator.....valloc(3SEQ)
editor output a.out: common assembler and link .....a.out(4)
to maintenance commands and application programs.....intro(1)
the working/ indicator: display application specific alarms and/or.....indicator(1F)
lookup apropos: locate commands by keyword.....apropos(1)
ar: archive file format.....see *C and Lang Tools Binder*
and libraries ar: maintain archives.....see *C and Lang Tools Binder*
bc: arbitrary-precision arithmetic language .....bc(1)
ott: files that hold object architecture information.....ott(4)
cpio: format of cpio archive .....cpio(4)
the archive header of a member of an archive file.....ldahread(3X)
ar: common archive file format .....see *C and Lang Tools Binder*
archive file ldahread: read the archive header of a member of an .....ldahread(3X)
2645A terminal tape file archiver.....hpio(1)
tar: tape file archiver.....tar(1)
cpio: copy file archives in and out.....cpio(1)
varargs: handle variable argument list .....varargs(5)
print formatted output of a varargs argument list .....vprintf(3S)
xargs: construct argument list(s) and execute command.....xargs(1)
getopt: get option letter from argument vector.....getopt(3C)
echo: echo arguments.....echo(1)
expr: evaluate arguments as an expression .....expr(1)
message: puts its arguments on message line .....message(1F)
bc: arbitrary-precision arithmetic language .....bc(1)
asa: interpret ASA carriage control characters.....asa(1)
characters asa: interpret ASA carriage control.....asa(1)
/localtime, gmtime, asctime, cftime, ascftime, mktime, tzset: convert/.....ctime(3C)
ascii: map of ASCII character set .....ascii(5)
ascii: map of ASCII character set.....ascii(5)
between long integer and base-64 ASCII string.....a64l(3C)
ctime, localtime, gmtime, asctime, cftime, ascftime, mktime, tzset:/ .....ctime(3C)
trig: sin, cos, tan, asin, acos, atan, atan2: trigonometric/.....trig(3M)
as: common assembler.....see *C and Lang Tools Binder*
a.out: common assembler and link editor output.....a.out(4)
assert: verify program assertion.....assert(3X)
assert: verify program assertion.....assert(3X)
system commands assist: assistance using operating.....assist(1)
/program for generating/modifying ASSIST menus or command forms.....astgen(1)
assist: assistance using operating system commands .....assist(1)
generating/modifying ASSIST menus/ astgen: program for.....astgen(1)
later time at, batch: execute commands at a .....at(1)
trig: sin, cos, tan, asin, acos, atan, atan2: trigonometric functions.....trig(3M)
/sin, cos, tan, asin, acos, atan, atan2: trigonometric functions.....trig(3M)
double-precision number strtod, atof: convert string to.....strtod(3C)
strtol, atol, atoi: convert string to integer.....strtol(3C)
strtol, atol, atoi: convert string to integer.....strtol(3C)
change login password and password attributes.....passwd(1)
control over various process attributes.....proc_ctl(2SEQ)
rcs: change RCS file attributes.....rcs(1)
dumpconf: dump kernel auto-configuration device table.....dumpconf(1M)

language awk: pattern scanning and processing.....awk(1)
backup.codes: backup cycle code definitions.....backup_codes(4)
backup.dev: backup devices.....backup_dev(4)
ckbupscd: check filesystem backup schedule.....ckbupscd(1M)
backup.info: backup set definitions.....backup_info(4)
backup.codes1: backup subcode definitions.....backup_codes1(4)
definitions backup.codes: backup cycle code.....backup_codes(4)
definitions backup.codes1: backup subcode.....backup_codes1(4)
backup.dev: backup devices.....backup_dev(4)
backup.info: backup set definitions.....backup_info(4)
banner: make posters.....banner(1)
m_sync: check in at barrier.....m_sync(3PPS)
initialize barrier, wait at barrier.....s_wait_barrier(3PPS)
/s_wait_barrier: initialize barrier, wait at barrier.....s_wait_barrier(3PPS)
of pathnames basename, dirname: deliver portions.....basename(1)
at, batch: execute commands at a later time.....at(1)
language bc: arbitrary-precision arithmetic.....bc(1)
procedures bcheckrc: system initialization.....bcheckrc(1M)
bdflush: update daemon.....bdflush(1M)
bdflush: update daemon.....bdflush(2SEQ)
bdiff: big diff.....bdiff(1)
bessel: j0, j1, jn, y0, y1, yn: Bessel functions.....bessel(3M)
Bessel functions bessel: j0, j1, jn, y0, y1, yn:.....bessel(3M)
bfs: big file scanner.....bfs(1)
fread, fwrite: binary input/output.....fread(3S)
bp: binary patcher.....bp(1M)
bsearch: binary search a sorted table.....bsearch(3C)
tfind, tdelete, twalk: manage binary search trees.....tsearch(3C)
tmp_affinity: bind a process to a processor.....tmp_affinity(2SEQ)
t_bind: bind an address to a transport endpoint.....t_bind(3N)
sum: print checksum and block count of a file.....sum(1)
prtvtoc: print the VTOC of a block device.....prtvtoc(1M)
sigprocmask: examine and change blocked signals.....sigprocmask(2)
number of blocks and indirect blocks.....nblocks(3SEQ)
nblocks: calculate number of blocks and indirect blocks.....nblocks(3SEQ)
df: report number of free disk blocks and inodes.....df(1M)
consistency check based on the BOM.....bomverify(1M)
consistency check based on the BOM bomverify: perform an installation.....bomverify(1M)
bootflags: display/change system boot flags.....bootflags(1M)
installboot: installs a boot program on a disk.....installboot(1M)
bring up the operating system boot: run a standalone program or.....boot(8)
boot flags bootflags: display/change system.....bootflags(1M)
bp: binary patcher.....bp(1M)
allocation brk, sbrk: change data segment space.....brk(2)
segment size brk, sbrk: change private data.....brk(3PPS)
table bsearch: binary search a sorted.....bsearch(3C)
stdio: standard buffered input/output package.....stdio(3S)
setbuf, setvbuf: assign buffering to a stream.....setbuf(3S)
mailbug: submit a bug report or enhancement request.....mailbug(1)
devbuild, devdestroy: build and destroy virtual devices.....devbuild(1M)
sccstorcs: build RCS file from SCCS file.....sccstorcs(1)
mknod: build special file.....mknod(1M)

config: build system configuration files.....config(1M)
swab: swap bytes.....swab(3C)
cc: C compiler.....see *C and Lang Tools Binder*
cflow: generate C flowgraph.....see *C and Lang Tools Binder*
cpp: the C language preprocessor.....see *C and Lang Tools Binder*
cb: C program beautifier.....see *C and Lang Tools Binder*
lint: a C program checker.....see *C and Lang Tools Binder*
cxref: generate a C program cross-reference.....see *C and Lang Tools Binder*
ctrace: C program debugger.....see *C and Lang Tools Binder*
list: produce C source listing.....see *C and Lang Tools Binder*
cal: print calendar.....cal(1)
indirect blocks nblocks: calculate number of blocks and.....nblocks(3SEQ)
dc: desk calculator.....dc(1)
cal: print calendar.....cal(1)
calendar: reminder service.....calendar(1)
stat: data returned by stat system call.....stat(5)
syscall: indirect system call.....syscall(2)
cu: call another UNIX system.....cu(1C)
malloc, free, realloc, calloc: main memory allocator.....malloc(3C)
malloc, free, realloc, calloc, mallopt, mallinfo: fast main memory/.....malloc(3X)
introduction to Sequent system calls.....intro(2SEQ)
intro: introduction to system calls and error numbers.....intro(2)
lp, cancel: send/cancel requests to the LP print service.....lp(1)
LP print service lp, cancel: send/cancel requests to the.....lp(1)
lpmod: parallel printer canonical processing module.....lpmod(7)
description into a terminfo/ captoinfo: convert a termcap.....captoinfo(1M)
asa: interpret ASA carriage control characters.....asa(1)
cat: concatenate and print files.....cat(1)
cb: C program beautifier.....see *C and Lang Tools Binder*
gencc: create a front-end to the cc command.....gencc(1M)
cc: C compiler.....see *C and Lang Tools Binder*
disk formatter CCSformat: standalone embedded SCSI.....ccsformat(8)
cd: change working directory.....cd(1)
an SCCS delta cdc: change the delta commentary of.....cdc(1)
remainder, absolute value/ floor, ceil, fmod, fabs: floor, ceiling,.....floor(3M)
floor, ceil, fmod, fabs: floor, ceiling, remainder, absolute value/.....floor(3M)
cflow: generate C flowgraph...see *C and Lang Tools Binder*
ctime, localtime, gmtime, asctime, cftime, ascftime, mktime, tzset:/.....ctime(3C)
cftime: language specific strings.....cftime(4)
sigprocmask: examine and change blocked signals.....sigprocmask(2)
brk, sbrk: change data segment space allocation.....brk(2)
attributes passwd: change login password and password.....passwd(1)
chmod: change mode.....chmod(1)
chmod: change mode of file.....chmod(2)
putenv: change or add value to environment.....putenv(3C)
chown: change owner and group of a file.....chown(2)
chown, chgrp: change owner or group.....chown(1)
nice: change priority of a process.....nice(2)
brk, sbrk: change private data segment size.....brk(3PPS)
rcs: change RCS file attributes.....rcs(1)
chroot: change root directory.....chroot(2)
chroot: change root directory for a command.....chroot(1M)

shbrk, shsbrk: change shared data segment size ..... shbrk(3PPS)
sigaction: examine and change signal actions ..... sigaction(2)
bootflags: display/ change system boot flags ..... bootflags(1M)
shutdown: shut down system, change system state ..... shutdown(1M)
cdc: change the delta commentary of an SCCS delta ..... cdc(1)
newform: change the format of a text file ..... newform(1)
helpadm: change the Help Facility database ..... helpadm(1M)
rename: change the name of a file ..... rename(2)
delta: make a delta (change) to an SCCS file ..... delta(1)
parameters vm_ctl: examine and change virtual memory tuning ..... vm_ctl(2SEQ)
cd: change working directory ..... cd(1)
chdir: change working directory ..... chdir(2)
pipe: create an interprocess channel ..... pipe(2)
zd: dual- channel disk controller ..... zd(7)
standalone disk formatter for dual- channel disk controller ..... zdformat(8)
strstr: find first occurrence of character ..... strstr(3C)
ungetc: push character back into input stream ..... ungetc(3S)
conversion tables chrtbl: generate character classification and ..... chrtbl(1M)
_tolower, _toupper, setchrclass: character handling ..... ctype(3C)
cuserid: get character login name of the user ..... cuserid(3S)
getc, getchar, fgetc, getw: get character or word from a stream ..... getc(3S)
putc, putchar, fputc, putw: put character or word on a stream ..... putc(3S)
ascii: map of ASCII character set ..... ascii(5)
fgrep: search a file for a character string ..... fgrep(1)
diskinfo: disk device characteristic database ..... diskinfo(4)
asa: interpret ASA carriage control characters ..... asa(1)
_tolower, toascii: translate characters ..... conv(3C)
tr: translate characters ..... tr(1)
lastlogin, monacct, nulladm,/ chargefee, ckpacct, dodisk, ... acctsh(1M)
chdir: change working directory ..... chdir(2)
filesystem directory consistency check ..... dcheck(1M)
filesystem storage consistency check ..... icheck(1M)
checkfsys: check a filesystem ..... checkfsys(1M)
fsck: filesystem consistency check and interactive repair ..... fsck(1M)
perform an installation consistency check based on the BOM ..... bomverify(1M)
checkeq: check documents for correct eqn syntax ..... checkeq(1)
ckbupscd: check filesystem backup schedule ..... ckbupscd(1M)
m_sync: check in at barrier ..... m_sync(3PPS)
ci: check in RCS revisions ..... ci(1)
co: check out RCS revisions ..... co(1)
permissions file uucheck: check the uucp directories and ..... uucheck(1M)
eqn syntax. checkeq: check documents for correct ..... checkeq(1)
checkfsys: check a filesystem ..... checkfsys(1M)
processed by fsck and ncheck checklist: list of filesystems ..... checklist(4)
sum: print checksum and block count of a file ..... sum(1)
chown, chgrp: change owner or group ..... chown(1)
/m_rele_procs: suspend and resume child process execution ..... m_park_procs(3PPS)
times: get process and child process times ..... times(2)
wait, waitpid: wait for a child process to stop or terminate ..... wait(2)
m_get_numprocs: return number of child processes ..... m_get_numprocs(3PPS)
m_kill_procs: kill child processes ..... m_kill_procs(3PPS)
m_set_procs: set number of child processes ..... m_set_procs(3PPS)

chmod: change mode ..... chmod(1)
chmod: change mode of file ..... chmod(2)
file chown: change owner and group of a ..... chown(2)
chown, chgrp: change owner or group ..... chown(1)
chroot: change root directory ..... chroot(2)
command chroot: change root directory for a ..... chroot(1M)
classification and conversion/ chrtbl: generate character ..... chrtbl(1M)
ci: check in RCS revisions ..... ci(1)
schedule ckbupscd: check filesystem backup ..... ckbupscd(1M)
nulladm, prctmp,/ chargefee, ckpacct, dodisk, lastlogin, monacct, ... acctsh(1M)
chrtbl: generate character classification and conversion tables ..... chrtbl(1M)
clri: clear inode ..... clri(1M)
ferror, feof, clearerr, fileno: stream status inquiries ..... ferror(3S)
a shell (command interpreter) with C-like syntax ..... csh(1)
alarm: set a process alarm clock ..... alarm(2)
usclk_init: get/init microsecond clock ..... getusclk(3SEQ)
cron: clock daemon ..... cron(1M)
crontabs: clock daemon file ..... crontabs(4)
clock: report CPU time used ..... clock(3C)
STREAMS driver clone: open any minor device on a ..... clone(7)
ldclose, ldaclose: close a common object file ..... ldclose(3X)
close: close a file descriptor ..... close(2)
t_close: close a transport endpoint ..... t_close(3N)
close: close a file descriptor ..... close(2)
fclose, fflush: close or flush a stream ..... fclose(3S)
/telldir, seekdir, rewinddir, closedir: directory operations ..... directory(3X)
clri: clear inode ..... clri(1M)
cmp: compare two files ..... cmp(1)
driver cmpt: compatibility partitioning ..... cmpt(7)
cn: pseudoconsole driver ..... cn(7)
co: check out RCS revisions ..... co(1)
communicate to a/ cocreate, cosend, cocheck, coreceive, codestroy: ..... coproc(1F)
coreceive, codestroy: communicate/ cocreate, cosend, cocheck, ... coproc(1F)
backup.codes: backup cycle code definitions ..... backup_codes(4)
m_multi: mark single-process code section ..... m_single(3PPS)
/cosend, cocheck, coreceive, codestroy: communicate to a process ..... coproc(1F)
col: filter reverse line-feeds ..... col(1)
to form error log dmesg: collect system diagnostic messages ..... dmesg(1M)
setcolor: redefine or create a color ..... setcolor(1F)
comb: combine SCCS deltas ..... comb(1)
comb: combine SCCS deltas ..... comb(1)
to two sorted files comm: select or reject lines common ..... comm(1)
mcs: manipulate the object file comment section ..... mcs(1)
cdc: change the delta commentary of an SCCS delta ..... cdc(1)
a.out: common assembler and link editor output ..... a.out(4)
ldclose, ldaclose: close a common object file ..... ldclose(3X)
ldfhread: read the file header of a common object file ..... ldfhread(3X)
number entries of a section of a common object file ..... ldlseek(3X)
to the optional file header of a common object file ..... ldohseek(3X)
relocation entries of a section of a common object file ..... ldrseek(3X)
indexednamed section header of a common object file ..... ldshread(3X)
to an indexednamed section of a common object file ..... ldsseek(3X)

index of a symbol table entry of a common object file ..... ldtbindex(3X)
an indexed symbol table entry of a common object file ..... ldtbread(3X)
seek to the symbol table of a common object file ..... ldtbseek(3X)
linenum: line number entries in a common object file ..... linenum(4)
reloc: relocation information for a common object file ..... reloc(4)
scnhdr: section header for a common object file ..... scnhdr(4)
ldfcn: common object file access routines ..... ldfcn(4)
ldopen, ldaopen: open a common object file for reading ..... ldopen(3X)
/manipulate line number entries of a common object file function ..... ldlread(3X)
ldgetname: retrieve symbol name for common object file symbol table/ ..... ldgetname(3X)
syms: common object file symbol table format ..... syms(4)
filehdr: file header for common object files ..... filehdr(4)
symbols glossary: definitions of common operating system terms and ..... glossary(1)
comm: select or reject lines common to two sorted files ..... comm(1)
cocheck, coreceive, codestroy: communicate to a process ..... coproc(1F)
ipcs: report inter-process communication facilities status ..... ipcs(1)
stdipc: ftok: standard interprocess communication package ..... stdipc(3C)
tfipc: interprocess communication reporter ..... tfipc(1M)
descriptions infocmp: compare or print out terminfo ..... infocmp(1M)
rcsdiff: compare RCS revisions ..... rcsdiff(1)
cmp: compare two files ..... cmp(1)
sccsdiff: compare two versions of an SCCS file ..... sccsdiff(1)
diff: differential file comparison ..... diff(1)
diff3: 3-way differential file comparison ..... diff3(1)
dircmp: directory comparison ..... dircmp(1)
cmpt: compatibility partitioning driver ..... cmpt(7)
regcmp: regular expression compile ..... regcmp(1)
regcmp, regex: compile and execute regular expression ..... regcmp(3X)
regexp: regular expression compile and match routines ..... regexp(5)
term: format of compiled term file ..... term(4)
cc: C compiler ..... see *C and Lang Tools Binder*
ec: forms compiler ..... ec(1M)
tic: terminfo compiler ..... tic(1M)
yacc: yet another compiler- compiler ..... yacc(1)
yacc: yet another compiler-compiler ..... yacc(1)
erf, erfc: error function and complementary error function ..... erf(3M)
cprs: compress an object file ..... see *C and Lang Tools Binder*
compress, uncompress, zcat: compress and expand data ..... compress(1)
pack, pcat, unpack: compress and expand files ..... pack(1)
and expand data compress, uncompress, zcat: compress ..... compress(1)
entry of a common object/ ldtbindex: compute the index of a symbol table ..... ldtbindex(3X)
cat: concatenate and print files ..... cat(1)
test: condition evaluation command ..... test(1)
files config: build system configuration ..... config(1M)
pathconf, fpathconf: get configurable pathname variables ..... pathconf(2)
sysconf: get configurable system variables ..... sysconf(2)
dumpconf: dump kernel auto- configuration device table ..... dumpconf(1M)
mod_config: format of a module configuration file ..... mod_config(4)
sys_config: format of a system configuration file ..... sys_config(4)
config: build system configuration files ..... config(1M)
showcfg: show the configuration of the machine ..... showcfg(1M)
lpadmin: configure the LP print service ..... lpadmin(1M)

t_rcvconnect: receive the — confirmation from a connect request ..... t_rcvconnect(3N)
fwtmp, wtmpfix: manipulate — connect accounting records ..... fwtmp(1M)
t_accept: accept a — connect request ..... t_accept(3N)
t_listen: listen for a — connect request ..... t_listen(3N)
receive the confirmation from a — connect request ..... t_rcvconnect(3N)
establish an out-going terminal line — connection ..... dial(3C)
data or expedited data sent over a — connection ..... t_rcv(3N)
send data or expedited data over a — connection ..... t_snd(3N)
t_connect: establish a — connection with another transport user ..... t_connect(3N)
acctcon1, acctcon2: — connect-time accounting ..... acctcon(1M)
dcheck: filesystem directory — consistency check ..... dcheck(1M)
icheck: filesystem storage — consistency check ..... icheck(1M)
fsck: filesystem — consistency check and interactive repair ..... fsck(1M)
bomverify: perform an installation — consistency check based on the BOM ..... bomverify(1M)
quotacheck: filesystem quota — consistency checker ..... quotacheck(1M)
drivers — consmem, sm, ss: console memory ..... consmem(7)
— console: console interface ..... console(7)
console: — console interface ..... console(7)
consmem, sm, ss: — console memory drivers ..... consmem(7)
— constab: constructed devices table ..... constab(4)
read_constab: read — constab entries into a structure ..... read_constab(3SEQ)
header for implementation-specific — constants ..... limits(4)
math: math functions and — constants ..... math(5)
unistd: file header for symbolic — constants ..... unistd(4)
constab: — constructed devices table ..... constab(4)
deroff: remove nroff, tbl, and eqn — constructs ..... deroff(1)
Uutry: try to — contact remote system with debugging on ..... uutry(1M)
ls: list — contents of directory ..... ls(1)
sigcontext: get signal — context ..... sigcontext(2SEQ)
set and/or get signal stack — context ..... sigstack(2SEQ)
csplit: — context split ..... csplit(1)
sd: SCSI disk — controller ..... sd(7)
tm: SCSI tape — controller ..... tm(7)
ts: SCSI tape — controller ..... ts(7)
wd: SSM SCSI disk — controller ..... wd(7)
zd: dual-channel disk — controller ..... zd(7)
disk formatter for dual-channel disk — controller ..... zdformat(8)
synchronize a co-process with the — controlling FMLI object ..... vsig(1F)
virtually hangup the specified — controlling terminal ..... fvhangup(2SEQ)
tty: — controlling terminal (gentty) interface ..... tty(7)
_tolower, toascii: translate/ — conv: toupper, tolower, _toupper, ... conv(3C)
units: — conversion program ..... units(1)
character classification and — conversion tables ..... chrtbl(1M)
terminfo description . captoinfo: — convert a termcap description into a ..... captoinfo(1M)
dd: — convert and copy a file ..... dd(1M)
long integers l3tol, ltol3: — convert between 3-byte integers and ..... l3tol(3C)
base-64 ASCII string a64l, l64a: — convert between long integer and ..... a64l(3C)
dev_to_ttyname, ttyname_to_dev: — convert major/minor numbers ..... dev_to_ttyname(3C)
/cftime, ascftime, mktime, tzset: — convert date and time to string ..... ctime(3C)
ecvt, fcvt, gcvt: — convert floating-point number to string ..... ecvt(3C)
scanf, fscanf, sscanf: — convert formatted input ..... scanf(3S)
strtod, atof: — convert string to double-precision number ..... strtod(3C)

strtol, atol, atoi: convert string to integer ..... strtol(3C)
password file scheme pwunconv: converts from a two to a one ..... pwunconv(1M)
vsig: synchronize a co-process with the controlling FMLI object ..... vsig(1F)
UNIX-to-UNIX system copy ..... uucp(1C)
system file copy ..... uuto(1C)
dd: convert and copy a file ..... dd(1M)
cpset: copy file and set mode, user, and group IDs ..... cpset(1)
cpio: copy file archives in and out ..... cpio(1)
cp, ln, mv: copy, link or move files ..... cp(1)
volcopy: make literal copy of filesystem ..... volcopy(1M)
transfer copy: perform a standalone data ..... copy(8)
savecore: save a core dump of the operating system ..... savecore(1M)
core: format of core image file ..... core(4)
core: format of core image file ..... core(4)
mem, kmem: core memory ..... mem(7)
and unset environment variables in core or in files ..... set(1F)
cocreate, cosend, cocheck, coreceive, codestroy: communicate to a/ ..... coproc(1F)
checkeq: check documents for correct eqn syntax ..... checkeq(1)
trigonometric functions trig: sin, cos, tan, asin, acos, atan, atan2: ..... trig(3M)
codestroy: communicate to/ cocreate, cosend, cocheck, coreceive, ... coproc(1F)
sinh, cosh, tanh: hyperbolic functions ..... sinh(3M)
wc: word count ..... wc(1)
sum: print checksum and block count of a file ..... sum(1)
m_next: increment global counter ..... m_next(3PPS)
usclk: 32-bit microsecond counter ..... usclk(7)
cp, ln, mv: copy, link or move files ..... cp(1)
cpio: format of cpio archive ..... cpio(4)
cpio: copy file archives in and out ..... cpio(1)
cpio: format of cpio archive ..... cpio(4)
cpp: C preprocessor ..... see *C and Lang Tools Binder*
cprs: compress object file ..... see *C and Lang Tools Binder*
and group IDs cpset: copy file and set mode, user, ... cpset(1)
clock: report CPU time used ..... clock(3C)
cpus_online: returns the number of CPUs on-line ..... cpus_online(3PPS)
CPUs on-line cpus_online: returns the number of ..... cpus_online(3PPS)
crash: examine system images ..... crash(1M)
an existing one creat: create a new file or rewrite ..... creat(2)
setcolor: redefine or create a color ..... setcolor(1F)
gencc: create a front-end to the cc command ..... gencc(1M)
tmpnam, tempnam: create a name for a temporary file ..... tmpnam(3S)
creat: create a new file or rewrite an existing one ..... creat(2)
fork: create a new process ..... fork(2)
ctags: create a tags file ..... ctags(1)
tmpfile: create a temporary file ..... tmpfile(3S)
pipe: create an interprocess channel ..... pipe(2)
admin: create and administer SCCS files ..... admin(1)
tasktbl: create security table in shared memory ..... tasktbl(1M)
setsid: create session and set process group ID ..... setsid(2)
pathconv: search Interpreter criteria for filename ..... pathconv(1F)
cron: clock daemon ..... cron(1M)
cronlog: cron history file ..... cronlog(4)
cronlog: cron history file ..... cronlog(4)

crontab: user — crontab file ..... crontab(1)
crontab: user crontab file ..... crontab(1)
crontabs: clock daemon file ..... crontabs(4)
cxref: generate C program — cross-reference ..... see *C and Lang Tools Binder*
pg: file perusal filter for — CRTs ..... pg(1)
crypt: encode/decode ..... crypt(1)
functions — crypt: password and file encryption ..... crypt(3X)
hashing encryption — crypt, setkey, encrypt: generate ..... crypt(3C)
with C-like syntax — csh: a shell (command interpreter) ..... csh(1)
csplit: context split ..... csplit(1)
ct: spawn getty to a remote terminal ..... ct(1C)
ctags: create a tags file ..... ctags(1)
terminal — ctermid: generate filename for ..... ctermid(3S)
cftime, ascftime, mktime, tzset:/ — ctime, localtime, gmtime, asctime, ... ctime(3C)
ctrace: C prog debugger ..... see *C and Lang Tools Binder*
cu: call another UNIX system ..... cu(1C)
scr_dump: format of — curses screen image file ..... scr_dump(4)
optimization package — curses: terminal screen handling and ..... curses(3X)
the user — cuserid: get character login name of ..... cuserid(3S)
line of a file — cut: cut out selected fields of each ..... cut(1)
cut: — cut out selected fields of each line of a file ..... cut(1)
cross-reference — cxref: generate C program ..... see *C and Lang Tools Binder*
backup.codes: backup — cycle code definitions ..... backup_codes(4)
bdflush: update — daemon ..... bdflush(1M)
bdflush: update — daemon ..... bdflush(2SEQ)
cron: clock — daemon ..... cron(1M)
strerr: STREAMS error logger — daemon ..... strerr(1M)
crontabs: clock — daemon file ..... crontabs(4)
creation pwdhd: — daemon to oversee password database ..... pwdhd(1M)
runacct: run — daily accounting ..... runacct(1M)
300s: handle special functions of — DASI 300 and 300s terminals ..... 300(1)
450: handle special functions of the — DASI 450 terminal ..... 450(1)
zcat: compress and expand — data ..... compress(1)
prof: display profile — data ..... prof(1)
time a command; report process — data and system activity ..... timex(1)
termcap: terminal capability — data base ..... termcap(4)
diskusg: generate disk accounting — data by user ID ..... diskusg(1M)
t_rcvuderr: receive a unit — data error indication ..... t_rcvuderr(3N)
sputl, sgetl: access long integer — data in a machine-independent/ ..... sputl(3X)
plock: lock process, text, or — data in memory ..... plock(2)
t_snd: send — data or expedited data over a connection ..... t_snd(3N)
t_rcv: receive — data or expedited data sent over a connection ..... t_rcv(3N)
t_snd: send data or expedited — data over a connection ..... t_snd(3N)
stat: — data returned by stat system call ..... stat(5)
brk, sbrk: change private — data segment size ..... brk(3PPS)
shbrk, shsbrk: change shared — data segment size ..... shbrk(3PPS)
brk, sbrk: change — data segment space allocation ..... brk(2)
t_rcv: receive data or expedited — data sent over a connection ..... t_rcv(3N)
copy: perform a standalone — data transfer ..... copy(8)
types: primitive system — data types ..... types(5)
t_rcvudata: receive a — data unit ..... t_rcvudata(3N)
t_sndudata: send a — data unit ..... t_sndudata(3N)

diskinfo: disk device characteristic database....diskinfo(4)
helpadm: change the Help Facility database....helpadm(1M)
terminfo: terminal capability database....terminfo(4)
a terminal or query terminfo database....tput(1)
pwdhd: daemon to oversee password database creation....pwdhd(1M)
mkpwdbm: make a database for fast password lookup....mkpwdbm(1M)
join: relational database operator....join(1)
dbm_clearerr, dmb_blkflush: database subroutines....ndbm(3X)
makewhatis: generate man page databases....makealiases(1M)
date: print and set the date....date(1)
ascftime, mktime, tzset: convert date and time to string....ctime(3C)
date: print and set the date....date(1)
/dbm_nextkey, dbm_error, dbm_clearerr, dmb_blkflush: databasc/....ndbm(3X)
dbm_delete,/ dbm_open, dbm_close, dbm_fetch, dbm_store,....ndbm(3X)
dbm_close, dbm_fetch, dbm_store, dbm_delete, dbm_firstkey,/....ndbm(3X)
/dbm_firstkey, dbm_nextkey, dbm_error, dbm_clearerr,/....ndbm(3X)
dbm_firstkey,/dbm_open, dbm_close, dbm_fetch, dbm_store, dbm_delete,....ndbm(3X)
/dbm_fetch, dbm_store, dbm_delete, dbm_firstkey, dbm_nextkey,/....ndbm(3X)
dbm_store, dbm_delete, dbm_firstkey, dbm_nextkey, dbm_error,/....ndbm(3X)
dbm_store, dbm_delete,/ dbm_open, dbm_close, dbm_fetch, ...ndbm(3X)
dbm_open, dbm_close, dbm_fetch, dbm_store, dbm_delete, dbm_firstkey,/....ndbm(3X)
pdbx, dbx: parallel debugger....see *C and Lang Tools Binder*
dc: desk calculator....dc(1)
enable/disable patrol seeks on DCC disks....sweepon(1M)
consistency check dcheck: filesystem directory....dcheck(1M)
dd: convert and copy a file....dd(1M)
ctrace: C program debugger....see *C and Lang Tools Binder*
pdbx, dbx: parallel debugger....see *C and Lang Tools Binder*
try to contact remote system with debugging on....uutry(1M)
crypt: encode/ decode....crypt(1)
timezone: set default system time zone....timezone(4)
reset the current form field to its default values....reset(1F)
addusr.def: add user defaults....addusr_def(4)
format disks or perform other defect management functions....format(1M)
gtty: set and get terminal state (defunct)....stty(2)
ut_add_user, ut_delete_user: add/ delete users in utmp file....utadduser(3X)
the delta commentary of an SCCS delta....cdc(1)
delta: make a delta (change) to an SCCS file....delta(1)
cdc: change the delta commentary of an SCCS delta....cdc(1)
rmdel: remove a delta from an SCCS file....rmdel(1)
SCCS file delta: make a delta (change) to an....delta(1)
comb: combine SCCS deltas....comb(1)
constructs deroff: remove nroff, tbl, and eqn....deroff(1)
whatis: describe what a command is....whatis(1)
close: close a file descriptor....close(2)
dup: duplicate an open file descriptor....dup(2)
dup2: duplicate an open file descriptor....dup2(2)
setmntent: get file system descriptor file entry....getmntent(3SEQ)
getdtablemax: get maximum in-use descriptor table index....getdtablemax(2SEQ)
getdtablesize: get descriptor table size....getdtablesize(2SEQ)
setdtablesize: set descriptor table size....setdtablesize(2SEQ)
dc: desk calculator....dc(1)

devbuild, devdestroy: build and destroy virtual devices ..... devbuild(1M)
virtual devices devbuild, devbuild(1M)
devdestroy: build and destroy ..... devbuild(1M)
ioctl: control device ..... ioctl(2)
prtvtoc: print the VTOC of a block device ..... prtvtoc(1M)
print the VTOC of a standalone disk device ..... prtvtoc(8)
sp: STREAMS pipe device ..... sp(7)
or the terminal associated with a device ..... ttydevname(1)
the terminal/ ttydevname: get the device associated with a terminal or ..... ttydevname(1)
diskinfo: disk device characteristic database ..... diskinfo(4)
devnm: device name ..... devnm(1M)
clone: open any minor device on a STREAMS driver ..... clone(7)
devtab: virtual device table ..... devtab(4)
dump kernel auto-configuration device table ..... dumpconf(1M)
backup.dev: backup devices ..... backup_dev(4)
build and destroy virtual devices ..... devbuild(1M)
of virtual-to-physical mapping devices ..... pmap(1M)
SA: devices administered by System Administration ..... sa(7)
constab: constructed devices table ..... constab(4)
devnm: device name ..... devnm(1M)
devtab: virtual device table ..... devtab(4)
convert between major/minor numbers/ dev_to_ttyname, ttyname_to_dev: ..... dev_to_ttyname(3C)
blocks and inodes df: report number of free disk ..... df(1M)
dmesg: collect system diagnostic messages to form error log ..... dmesg(1M)
SCED: SCSI/Ethernet/ Diagnostics board ..... sced(7)
terminal line connection dial: establish an out-going ..... dial(3C)
bdiff: big diff ..... bdiff(1)
diff: differential file comparison ..... diff(1)
comparison diff3: 3-way differential file ..... diff3(1)
sdiff: side-by-side difference program ..... sdiff(1)
diff: differential file comparison ..... diff(1)
diff3: 3-way differential file comparison ..... diff3(1)
dir: format of directories ..... dir(4)
dircmp: directory comparison ..... dircmp(1)
dir: format of directories ..... dir(4)
unlink: link and unlink files and directories ..... link(1M)
mkdir: make directories ..... mkdir(1)
rm, rmdir: remove files or directories ..... rm(1)
uucheck: check the uucp directories and permissions file ..... uucheck(1M)
cd: change working directory ..... cd(1)
chdir: change working directory ..... chdir(2)
chroot: change root directory ..... chroot(2)
get pathname of current working directory ..... getcwd(3C)
system level transition files directory ..... init.d(4)
ls: list contents of directory ..... ls(1)
mkdir: make a directory ..... mkdir(2)
mvdir: move a directory ..... mvdir(1M)
rmdir: remove a directory ..... rmdir(2)
uucleanup: uucp spool directory clean-up ..... uucleanup(1M)
dircmp: directory comparison ..... dircmp(1)
dcheck: filesystem directory consistency check ..... dcheck(1M)
system independent/ getdents: read directory entries and put in a file ..... getdents(2)

dirent: file system independent — directory entry ..... dirent(4)
unlink: remove — directory entry ..... unlink(2)
chroot: change root — directory for a command ..... chroot(1M)
pwd: working — directory name ..... pwd(1)
telldir, seekdir, rewinddir,/ — directory: opendir, readdir, ... directory(3X)
seekdir, rewinddir, closedir: — directory operations ..... directory(3X)
mknod: make a — directory, or a special or ordinary file ..... mknod(2)
directory entry — dirent: file system independent ..... dirent(4)
basename, — dirname: deliver portions of pathnames ..... basename(1)
— dis: code disassembler ..... see *C and Lang Tools Binder*
instructions — dis386: disassemble Series 386 ..... dis386(3SEQ)
t_unbind: — disable a transport endpoint ..... t_unbind(3N)
enable, — disable: enable/disable LP printers ..... enable(1)
enable, disable: enable/ — disable LP printers ..... enable(1)
sweepon, sweepoff: enable/ — disable patrol seeks on DCC disks ..... sweepon(1M)
acct: enable or — disable process accounting ..... acct(2)
swapping, or processor migration — disabled ..... noage(1M)
dis386: — disassemble Series 386 instructions ..... dis386(3SEQ)
dis: object code — disassembler ..... see *C and Lang Tools Binder*
t_rcvdis: retrieve information from — disconnect ..... t_rcvdis(3N)
t_snddis: send user-initiated — disconnect request ..... t_snddis(3N)
installs a boot program on a — disk ..... installboot(1M)
diskusg: generate — disk accounting data by user ID ..... diskusg(1M)
df: report number of free — disk blocks and inodes ..... df(1M)
sd: SCSI — disk controller ..... sd(7)
wd: SSM SCSI — disk controller ..... wd(7)
zd: dual-channel — disk controller ..... zd(7)
disk formatter for dual-channel — disk controller ..... zdformat(8)
print the VTOC of a standalone — disk device ..... prtvtoc(8)
diskinfo: — disk device characteristic database ..... diskinfo(4)
CCSformat: standalone embedded SCSI — disk formatter ..... ccsformat(8)
controller zdformat: standalone — disk formatter for dual-channel disk ..... zdformat(8)
getgeombyname: get — disk geometry by name ..... getgeombyname(3SEQ)
quotactl: manipulate — disk quotas ..... quotactl(2SEQ)
du: summarize — disk usage ..... du(1M)
quota: display — disk usage and limits ..... quota(1)
database — diskinfo: disk device characteristic ..... diskinfo(4)
mkvtoc: populate VTOC on — disks ..... mkvtoc(1M)
enable/disable patrol seeks on DCC — disks ..... sweepon(1M)
mgmt functions format: format — disks or perform other defect ..... format(1M)
data by user ID — diskusg: generate disk accounting ..... diskusg(1M)
and/or the working/ indicator: — display application specific alarms ..... indicator(1F)
quota: — display disk usage and limits ..... quota(1)
vi: screen-oriented (visual) — display editor based on ex ..... vi(1)
man: — display on-line man pages ..... man(1)
virtual-to-physical mapping/ pmap: — display or alter parameters of ..... pmap(1M)
prof: — display profile data ..... prof(1)
bootflags: — display/change system boot flags ..... bootflags(1M)
hypot: Euclidean — distance function ..... hypot(3M)
— dltsectbl: refresh security table ..... dltsectbl(1M)
/dbm_error, dbm_clearerr, — dmb_blkflush: database subroutines ..... ndbm(3X)
messages to form error log — dmesg: collect system diagnostic ..... dmesg(1M)

checkeq: check — documents for correct eqn syntax ..... checkeq(1)
prctmp,/ chargefee, ckpacct, — dodisk, lastlogin, monacct, nulladm, ... acctsh(1M)
strtod, atof: convert string to — double-precision number ..... strtod(3C)
mrand48, jrand48, srand48, seed48,/ — drand48, erand48, lrand48, nrand48, ... drand48(3C)
open any minor device on a STREAMS — driver ..... clone(7)
cmpt: compatibility partitioning — driver ..... cmpt(7)
cn: pseudoconsole — driver ..... cn(7)
terminal multiplexor — driver ..... st(7)
streampty: STREAMS pseudo terminal — driver ..... streampty(7)
vtoc: VTOC partitioning — driver ..... vtoc(7)
Xylogics 781 terminal multiplexor — driver ..... xs(7)
streamtty: Generic STREAMS tty — driver interface ..... streamtty(7)
consmem, sm, ss: console memory — drivers ..... consmem(7)
du: summarize disk usage ..... du(1M)
zd: — dual-channel disk controller ..... zd(7)
/standalone disk formatter for — dual-channel disk controller ..... zdformat(8)
dump: incremental filesystem — dump ..... dump(1M)
dump: standalone memory — dump ..... dump(8)
od: octal — dump ..... od(1)
of an object file — dump: dump parts ..... see *C and Lang Tools Binder*
format — dump, dumpdates: incremental dump ..... dump(4)
dumpfs: — dump filesystem information ..... dumpfs(1M)
dump, dumpdates: incremental — dump format ..... dump(4)
dump: incremental filesystem dump ..... dump(1M)
device table dumpconf: — dump kernel auto-configuration ..... dumpconf(1M)
savecore: save a core — dump of the operating system ..... savecore(1M)
dump: standalone memory dump ..... dump(8)
auto-configuration device table — dumpconf: dump kernel ..... dumpconf(1M)
dump, — dumpdates: incremental dump format ..... dump(4)
dumpfs: dump filesystem information ..... dumpfs(1M)
descriptor — dup: duplicate an open file ..... dup(2)
descriptor — dup2: duplicate an open file ..... dup2(2)
dup: — duplicate an open file descriptor ..... dup(2)
dup2: — duplicate an open file descriptor ..... dup2(2)
ec: forms compiler ..... ec(1M)
echo: — echo arguments ..... echo(1)
echo: echo arguments ..... echo(1)
echo: put string on virtual output ..... echo(1F)
floating-point number to string — ecvt, fcvt, gcvt: convert ..... ecvt(3C)
ed, red: text editor ..... ed(1)
end, etext, — edata: last locations in program ..... end(3C)
command — edc: run an externally described ..... edc(1M)
casual users) — edit: text editor (variant of ex for ..... edit(1)
edquota: — edit user quotas ..... edquota(1M)
sact: print current SCCS file — editing activity ..... sact(1)
ed, red: text — editor ..... ed(1)
ex: text — editor ..... ex(1)
sed: stream — editor ..... sed(1)
vi: screen-oriented (visual) display — editor based on ex ..... vi(1)
ld: link — editor for object files ..... see *C and Lang Tools Binder*
a.out: common assembler and link — editor output ..... a.out(4)
edit: text — editor (variant of ex for casual users) ..... edit(1)

edquota: edit user quotas ..... edquota(1M)
effective user, real group, and effective group IDs ..... getuid(2)
/getgid, getegid: get real user, effective user, real group, and/ ..... getuid(2)
efl: Extended Fortran Language ..... efl(1)
fsplit: split f77, ratfor, or efl files ..... fsplit(1)
using full regular expressions egrep: search a file for a pattern ..... egrep(1)
CCSformat: standalone embedded SCSI disk formatter ..... ccsformat(8)
printers enable, disable: enable/disable LP ..... enable(1)
acct: enable or disable process accounting ..... acct(2)
enable, disable: enable/disable LP printers ..... enable(1)
sweepon, sweepoff: enable/disable patrol seeks on DCC disks ..... sweepon(1M)
crypt: encode/decode ..... crypt(1)
crypt, setkey, encrypt: generate hashing encryption ..... crypt(3C)
setkey, encrypt: generate hashing encryption ..... crypt(3C)
crypt: password and file encryption functions ..... crypt(3X)
makekey: generate encryption key ..... makekey(1)
program end, etext, edata: last locations in ..... end(3C)
gencc: create a front- end to the cc command ..... gencc(1M)
/getgrgid, getgrnam, setgrent, endgrent, fgetgrent: get group file entry ..... getgrent(3C)
setmntent: get file/ addmntent, endmntent, getmntent, hasmntopt, ... getmntent(3SEQ)
bind an address to a transport endpoint ..... t_bind(3N)
t_close: close a transport endpoint ..... t_close(3N)
at the current event on a transport endpoint ..... t_look(3N)
t_open: establish a transport endpoint ..... t_open(3N)
manage options for a transport endpoint ..... t_optmgmt(3N)
t_unbind: disable a transport endpoint ..... t_unbind(3N)
/getpwuid, getpwnam, setpwent, endpwent, fgetpwent: get password file/ ..... getpwent(3C)
getspent, getspnam, setspent, endspent, fgetspent, lckpwdf,/ ..... getspent(3X)
/getutline, pututline, setutent, endutent, utmpname, getutpid: access/ ..... getut(3C)
mailbug: submit a bug report or enhancement request ..... mailbug(1)
execution env: set environment for command ..... env(1)
files .environ, .pref, .variables: FACE ..... env(4)
environ: user environment ..... environ(5)
environ: user environment ..... environ(5)
putenv: change or add value to environment ..... putenv(3C)
commands performed for multi-user environment ..... rc2(1M)
profile: setting up an environment at login time ..... profile(4)
fpsetsticky: IEEE floating-point environment control ..... fpgetround(3C)
env: set environment for command execution ..... env(1)
getenv: return value for environment name ..... getenv(3C)
set, unset: set and unset environment variables in core or in files ..... set(1F)
deroff: remove nroff, tbl, and eqn constructs ..... deroff(1)
checkeq: check documents for correct eqn syntax ..... checkeq(1)
jrand48, srand48, seed48,/ drand48, erand48, lrand48, nrand48, mrand48, ... drand48(3C)
complementary error function erf, erfc: error function and ..... erf(3M)
complementary error function erf, erfc: error function and ..... erf(3M)
error messages perror, errno, sys_errlist, sys_nerr: system ..... perror(3C)
error function and complementary error function ..... erf(3M)
error function erf, erfc: error function and complementary ..... erf(3M)
t_rcvuderr: receive a unit data error indication ..... t_rcvuderr(3N)
system diagnostic messages to form error log ..... dmesg(1M)
errorlog: internal error log file ..... errorlog(4)

strclean: STREAMS error logger cleanup program ..... strclean(1M)
strerr: STREAMS error logger daemon ..... strerr(1M)
log: interface to STREAMS error logging and event tracing ..... log(7)
t_error: produce error message ..... t_error(3N)
errormsg: error message text file ..... errormsg(4)
errno, sys_errlist, sys_nerr: system error messages ..... perror(3C)
introduction to system calls and error numbers ..... intro(2)
matherr: error-handling function ..... matherr(3M)
errorlog: internal error log file ..... errorlog(4)
errormsg: error message text file ..... errormsg(4)
spellin, hashcheck: find spelling errors ..... spell(1)
/etc/shadow with information from / etc/passwd ..... pwconv(1M)
pwconv: installs and updates / etc/shadow with information from/ ..... pwconv(1M)
end, etext, edata: last locations in program ..... end(3C)
SCED: SCSI/ Ethernet/Diagnostics board ..... sced(7)
hypot: Euclidean distance function ..... hypot(3M)
expr: evaluate arguments as an expression ..... expr(1)
test: condition evaluation command ..... test(1)
t_look: look at the current event on a transport endpoint ..... t_look(3N)
to STREAMS error logging and event tracing ..... log(7)
sulog: log of substitute user events ..... sulog(4)
(visual) display editor based on ex ..... vi(1)
edit: text editor (variant of ex for casual users) ..... edit(1)
ex: text editor ..... ex(1)
execlp, execvp: execute a file exec: execl, execv, execle, execve, ... exec(2)
execlp, execvp: execute a/ exec: execl, execv, execle, execve, ... exec(2)
exec: execl, execv, execle, execve, execlp, execvp: execute a file ..... exec(2)
exec: execl, execv, execle, execve, execlp, execvp: execute a file ..... exec(2)
run: run an executable ..... run(1F)
execle, execve, execlp, execvp: execute a file ..... exec(2)
m_fork: execute a subprogram in parallel ..... m_fork(3PPS)
construct argument list(s) and execute command ..... xargs(1)
at, batch: execute commands at a later time ..... at(1)
regcmp, regex: compile and execute regular expression ..... regcmp(3X)
uuxqt: execute remote command requests ..... uuxqt(1M)
env: set environment for command execution ..... env(1)
suspend and resume child process execution ..... m_park_procs(3PPS)
/system command execution ..... uux(1C)
sleep: suspend execution for an interval ..... sleep(1)
sleep: suspend execution for interval ..... sleep(3C)
monitor: prepare execution profile ..... monitor(3C)
profil: execution time profile ..... profil(2)
execvp: execute a file exec: execl, execv, execle, execve, execlp, ... exec(2)
exec: execl, execv, execle, execve, execlp, execvp: execute a file ..... exec(2)
execv, execle, execve, execlp, execvp: execute a file ..... exec(2)
exit, _exit: terminate process ..... exit(2)
_exit: terminate process ..... exit(2)
exponential, logarithm, power/ exp, log, log10, pow, sqrt: ..... exp(3M)
uncompress, zcat: compress and expand data ..... compress(1)
pack, pcat, unpack: compress and expand files ..... pack(1)
exp, log, log10, pow, sqrt: exponential, logarithm, power, square/ ..... exp(3M)
expression expr: evaluate arguments as an ..... expr(1)

expr: evaluate arguments as an expression ..... expr(1)
regex: compile and execute regular expression ..... regcmp(3X)
regcmp: regular expression compile ..... regcmp(1)
regexp: regular expression compile and match routines ..... regexp(5)
for a pattern using full regular expressions ..... egrep(1)
efl: Extended Fortran Language ..... efl(1)
edc: run an externally described command ..... edc(1M)
fsplit: split f77, ratfor, or efl files ..... fsplit(1)
floor, ceil, fmod, fabs: floor, ceiling, remainder, absolute value/ ..... floor(3M)
pathalias: alias file for FACE ..... alias(4)
.environ, .pref, .variables: FACE files ..... env(4)
report inter-process communication facilities status ..... ipcs(1)
a number factor: obtain the prime factors of ..... factor(1)
factor: obtain the prime factors of a number ..... factor(1)
true, false: provide truth values ..... true(1)
abort: generate an IOT fault ..... abort(3C)
stream fclose, fflush: close or flush a ..... fclose(3S)
fcntl: file control ..... fcntl(2)
fcntl: file control options ..... fcntl(5)
number to string ecvt, fcvt, gcvt: convert floating-point ..... ecvt(3C)
fopen, freopen, fdopen: open a stream ..... fopen(3S)
ferror, feof, clearerr, fileno: stream status inquiries ..... ferror(3S)
stream status inquiries ferror, feof, clearerr, fileno: ..... ferror(3S)
for a filesystem ff: list filenames and statistics ..... ff(1M)
fclose, fflush: close or flush a stream ..... fclose(3S)
from a stream getc, getchar, fgetc, getw: get character or word ..... getc(3S)
/getgrnam, setgrent, endgrent, fgetgrent: get group file entry ..... getgrent(3C)
/getpwnam, setpwent, endpwent, fgetpwent: get password file entry ..... getpwent(3C)
gets, fgets: get a string from a stream ..... gets(3S)
/getspnam, setspent, endspent, fgetspent, lckpwdf, ulckpwdf: get/ ..... getspent(3X)
string fgrep: search a file for a character ..... fgrep(1)
cut: cut out selected fields of each line of a file ..... cut(1)
mkfifo: make a FIFO special file ..... mkfifo(3C)
access: determine accessibility of a file ..... access(2)
chmod: change mode of file ..... chmod(2)
chown: change owner and group of a file ..... chown(2)
core: format of core image file ..... core(4)
cronlog: cron history file ..... cronlog(4)
crontab: user crontab file ..... crontab(1)
crontabs: clock daemon file ..... crontabs(4)
ctags: create a tags file ..... ctags(1)
selected fields of each line of a file ..... cut(1)
dd: convert and copy a file ..... dd(1M)
make a delta (change) to an SCCS file ..... delta(1)
errorlog: internal error log file ..... errorlog(4)
errormsg: error message text file ..... errormsg(4)
execve, execlp, execvp: execute a file ..... exec(2)
get: get a version of an SCCS file ..... get(1)
group: group file ..... group(4)
issue: issue identification file ..... issue(4)
header of a member of an archive file ..... ldahread(3X)
ldaclose: close a common object file ..... ldclose(3X)

the file header of a common object file.....ldfhread(3X)
of a section of a common object file.....ldlseek(3X)
file header of a common object file.....ldohseek(3X)
of a section of a common object file.....ldrseek(3X)
section header of a common object file.....ldshread(3X)
section of a common object file.....ldsseek(3X)
table entry of a common object file.....ldtbindex(3X)
table entry of a common object file.....ldtbread(3X)
the symbol table of a common object file.....ldtbseek(3X)
number entries in a common object file.....linenum(4)
link: link to a file.....link(2)
mkfifo: make a FIFO special file.....mkfifo(3C)
mknod: build special file.....mknod(1M)
directory, or a special or ordinary file.....mknod(2)
format of a module configuration file.....mod_config(4)
motd: message-of-the-day file.....motd(4)
newform: change the format of a text file.....newform(1)
null: the null file.....null(7)
passwd: password file.....passwd(4)
files or subsequent lines of one file.....paste(1)
product: installed products file.....product(4)
prs: print an SCCS file.....prs(1)
rcsfile: format of RCS file.....rcsfile(4)
read: read from file.....read(2)
readv: read vector from file.....readv(2SEQ)
reinit: run an initialization file.....reinit(1F)
information for a common object file.....reloc(4)
rename: change the name of a file.....rename(2)
rmdel: remove a delta from an SCCS file.....rmdel(1)
compare two versions of an SCCS file.....sccsdiff(1)
sccsfile: format of SCCS file.....sccsfile(4)
sccstorcs: build RCS file from SCCS file.....sccstorcs(1)
section header for a common object file.....scnhdr(4)
format of curses screen image file.....scr_dump(4)
shadow: restricted password file.....shadow(4)
system shutdown information file.....shutdown(4)
spellhist: unmatched spelling words file.....spellhist(4)
print checksum and block count of a file.....sum(1)
symlink: make symbolic link to a file.....symlink(2SEQ)
format of a system configuration file.....sys_config(4)
tail: deliver the last part of a file.....tail(1)
term: format of compiled term file.....term(4)
tmpfile: create a temporary file.....tmpfile(3S)
create a name for a temporary file.....tmpnam(3S)
access and modification times of a file.....touch(1)
undo a previous get of an SCCS file.....unget(1)
uniq: report repeated lines in a file.....uniq(1)
add/delete users in utmp file.....utadduser(3X)
the uucp directories and permissions file.....uucheck(1M)
val: validate SCCS file.....val(1)
write: write on a file.....write(2)
writev: write vector on a file.....writev(2SEQ)

utime: set file access and modification times.....utime(2)
ldfcn: common object file access routines.....ldfcn(4)
readfile, longline: reads file and gets longest line.....readfile(1F)
cpset: copy file and set mode, user, and group IDs.....cpset(1)
Hewlett-Packard 2645A terminal tape file archiver.....hpio(1)
tar: tape file archiver.....tar(1)
cpio: copy file archives in and out.....cpio(1)
rcs: change RCS file attributes.....rcs(1)
pwck, grpck: password/group file checkers.....pwck(1M)
mcs: manipulate the object file comment section.....mcs(1)
diff: differential file comparison.....diff(1)
diff3: 3-way differential file comparison.....diff3(1)
fcntl: file control.....fcntl(2)
fcntl: file control options.....fcntl(5)
UNIX-to-UNIX system file copy.....uuto(1C)
umask: set and get file creation mask.....umask(2)
close: close a file descriptor.....close(2)
dup: duplicate an open file descriptor.....dup(2)
dup2: duplicate an open file descriptor.....dup2(2)
file: determine file type.....file(1)
sact: print current SCCS file editing activity.....sact(1)
crypt: password and file encryption functions.....crypt(3X)
endgrent, fgetgrent: get group file entry.....getgrent(3C)
get file system descriptor file entry.....getmntent(3SEQ)
endpwent, fgetpwent: get password file entry.....getpwent(3C)
ulckpwdf: get shadow password file entry.....getspent(3X)
utmpname, getutpid: access utmp file entry.....getut(3C)
putpwent: write password file entry.....putpwent(3C)
putspent: write shadow password file entry.....putspent(3X)
fgrep: search a file for a character string.....fgrep(1)
grep: search a file for a pattern.....grep(1)
egrep: search a file for a pattern using full regular expressions.....egrep(1)
proto: prototype job file for at.....proto(4)
pathalias: alias file for FACE.....alias(4)
ldaopen: open a common object file for reading.....ldopen(3X)
acct: per-process accounting file format.....acct(4)
ar: common archive file format.....see *C and Lang Tools Binder*
manager, updates text files in ISIS file format.....isismgr(1M)
intro: introduction to file formats.....intro(4)
sccstorcs: build RCS file from SCCS file.....sccstorcs(1)
number entries of a common object file function.....ldlread(3X)
filehdr: file header for common object files.....filehdr(4)
limits: file header for implementation-specific/.....limits(4)
unistd: file header for symbolic constants.....unistd(4)
ldfhread: read the file header of a common object file.....ldfhread(3X)
ldohseek: seek to the optional file header of a common object file.....ldohseek(3X)
split: split a file into pieces.....split(1)
mmap: map an open file into the processs address space.....mmap(2SEQ)
ISIS file format isismgr: ISIS file manager, updates text files in.....isismgr(1M)
merge: three-way file merge.....merge(1)
ttyslot: find the slot in the utmp file of the current user.....ttyslot(3C)
more: view a file one full screen at a time.....more(1)

fuser: identify processes using a — file or file structure — fuser(1M)
creat: create a new — file or rewrite an existing one — creat(2)
pg: — file perusal filter for CRTs — pg(1)
lseek: move read/write — file pointer — lseek(2)
fseek, rewind, ftell: reposition a — file pointer in a stream — fseek(3S)
bfs: big — file scanner — bfs(1)
from a two to a one password — file scheme — pwunconv(1M)
stat, lstat, fstat: get — file status — stat(2)
identify processes using a file or — file structure — fuser(1M)
symbol name for common object — file symbol table entry — ldgetname(3X)
syms: common object — file symbol table format — syms(4)
newfs: construct a new — file system — newfs(1M)
/read directory entries and put in a — file system independent format — getdents(2)
/getmntent, hasmntopt, setmntent: get — file system descriptor file entry — getmntent(3SEQ)
dirent: — file system independent directory entry — dirent(4)
fs, inode: format of — file system volume — fs(4)
truncate, ftruncate: truncate a — file to a specified length — truncate(2SEQ)
uusched: the scheduler for the uucp — file transport program — uusched(1M)
uucico: — file transport program for the uucp system — uucico(1M)
ftw: walk a — file tree — ftw(3C)
file: determine — file type — file(1)
umask: set — file-creation mode mask — umask(1)
object files — filehdr: file header for common — filehdr(4)
mktemp: make a unique — filename — mktemp(3C)
search Interpreter criteria for — filename — pathconv(1F)
ctermid: generate — filename for terminal — ctermid(3S)
ff: list — filenames and statistics for a filesystem — ff(1M)
ferror, feof, clearerr, — fileno: stream status inquiries — ferror(3S)
search and print process accounting — files — acctcom(1)
merge or add total accounting — files — acctmerg(1M)
admin: create and administer SCCS — files — admin(1)
cat: concatenate and print — files — cat(1)
cmp: compare two — files — cmp(1)
or reject lines common to two sorted — files — comm(1)
config: build system configuration — files — config(1M)
cp, ln, mv: copy, link or move — files — cp(1)
.environ, .pref, .variables: FACE — files — env(4)
file header for common object — files — filehdr(4)
find: find — files — find(1)
fspec: format specification in text — files — fspec(4)
fsplit: split f77, ratfor, or efl — files — fsplit(1)
string, format of graphical — files — gps(4)
ident: identify — files — ident(1)
intro: introduction to special — files — intro(7)
lockf: record locking on — files — lockf(3C)
makedev: make system special — files — makedev(1M)
pcat, unpack: compress and expand — files — pack(1)
pr: print — files — pr(1)
and other information about RCS — files — rlog(1)
environment variables in core or in — files — set(1F)
sort: sort and/or merge — files — sort(1)
uucp: networking support — files — uucp(4)

what: identify SCCS files ..... what(1)
link, unlink: link and unlink files and directories ..... link(1M)
init.d: system level transition files directory ..... init.d(4)
ISIS file manager, updates text files in ISIS file format ..... isismgr(1M)
passmgmt: password files management ..... passmgmt(1M)
rm, rmdir: remove files or directories ..... rm(1)
paste: merge same lines of several files or subsequent lines of one/ ..... paste(1)
ott: files that hold object architecture information ..... ott(4)
checkfsys: check a filesystem ..... checkfsys(1M)
list filenames and statistics for a filesystem ..... ff(1M)
mkfs: construct a filesystem ..... mkfs(1M)
mount: mount a filesystem ..... mount(2)
repquota: summarize quotas for a filesystem ..... repquota(1M)
tunefs: tune up an existing filesystem ..... tunefs(1M)
umount: unmount a filesystem ..... umount(2)
volcopy: make literal copy of filesystem ..... volcopy(1M)
ckbupscd: check filesystem backup schedule ..... ckbupscd(1M)
interactive repair fsck: filesystem consistency check and ..... fsck(1M)
dcheck: filesystem directory consistency check ..... dcheck(1M)
dump: incremental filesystem dump ..... dump(1M)
fstyp: determine filesystem identifier ..... fstyp(1M)
dumpfs: dump filesystem information ..... dumpfs(1M)
statfs, fstatfs: get filesystem information ..... statfs(2)
quotacheck: filesystem quota consistency checker ..... quotacheck(1M)
quotaon, quotaoff: turn filesystem quotas on and off ..... quotaon(1M)
restore: incremental filesystem restore ..... restore(1M)
ustat: get filesystem statistics ..... ustat(2)
fsstat: report filesystem status ..... fsstat(1M)
icheck: filesystem storage consistency check ..... icheck(1M)
mnttab: mounted filesystem table ..... mnttab(4)
labelit: provide labels for filesystems ..... labelit(1M)
mount, umount: mount and unmount filesystems ..... mount(1M)
umountall: mount, unmount multiple filesystems ..... mountall(1M)
checklist: list of filesystems processed by fsck and ncheck ..... checklist(4)
fstab: file-system-table ..... fstab(4)
greek: select terminal filter ..... greek(1)
nl: line numbering filter ..... nl(1)
pg: file perusal filter for CRTs ..... pg(1)
col: filter reverse line-feeds ..... col(1)
lpfilter: administer filters used with the LP print service ..... lpfilter(1M)
find: find files ..... find(1)
for an object library lorder: find ordering relation ..... see *C and Lang Tools Binder*
display/change system boot flags ..... bootflags(1M)
/fpgetsticky, fpsetsticky: IEEE floating-point environment control ..... fpgetround(3C)
isnan: isnand, isnanf: test for floating-point NaN (Not-A-Number) ..... isnan(3C)
ecvt, fcvt, gcvt: convert floating-point number to string ..... ecvt(3C)
ldexp, modf: manipulate parts of floating-point numbers ..... frexp(3C)
ceiling, remainder, absolute value/ floor, ceil, fmod, fabs: floor, ... floor(3M)
floor, ceil, fmod, fabs: floor, ceiling, remainder, absolute value/ ..... floor(3M)
fclose, fflush: close or flush a stream ..... fclose(3S)
a co-process with the controlling FMLI object ..... vsig(1F)
floor, ceil, fmod, fabs: floor, ceiling, remainder, absolute/ ..... floor(3M)

stream fopen, freopen, fdopen: open a ..... fopen(3S)
tcgetpgrp: get foreground process group ID..... tcgetpgrp(2)
tcsetpgrp: set foreground process group ID..... tcsetpgrp(2)
fork: create a new process..... fork(2)
system diagnostic messages to form error log..... dmesg(1M)
reset: reset the current form field to its default values..... reset(1F)
acct: per-process accounting file format..... acct(4)
dump, dumpdates: incremental dump format..... dump(4)
put in a file system independent format..... getdents(2)
updates text files in ISIS file format..... isismgr(1M)
common object file symbol table format..... syms(4)
management functions format: format disks or perform other defect..... format(1M)
other defect management functions format: format disks or perform ..... format(1M)
mod_config: format of a module configuration file..... mod_config(4)
sys_config: format of a system configuration file..... sys_config(4)
newform: change the format of a text file..... newform(1)
term: format of compiled term file..... term(4)
core: format of core image file..... core(4)
cpio: format of cpio archive..... cpio(4)
scr_dump: format of curses screen image file..... scr_dump(4)
dir: format of directories..... dir(4)
fs, inode: format of file system volume ..... fs(4)
gps: graphical primitive string, format of graphical files..... gps(4)
rcsfile: format of RCS file..... rcsfile(4)
sccsfile: format of SCCS file ..... sccsfile(4)
fspec: format specification in text files ..... fspec(4)
strftime: format time information ..... strftime(3C)
intro: introduction to file formats..... intro(4)
utmp, wtmp: utmp and wtmp entry formats..... utmp(4)
scanf, fscanf, sscanf: convert formatted input ..... scanf(3S)
printf, fprintf, sprintf: print formatted output ..... printf(3S)
vprintf, vfprintf, vsprintf: print formatted output of a varargs/..... vprintf(3S)
pwfix: remove incorrectly formatted password entries..... pwfix(1M)
standalone embedded SCSI disk formatter..... ccsformat(8)
zdformat: standalone disk formatter for dual-channel disk/..... zdformat(8)
ASSIST menus or command forms..... astgen(1)
ec: forms compiler..... ec(1M)
lpforms: administer forms used with the LP print service..... lpforms(1M)
addformopt: add a form/SELECT window option to a menu ..... addformopt(1M)
ratfor: rational Fortran dialect..... ratfor(1)
efl: Extended Fortran Language ..... efl(1)
variables pathconf, fpathconf: get configurable pathname..... pathconf(2)
fpgetround, fpsetround, fpgetmask, fpsetmask, fpgetsticky,/ ..... fpgetround(3C)
fpsetmask, fpgetsticky,/ fpgetround, fpsetround, fpgetmask, ... fpgetround(3C)
/fpsetround, fpgetmask, fpsetmask, fpgetsticky, fpsetsticky: IEEE/..... fpgetround(3C)
printf, fprintf, sprintf: print formatted output..... printf(3S)
fpgetround, fpsetround, fpgetmask, fpsetmask, fpgetsticky, fpsetsticky:/ ..... fpgetround(3C)
fpgetsticky,/ fpgetround, fpsetround, fpgetmask, fpsetmask, ... fpgetround(3C)
/fpgetmask, fpsetmask, fpgetsticky, fpsetsticky: IEEE floating-point/ ..... fpgetround(3C)
putc, putchar, fputc, putw: put character or word on a stream ..... putc(3S)
puts, fputs: put a string on a stream..... puts(3S)
getfrm: returns the current frame number ..... getfrm(1F)

fread, fwrite: binary input/output ..... fread(3S)
t_free: free a library structure ..... t_free(3N)
df: report number of free disk blocks and inodes ..... df(1M)
malloc, free, realloc, calloc: main memory allocator ..... malloc(3C)
mallinfo: fast main memory/ malloc, free, realloc, calloc, mallopt, ... malloc(3X)
fopen, freopen, fdopen: open a stream ..... fopen(3S)
of floating-point numbers frexp, ldexp, modf: manipulate parts ..... frexp(3C)
gencc: create a front-end to the cc command ..... gencc(1M)
volume fs, inode: format of file system ..... fs(4)
scanf, fscanf, sscanf: convert formatted input ..... scanf(3S)
list of filesystems processed by fsck and ncheck ..... checklist(4)
and interactive repair fsck: filesystem consistency check ..... fsck(1M)
file pointer in a stream fseek, rewind, ftell: reposition a ..... fseek(3S)
generation numbers fsirand: install random inode ..... fsirand(1M)
files fspec: format specification in text ..... fspec(4)
files fsplit: split f77, ratfor, or efl ..... fsplit(1)
fsstat: report filesystem status ..... fsstat(1M)
fstab: file-system-table ..... fstab(4)
stat, lstat, fstat: get file status ..... stat(2)
statfs, fstatfs: get filesystem information ..... statfs(2)
identifier fstyp: determine filesystem ..... fstyp(1M)
fseek, rewind, ftell: reposition a file pointer in a stream ..... fseek(3S)
communication package stdipc: ftok: standard interprocess ..... stdipc(3C)
specified length truncate, ftruncate: truncate a file to a ..... truncate(2SEQ)
ftw: walk a file tree ..... ftw(3C)
function and complementary error function ..... erf(3M)
gamma: log gamma function ..... gamma(3M)
hypot: Euclidean distance function ..... hypot(3M)
entries of a common object file function ..... ldlread(3X)
matherr: error-handling function ..... matherr(3M)
prof: profile within a function ..... prof(5)
erf, erfc: error function and complementary error function ..... erf(3M)
j0, j1, jn, y0, y1, yn: Bessel functions ..... bessel(3M)
crypt: password and file encryption functions ..... crypt(3X)
logarithm, power, square root functions ..... exp(3M)
ceiling, remainder, absolute value functions ..... floor(3M)
or perform other defect management functions ..... format(1M)
sinh, cosh, tanh: hyperbolic functions ..... sinh(3M)
sysi86: machine specific functions ..... sysi86(2)
keys to internal menu system functions ..... syskeymap(4)
acos, atan, atan2: trigonometric functions ..... trig(3M)
math: math functions and constants ..... math(5)
intro: introduction to functions and libraries ..... intro(3)
300, 300s: handle special functions of DASI 300 and 300s terminals ..... 300(1)
hp: handle special functions of Hewlett-Packard terminals ..... hp(1)
450: handle special functions of the DASI 450 terminal ..... 450(1)
file or file structure fuser: identify processes using a ..... fuser(1M)
specified controlling terminal fvhangup: virtually hangup the ..... fvhangup(2SEQ)
fread, fwrite: binary input/output ..... fread(3S)
accounting records fwtmp, wtmpfix: manipulate connect ..... fwtmp(1M)
gamma: log gamma function ..... gamma(3M)
gamma: log gamma function ..... gamma(3M)

ecvt, fcvt, gcvt: convert floating-point number to string .....ecvt(3C)
command gencc: create a front-end to the cc.....gencc(1M)
command forms astgen: program for generating/modifying ASSIST menus or.....astgen(1)
fsirand: install random inode generation numbers.....fsirand(1M)
rand, srand: simple random-number generator .....rand(3C)
streamtty: Generic STREAMS tty driver interface.....streamtty(7)
tty: controlling terminal (gentty) interface .....tty(7)
getgeombyname: get disk geometry by name .....getgeombyname(3SEQ)
character or word from a stream getc, getchar, fgetc, getw: get.....getc(3S)
or word from a stream getc, getchar, fgetc, getw: get character .....getc(3S)
working directory getcwd: get pathname of current.....getcwd(3C)
put in a file system independent/ getdents: read directory entries and.....getdents(2)
descriptor table index getdtablemax: get max in-use.....getdtablemax(2SEQ)
size getdtablesize: get descriptor table.....getdtablesize(2SEQ)
getuid, geteuid, getgid, getegid: get real user, effective user, real/.....getuid(2)
name getenv: return value for environment.....getenv(3C)
user, effective user, real/ getuid, geteuid, getgid, getegid: get real.....getuid(2)
number getfrm: returns the current frame.....getfrm(1F)
geometry by name getgeombyname: get disk/.....getgeombyname(3SEQ)
getuid, geteuid, getgid, getegid: get real user, effective user,/.....getuid(2)
setgrent, endgrent, fgetgrent: get/ getgrent, getgrgid, getgrnam, ...getgrent(3C)
endgrent, fgetgrent: get/ getgrent, getgrgid, getgrnam, setgrent,.....getgrent(3C)
fgetgrent: get/ getgrent, getgrgid, getgrnam, setgrent, endgrent, ...getgrent(3C)
IDs getgroups: get supplementary group.....getgroups(2)
currently marked menu items getitems: returns a list of the.....getitems(1F)
getlogin: get login name.....getlogin(3C)
file system/ addmntent, endmntent, getmntent, hasmntopt, setmntent: get....getmntent(3SEQ)
stream getmsg: get next message off a .....getmsg(2)
argument vector getopt: get option letter from.....getopt(3C)
getopt: parse command options.....getopt(1)
getopts, getoptcvt: parse command options.....getopts(1)
options getopts, getoptcvt: parse command.....getopts(1)
getpagesize: get system page size.....getpagesize(2SEQ)
getpass: read a password.....getpass(3C)
process group, and parent/ getpid, getpgrp, getppid: get process, ...getpid(2)
process, process group, and parent/ getpid, getpgrp, getppid: get.....getpid(2)
and parent process/ getpid, getpgrp, getppid: get process, process group,.....getpid(2)
utilization information get_process_stats: get resource ....get_process_stats(2SEQ)
getpseudotty: get pseudo terminals.....getpseudotty(3SEQ)
getpw: get name from IUD .....getpw(3C)
setpwent, endpwent, fgetpwent: get/ getpwent, getpwuid, getpwnam,.....getpwent(3C)
fgetpwent: get/ getpwent, getpwuid, getpwnam, setpwent, endpwent,.....getpwent(3C)
endpwent, fgetpwent: get/ getpwent, getpwuid, getpwnam, setpwent,.....getpwent(3C)
stream gets, fgets: get a string from a.....gets(3S)
readfile, longline: reads file and gets longest line.....readfile(1F)
scsiinfo structure getscsiinfo, getscsimatch: get .....getscsiinfo(3SEQ)
getscsiinfo, getscsimatch: get scsiinfo structure.....getscsiinfo(3SEQ)
endspent, fgetspent, lckpwdf,/ getspent, getspnam, setspent,.....getspent(3X)
fgetspent, lckpwdf,/ getspent, getspnam, setspent, endspent,.....getspent(3X)
speed and terminal settings used by getty.....gettydefs(4)
speed, and line discipline getty: set terminal type, modes,.....getty(1M)
ct: spawn getty to a remote terminal.....ct(1C)

settings used by getty — gettydefs: speed and terminal ..... gettydefs(4)
get real user, effective user, real/ — getuid, geteuid, getgid, getegid: ..... getuid(2)
microsecond clock — getusclk, usclk_init: get/init ..... getusclk(3SEQ)
pututline, setutent, endutent,/ — getut: getutent, getutid, getutline, ... getut(3C)
pututline, setutent,/ getut: — getutent, getutid, getutline, ... getut(3C)
getut: getutent, — getutid, getutline, pututline, setutent,/ ..... getut(3C)
getut: getutent, getutid, — getutline, pututline, setutent, endutent,/ ..... getut(3C)
/setutent, endutent, utmpname, — getutpid: access utmp file entry ..... getut(3C)
getc, getchar, fgetc, — getw: get character or word from a stream ..... getc(3S)
zdinfo structure — getzdinfo, getzdinfobyname: get ..... getzdinfo(3SEQ)
getzdinfo, — getzdinfobyname: get zdinfo structure ..... getzdinfo(3SEQ)
m_next: increment — global counter ..... m_next(3PPS)
task: — global security table ..... task(4)
operating system terms and symbols — glossary: definitions of common ..... glossary(1)
mktime, tzset:/ ctime, localtime, — gmtime, asctime, cftime, ascftime, ... ctime(3C)
setjmp, longjmp: non-local — goto ..... setjmp(3C)
format of graphical files — gps: graphical primitive string, ... gps(4)
primitive string, format of — graphical files ..... gps(4)
of graphical files gps: — graphical primitive string, format ..... gps(4)
plot: — graphics interface ..... plot(4)
plot: — graphics interface subroutines ..... plot(3X)
— greek: select terminal filter ..... greek(1)
— grep: search a file for a pattern ..... grep(1)
chown, chgrp: change owner or — group ..... chown(1)
newgrp: log in to a new — group ..... newgrp(1M)
run command in its own process — group ..... setpgrp(1)
initgroups: initialize — group access list ..... initgroups(3SEQ)
setgroups: set — group access list ..... setgroups(2SEQ)
/get real user, effective user, real — group, and effective group IDs ..... getuid(2)
/getppid: get process, process — group, and parent process IDs ..... getpid(2)
group: — group file ..... group(4)
pwck, grpck: password/ — group file checkers ..... pwck(1M)
setgrent, endgrent, fgetgrent: get — group file entry ..... getgrent(3C)
— group: group file ..... group(4)
setpgrp: set process — group ID ..... setpgrp(2)
create session and set process — group ID ..... setsid(2)
tcgetpgrp: get foreground process — group ID ..... tcgetpgrp(2)
tcsetpgrp: set foreground process — group ID ..... tcsetpgrp(2)
copy file and set mode, user, and — group IDs ..... cpset(1)
getgroups: get supplementary — group IDs ..... getgroups(2)
user, real group, and effective — group IDs ..... getuid(2)
setuid, setgid: set user and — group IDs ..... setuid(2)
id: print user and — group IDs and names ..... id(1M)
groups: show — group memberships ..... groups(1)
chown: change owner and — group of a file ..... chown(2)
send a signal to a process or a — group of processes ..... kill(2)
maintain, update, and regenerate — groups of programs ..... make(1)
— groups: show group memberships ..... groups(1)
pwck, — grpck: password/group file checkers ..... pwck(1M)
ssignal, — gsignal: software signals ..... ssignal(3C)
stty, — gtty: set and get terminal state (defunct) ..... stty(2)
terminal fvhangup: virtually — hangup the specified controlling ..... fvhangup(2SEQ)

nohup: run a command immune to hangups and quits.....nohup(1)
hsearch, hcreate, hdestroy: manage hash search tables.....hsearch(3C)
spell, hashmake, spellin, hashcheck: find spelling errors.....spell(1)
crypt, setkey, encrypt: generate hashing encryption.....crypt(3C)
spelling errors spell, hashmake, spellin, hashcheck: find.....spell(1)
addmntent, endmntent, getmntent, hasmntopt, setmntent: get file/.....getmntent(3SEQ)
search tables hsearch, hcreate, hdestroy: manage hash.....hsearch(3C)
hsearch, hcreate, hdestroy: manage hash search tables.....hsearch(3C)
scnhdr: section header for a common object file.....scnhdr(4)
filehdr: file header for common object files.....filehdr(4)
limits: file header for implementation-specific constants.....limits(4)
unistd: file header for symbolic constants.....unistd(4)
ldfhread: read the file header of a common object file.....ldfhread(3X)
ldohseek: seek to the optional file header of a common object file.....ldohseek(3X)
/read an indexednamed section header of a common object file.....ldshread(3X)
ldahread: read the archive header of a member of an archive file.....ldahread(3X)
helplog: monitor help actions.....helplog(4)
help: operating system Help Facility.....help(1)
helpadm: change the Help Facility database.....helpadm(1M)
help: operating system Help Facility.....help(1)
database helpadm: change the Help Facility.....helpadm(1M)
helplog: monitor help actions.....helplog(4)
file archiver hpio: Hewlett-Packard 2645A terminal tape.....hpio(1)
hp: handle special functions of Hewlett-Packard terminals.....hp(1)
cronlog: cron history file.....cronlog(4)
ott: files that hold object architecture information.....ott(4)
uptime: show how long system has been up.....uptime(1)
Hewlett-Packard terminals hp: handle special functions of.....hp(1)
tape file archiver hpio: Hewlett-Packard 2645A terminal.....hpio(1)
hash search tables hsearch, hcreate, hdestroy: manage.....hsearch(3C)
sinh, cosh, tanh: hyperbolic functions.....sinh(3M)
hypot: Euclidean distance function.....hypot(3M)
u3b5, vax: get processor/ machid: i386, ns32000, pdp11, u3b, u3b2,.....machid(1)
consistency check icheck: filesystem storage.....icheck(1M)
ident: identify files.....ident(1)
m_get_myid: return process identification.....m_get_myid(3PPS)
issue: issue identification file.....issue(4)
fstyp: determine filesystem identifier.....fstyp(1M)
shmget: get shared memory segment identifier.....shmget(2)
file and set mode, user, and group IDs.....cpset(1)
getgroups: get supplementary group IDs.....getgroups(2)
process group, and parent process IDs.....getpid(2)
real group, and effective group IDs.....getuid(2)
setuid, setgid: set user and group IDs.....setuid(2)
id: print user and group IDs and names.....id(1M)
/fpsetmask, fpgetsticky, fpsetsticky: IEEE floating-point environment/.....fpgetround(3C)
core: format of core image file.....core(4)
scr_dump: format of curses screen image file.....scr_dump(4)
crash: examine system images.....crash(1M)
limits: file header for implementation-specific constants.....limits(4)
pwfix: remove incorrectly formatted password entries.....pwfix(1M)
m_next: increment global counter.....m_next(3PPS)

dump, dumpdates: incremental dump format.....dump(4)
dump: incremental filesystem dump.....dump(1M)
restore: incremental filesystem restore.....restore(1M)
get maximum in-use descriptor table index.....getdtablemax(2SEQ)
common/ ldtbindex: compute the index of a symbol table entry of a.....ldtbindex(3X)
common object/ ldtbread: read an indexed symbol table entry of a.....ldtbread(3X)
common/ldshread, ldnshread: read an indexednamed section header of a.....ldshread(3X)
ldsseek, ldnsseek: seek to an indexednamed section of a common/.....ldsseek(3X)
receipt of an orderly release indication.....t_rcvrel(3N)
receive a unit data error indication.....t_rcvuderr(3N)
specific alarms and/or the working indicator.....indicator(1F)
specific alarms and/or the working/ indicator: display application.....indicator(1F)
calculate number of blocks and indirect blocks.....nblocks(3SEQ)
syscall: indirect system call.....syscall(2)
terminfo descriptions infocmp: compare or print out.....infocmp(1M)
getusclk, usclk_init: get/ init microsecond clock.....getusclk(3SEQ)
inittab: script for the init process.....inittab(4)
initialization init, telinit: process control.....init(1M)
files directory init.d: system level transition.....init.d(4)
list initgroups: initialize group access.....initgroups(3SEQ)
init, telinit: process control initialization.....init(1M)
reinit: run an initialization file.....reinit(1F)
bcheckrc: system initialization procedures.....bcheckrc(1M)
tput: initialize a terminal or query terminfo database.....tput(1)
s_init_barrier, s_wait_barrier: initialize barrier, wait at barrier.....s_wait_barrier(3PPS)
initgroups: initialize group access list.....initgroups(3SEQ)
/s_lock, s_clock, s_unlock: initialize, lock, unlock locks.....s_lock(3PPS)
setup: initialize system for first user.....setup(1)
inittab: script for the init process.....inittab(4)
clri: clear inode.....clri(1M)
fs, inode: format of file system volume.....fs(4)
fsirand: install random inode generation numbers.....fsirand(1M)
number of free disk blocks and inodes.....df(1M)
fscanf, sscanf: convert formatted input.....scanf(3S)
ungetc: push character back into input stream.....ungetc(3S)
fread, fwrite: binary input/output.....fread(3S)
poll: STREAMS input/output multiplexing.....poll(2)
stdio: standard buffered input/output package.....stdio(3S)
uustat: uucp status inquiry and job control.....uustat(1C)
install: install commands.....install(1M)
install: install commands.....install(1M)
fsirand: install random inode generation numbers.....fsirand(1M)
on the BOM bomverify: perform an installation consistency check based.....bomverify(1M)
on a disk installboot: installs a boot program.....installboot(1M)
product: installed products file.....product(4)
installboot: installs a boot program on a disk.....installboot(1M)
with information from/ pwconv: installs and updates /etc/shadow.....pwconv(1M)
dis386: disassemble Series 386 instructions.....dis386(3SEQ)
mailx: interactive message processing system.....mailx(1)
filesystem consistency check and interactive repair.....fsck(1M)
console: console interface.....console(7)
lp: Systech parallel line printer interface.....lp(7)

mtio: UNIX magnetic tape interface ..... mtio(7)
plot: graphics interface ..... plot(4)
Generic STREAMS tty driver interface ..... streamtty(7)
swap: swap administrative interface ..... swap(1M)
termio: general terminal interface ..... termio(7)
tg: SCSI 1/2-inch magnetic tape interface ..... tg(7)
tty: controlling terminal (gentty) interface ..... tty(7)
wp: SSM parallel line printer interface ..... wp(7)
Xylogics 472 MULITBUS magnetic tape interface ..... xt(7)
timod: Transport Interface cooperating STREAMS module ..... timod(7)
STREAMS module tirdwr: Transport Interface read/write ..... tirdwr(7)
Transport Interface read/write interface STREAMS module ..... tirdwr(7)
plot: graphics interface subroutines ..... plot(3X)
sysadm: menu interface to do system administration ..... sysadm(1)
and event tracing log: interface to STREAMS error logging ..... log(7)
errorlog: internal error log file ..... errorlog(4)
syskeymap: identify keys to internal menu system functions ..... syskeymap(4)
asa: interpret ASA carriage control characters ..... asa(1)
pathconv: search Interpreter criteria for filename ..... pathconv(1F)
csh: a shell (command interpreter) with C-like syntax ..... csh(1)
pipe: create an interprocess channel ..... pipe(2)
ipcs: report inter-process communication facilities status ..... ipcs(1)
stdipc: ftok: standard interprocess communication package ..... stdipc(3C)
tfipc: interprocess communication reporter ..... tfipc(1M)
ncheck: generate pathnames from i-numbers ..... ncheck(1M)
streamio: STREAMS ioctl commands ..... streamio(7)
ioctl: control device ..... ioctl(2)
abort: generate an IOT fault ..... abort(3C)
semaphore set, or shared memory ID ipcrm: remove a message queue, ... ipcrm(1)
communication facilities status ipcs: report inter-process ..... ipcs(1)
/isxdigit, islower, isupper, isalpha, isalnum, isspace, iscntrl, ispunct,/ ..... ctype(3C)
isdigit, isxdigit, islower, isupper, isalpha, isalnum, isspace, iscntrl,/ ..... ctype(3C)
/iscntrl, ispunct, isprint, isgraph, isascii, tolower, toupper, toascci,/ ..... ctype(3C)
ttyname, isatty: find name of a terminal ..... ttyname(3C)
/isupper, isalpha, isalnum, isspace, iscntrl, ispunct, isprint, isgraph,/ ..... ctype(3C)
isalpha, isalnum, isspace, iscntrl,/ isdigit, isxdigit, islower, isupper, ... ctype(3C)
/isspace, iscntrl, ispunct, isprint, isgraph, isascii, tolower, toupper,/ ..... ctype(3C)
file manager, updates text files in ISIS file format ..... isismgr(1M)
files in ISIS file format isismgr: ISIS file manager, updates text ..... isismgr(1M)
text files in ISIS file format isismgr: ISIS file manager, updates ..... isismgr(1M)
isdigit, isxdigit, islower, isupper, isalpha, isalnum, isspace,/ ..... ctype(3C)
floating-point NaN (Not-A-Number) isnan: isnan, isnanf: test for ..... isnan(3C)
isnan: isnand, isnanf: test for floating-point NaN/ ..... isnan(3C)
(Not-A-Number) isnan: isnand, isnanf: test for floating-point NaN ..... isnan(3C)
/isalnum, isspace, iscntrl, ispunct, isprint, isgraph, isascii, tolower,/ ..... ctype(3C)
/isalpha, isalnum, isspace, iscntrl, ispunct, isprint, isgraph, isascii,/ ..... ctype(3C)
/islower, isupper, isalpha, isalnum, isspace, iscntrl, ispunct, isprint,/ ..... ctype(3C)
issue identification file ..... issue(4)
isdigit, isxdigit, islower, isupper, isalpha, isalnum, isspace,/ ..... ctype(3C)
isalnum, isspace, iscntrl,/ isdigit, isxdigit, islower, isupper, isalpha, ... ctype(3C)
getpw: get name from IUD ..... getpw(3C)
bessel: j0, j1, jn, y0, y1, yn: Bessel functions ..... bessel(3M)

bessel: j0, j1, jn, y0, y1, yn: Bessel functions ..... bessel(3M)
bessel: j0, j1, jn, y0, y1, yn: Bessel functions ..... bessel(3M)
uustat: uucp status inquiry and job control ..... uustat(1C)
proto: prototype job file for at ..... proto(4)
join: relational database operator ..... join(1)
/erand48, lrand48, nrand48, mrand48, jrand48, srand48, seed48, lcong48:/ ..... drand48(3C)
sigsetjmp, siglongjmp: non-local jumps ..... sigsetjmp(3C)
dumpconf: dump kernel auto-configuration device table ..... dumpconf(1M)
makekey: generate encryption key ..... makekey(1)
syskeymap: identify keys to internal menu system functions ..... syskeymap(4)
apropos: locate commands by keyword lookup ..... apropos(1)
system command locate: use keywords to identify an operating ..... locate(1)
killall: kill all active processes ..... killall(1M)
m_kill_procs: kill child processes ..... m_kill_procs(3PPS)
a group of processes kill: send a signal to a process or ..... kill(2)
kill: terminate a process ..... kill(1)
killall: kill all active processes ..... killall(1M)
kMBmem, kMWmem: MULITBUS memory ..... kmmem(7)
mem, kmem: core memory ..... mem(7)
kMBmem, kMWmem: MULITBUS memory ..... kmmem(7)
ksh, rksh: KornShell, a standard/restricted command and /..... ksh(1)
standard/restricted command and/ ksh, rksh: KornShell, a ..... ksh(1)
integers and long integers l3tol, ltol3: convert between 3-byte ..... l3tol(3C)
and base-64 ASCII string a64l, l64a: convert between long integer ..... a64l(3C)
filesystems labelit: provide labelit(1M)
labelit: provide labels for filesystems ..... labelit(1M)
awk: pattern scanning and processing language ..... awk(1)
bc: arbitrary-precision arithmetic language ..... bc(1)
efl: Extended Fortran Language ..... efl(1)
command and programming language ..... ksh(1)
pattern scanning and processing language ..... nawk(1)
command programming language ..... sh(1)
cftime: language specific strings ..... cftime(4)
chargefee, ckpacct, dodisk, lastlogin, monacct, nulladm, prctmp,/ ..... acctsh(1M)
/setspent, endspent, fgetspent, lckpwdf, ulckpwdf: get shadow/ ..... getspent(3X)
/mrand48, jrand48, srand48, seed48, lcong48: generate uniformly/ ..... drand48(3C)
files ld: link editor for object ..... see *C and Lang Tools Binder*
ld0: STREAMS tty line discipline ..... ld0(7)
ldclose, ldaclose: close a common object file ..... ldclose(3X)
a member of an archive file ldahread: read the archive header of ..... ldahread(3X)
for reading ldopen, ldaopen: open a common object file ..... ldopen(3X)
object file ldclose, ldaclose: close a common ..... ldclose(3X)
floating-point numbers frexp, ldexp, modf: manipulate parts of ..... frexp(3C)
routines ldfcn: common object file access ..... ldfcn(4)
common object file ldfhread: read the file header of a ..... ldfhread(3X)
common object file symbol table/ ldgetname: retrieve symbol name for ..... ldgetname(3X)
number entries of a common/ldlread, ldlinit, manipulate line number entries of a/ ..... ldlread(3X)
entries of a/ ldlread, ldlitem: manipulate line number ..... ldlread(3X)
manipulate line number entries of a/ ldlread, ldlinit, ldlitem: ..... ldlread(3X)
number entries of a section of a/ ldlseek, ldnlseek: seek to line ..... ldlseek(3X)
entries of a section of a/ ldlseek, ldnlseek: seek to line number ..... ldlseek(3X)
of a section of a common/ ldrseek, ldnrseek: seek to relocation entries ..... ldrseek(3X)

section header of a/ ldshread, ldnshread: read an indexednamed.....ldshread(3X)
section of a common object/ ldsseek, ldnsseek: seek to an indexednamed.....ldsseek(3X)
header of a common object file ldohseek: seek to the optional file.....ldohseek(3X)
object file for reading ldopen, ldaopen: open a common.....ldopen(3X)
relocation entries of a section of/ ldrseek, ldnrseek: seek to.....ldrseek(3X)
indexednamed section header of a/ ldshread, ldnshread: read an.....ldshread(3X)
indexednamed section of a common/ ldsseek, ldnsseek: seek to an.....ldsseek(3X)
symbol table entry of a common/ ldtbindex: compute the index of a.....ldtbindex(3X)
table entry of a common object file ldtbread: read an indexed symbol.....ldtbread(3X)
of a common object file ldtbseek: seek to the symbol table.....ldtbseek(3X)
truncate a file to a specified length.....truncate(2SEQ)
init.d: system level transition files directory.....init.d(4)
lexical tasks lex: generate programs for simple.....lex(1)
lex: generate programs for simple lexical tasks.....lex(1)
lsearch, lfind: linear search and update.....lsearch(3C)
intro: introduction to functions and libraries.....intro(3)
find ordering relation for an object library.....see *C and Lang Tools Binder*
introduction to Parallel Programming Library.....intro(3PPS)
t_sync: synchronize transport library.....t_sync(3N)
t_alloc: allocate a library structure.....t_alloc(3N)
t_free: free a library structure.....t_free(3N)
shell (command interpreter) with C- like syntax.....csh(1)
quota: display disk usage and limits.....quota(1)
ulimit: get and set user limits.....ulimit(2)
implementation-specific constants limits: file header for.....limits(4)
returns the number of CPUs on- line.....cpus_online(3PPS)
line: read one line.....line(1)
puts its arguments on message line.....message(1F)
all but one active processor off line.....offline_all(3SEQ)
reads file and gets longest line.....readfile(1F)
establish an out-going terminal line connection.....dial(3C)
set terminal type, modes, speed, and line discipline.....getty(1M)
ld0: STREAMS tty line discipline.....ld0(7)
set terminal type, modes, speed, and line discipline.....uugetty(1M)
man: display on- line man pages.....man(1)
linenum: line number entries in a common object file.....linenum(4)
/ldlinit, ldlitem: manipulate line number entries of a common object/.....ldlread(3X)
a common/ ldlseek, ldnlseek: seek to line number entries of a section of.....ldlseek(3X)
nl: line numbering filter.....nl(1)
cut: cut out selected fields of each line of a file.....cut(1)
lp: Systech parallel line printer interface.....lp(7)
wp: SSM parallel line printer interface.....wp(7)
line: read one line.....line(1)
lsearch, lfind: linear search and update.....lsearch(3C)
col: filter reverse line-feeds.....col(1)
common object file linenum: line number entries in a.....linenum(4)
readlink: read value of a symbolic link.....readlink(2SEQ)
link, unlink: link and unlink files and directories.....link(1M)
ld: link editor for object files.....see *C and Lang Tools Binder*
a.out: common assembler and link editor output.....a.out(4)
link: link to a file.....link(2)
cp, ln, mv: copy, link or move files.....cp(1)

link: link to a file ..... link(2)
symlink: make symbolic link to a file ..... symlink(2SEQ)
and directories link, unlink: link and unlink files ..... link(1M)
lint: a C program checker ..... see *C and Lang Tools Binder*
initgroups: initialize group access list ..... initgroups(3SEQ)
nlist: get entries from name list ..... nlist(3C)
setgroups: set group access list ..... setgroups(2SEQ)
varargs: handle variable argument list ..... varargs(5)
output of a varargs argument list ..... vprintf(3S)
ls: list contents of directory ..... ls(1)
ff: list filenames and statistics for a filesystem ..... ff(1M)
fsck and ncheck checklist: list of filesystems processed by ..... checklist(4)
getitems: returns a list of the currently marked menu items ..... getitems(1F)
t_listen: listen for a connect request ..... t_listen(3N)
listing list: produce C source ..... see *C and Lang Tools Binder*
xargs: construct argument list(s) and execute command ..... xargs(1)
volcopy: make literal copy of filesystem ..... volcopy(1M)
cp, ln, mv: copy, link or move files ..... cp(1)
ascftime, mktime, tzset:/ ctime, localtime, gmtime, asctime, cftime, ... ctime(3C)
apropos: locate commands by keyword lookup ..... apropos(1)
operating system command locate: use keywords to identify an ..... locate(1)
setlocale: set programs location ..... setlocale(3C)
end, etext, edata: last locations in program ..... end(3C)
plock: lock process, text, or data in memory ..... plock(2)
m_lock, m_unlock: lock, unlock locks ..... m_lock(3PPS)
s_clock, s_unlock: initialize, lock, unlock locks ..... s_lock(3PPS)
lockf: record locking on files ..... lockf(3C)
lockf: record locking on files ..... lockf(3C)
m_lock, m_unlock: lock, unlock locks ..... m_lock(3PPS)
s_unlock: initialize, lock, unlock locks ..... s_lock(3PPS)
diagnostic messages to form error log ..... dmesg(1M)
errorlog: internal error log file ..... errorlog(4)
gamma: log gamma function ..... gamma(3M)
newgrp: log in to a new group ..... newgrp(1M)
logging and event tracing log: interface to STREAMS error ..... log(7)
logarithm, power, square root/ exp, log, log10, pow, sqrt: exponential, ... exp(3M)
about RCS files rlog: print log messages and other information ..... rlog(1)
/usr/adm/loginlog: log of failed login attempts ..... loginlog(4)
sulog: log of substitute user events ..... sulog(4)
logarithm, power, square/ exp, log, log10, pow, sqrt: exponential, ... exp(3M)
log, log10, pow, sqrt: exponential, logarithm, power, square root/ ..... exp(3M)
strclean: STREAMS error logger cleanup program ..... strclean(1M)
strerr: STREAMS error logger daemon ..... strerr(1M)
log: interface to STREAMS error logging and event tracing ..... log(7)
/usr/adm/loginlog: log of failed login attempts ..... loginlog(4)
getlogin: get login name ..... getlogin(3C)
logname: get login name ..... logname(1)
cuserid: get character login name of the user ..... cuserid(3S)
logname: return login name of user ..... logname(3X)
passwd: change login password and password attributes ..... passwd(1)
tfmenush: assign menu system as user login shell ..... tfmenush(1M)
login: sign on ..... login(1)

setting up an environment at login time.....profile(4)
/usr/adm/ loginlog: log of failed login attempts.....loginlog(4)
logname: get login name.....logname(1)
logname: return login name of user.....logname(3X)
setjmp, longjmp: non-local goto.....setjmp(3C)
readfile, longline: reads file and gets longest line.....readfile(1F)
apropos: locate commands by keyword lookup.....apropos(1)
make a database for fast password lookup.....mkpwdbm(1M)
relation for an object library lorder: find ordering.....see *C and Lang Tools Binder*
the LP print service lp, cancel: send/cancel requests to.....lp(1)
cancel: send/cancel requests to the LP print service.....lp(1)
lpadmin: configure the LP print service.....lpadmin(1M)
administer filters used with the LP print service.....lpfilter(1M)
administer forms used with the LP print service.....lpforms(1M)
information about the status of the LP print service.....lpstat(1)
/lpshut, lpmove: start/stop the LP print service and move requests.....lpsched(1M)
enable, disable: enable/disable LP printers.....enable(1)
accept, reject: allow or prevent LP requests.....accept(1M)
interface lp: Systech parallel line printer.....lp(7)
service lpadmin: configure the LP print.....lpadmin(1M)
with the LP print service lpfilter: administer filters used.....lpfilter(1M)
the LP print service lpforms: administer forms used with.....lpforms(1M)
processing module lpmod: parallel printer canonical.....lpmod(7)
service and move/ lpsched, lpshut, lpmove: start/stop the LP print.....lpsched(1M)
the LP print service and move/ lpsched, lpshut, lpmove: start/stop.....lpsched(1M)
print service and move/ lpsched, lpshut, lpmove: start/stop the LP.....lpsched(1M)
status of the LP print service lpstat: print information about the.....lpstat(1)
supported by the menu system lptypes: table of printer types.....lptypes(4)
priorities lpusers: set printing queue.....lpusers(1M)
srand48, seed48,/ drand48, erand48, lrand48, nrand48, mrand48, jrand48,.....drand48(3C)
ls: list contents of directory.....ls(1)
update lsearch, lfind: linear search and.....lsearch(3C)
lseek: move read/write file pointer.....lseek(2)
stat, lstat, fstat: get file status.....stat(2)
integers and long integers l3tol, ltol3: convert between 3-byte.....l3tol(3C)
m4: macro processor.....m4(1)
u3b2, u3b5, vax: get processor type/ machid: i386, ns32000, pdp11, u3b,.....machid(1)
show the configuration of the machine.....showcfg(1M)
sysi86: machine specific functions.....sysi86(2)
values: machine-dependent values.....values(5)
sgetl: access long integer data in a machine-independent fashion.....sputl(3X)
m4: macro processor.....m4(1)
mtio: UNIX magnetic tape interface.....mtio(7)
tg: SCSI 1/2-inch magnetic tape interface.....tg(7)
xt: Xylogics 472 MULITBUS magnetic tape interface.....xt(7)
mt: magnetic tape manipulating program.....mt(1)
rmail: send mail to users or read mail.....mail(1)
read mail mail, rmail: send mail to users or.....mail(1)
mail, rmail: send mail to users or read mail.....mail(1)
enhancement request mailbug: submit a bug report or.....mailbug(1)
processing system mailx: interactive message.....mailx(1)
groups of programs make: maintain, update, and regenerate.....make(1)

programs intro: introduction to ..... intro(1)
/ttyname_to_dev: convert between major/minor numbers/tty names ..... dev_to_ttyname(3C)
regenerate groups of programs make: maintain, update, and ..... make(1)
man page databases makealiases, makewhatis: generate ..... makealiases(1M)
makedev: make system special files ..... makedev(1M)
makekey: generate encryption key ..... makekey(1)
databases makealiases, makewhatis: generate man page ..... makealiases(1M)
/free, realloc, calloc, mallopt, mallinfo: fast main memory allocator ..... malloc(3X)
memory allocator malloc, free, realloc, calloc: main ..... malloc(3C)
mallopt, mallinfo: fast main memory/ malloc, free, realloc, calloc, ... malloc(3X)
malloc, free, realloc, calloc, mallopt, mallinfo: fast main memory/ ..... malloc(3X)
man: display on-line man pages ..... man(1)
makealiases, makewhatis: generate man page databases ..... makealiases(1M)
man: display on-line man pages ..... man(1)
tsearch, tfind, tdelete, twalk: manage binary search trees ..... tsearch(3C)
hsearch, hcreate, hdestroy: manage hash search tables ..... hsearch(3C)
endpoint t_optmgmt: manage options for a transport ..... t_optmgmt(3N)
passmgmt: password files management ..... passmgmt(1M)
sigignore, sigpause: signal management ..... sigset(2)
format disks or perform other defect management functions ..... format(1M)
file format isismgr: ISIS file manager, updates text files in ISIS ..... isismgr(1M)
mt: magnetic tape manipulating program ..... mt(1)
parameters of virtual-to-physical mapping devices ..... pmap(1M)
m_single, m_multi: mark single-process code section ..... m_single(3PPS)
umask: set file-creation mode mask ..... umask(1)
umask: set and get file creation mask ..... umask(2)
regex: match patterns against a string ..... regex(1F)
regular expression compile and match routines ..... regexp(5)
math: math functions and constants ..... math(5)
math: math functions and constants ..... math(5)
matherr: error-handling function ..... matherr(3M)
mbad: MULITBUS adapter ..... mbad(7)
comment section mcs: manipulate the object file ..... mcs(1)
mem, kmem: core memory ..... mem(7)
read the archive header of a member of an archive file ..... ldahread(3X)
groups: show group memberships ..... groups(1)
memset: memory operations memcpy, memchr, memcmp, memcpy, ... memory(3C)
memory operations memchr, memcmp, memcpy, memset: ..... memory(3C)
operations memcmp, memcpy, memset: memory ..... memory(3C)
memory: memccpy, memchr, memcmp, memcpy, memset: memory operations ..... memory(3C)
kMBmem, kMWmem: MULITBUS memory ..... kmmem(7)
mem, kmem: core memory ..... mem(7)
lock process, text, or data in memory ..... plock(2)
create security table in shared memory ..... tasktbl(1M)
malloc, free, realloc, calloc: main memory allocator ..... malloc(3C)
calloc, mallopt, mallinfo: fast main memory allocator ..... malloc(3X)
shmalloc, shrealloc, shfree: shared memory allocator ..... shmalloc(3PPS)
valloc: aligned memory allocator ..... valloc(3SEQ)
shmctl: shared memory control operations ..... shmctl(2)
consmem, sm, ss: console memory drivers ..... consmem(7)
dump: standalone memory dump ..... dump(8)
spawn new process in a virtual memory efficient way ..... vfork(2)

queue, semaphore set, or shared memory ID ..... ipcrm(1)
memcpy, memset: memory operations memory: memccpy, memchr, memcmp, ... memory(3C)
memchr, memcmp, memcpy, memset: memory operations ..... memory(3C)
shmop, shmat, shmdt: shared memory operations ..... shmop(2)
vmtune: tune system-wide virtual- memory parameters ..... vmtune(1M)
shmget: get shared memory segment identifier ..... shmget(2)
vm_ctl: examine and change virtual memory tuning parameters ..... vm_ctl(2SEQ)
memccpy, memchr, memcmp, memcpy, memset: memory operations ..... memory(3C)
addcmdopt: add a command option to a menu ..... addcmdopt(1M)
add a form/SELECT window option to a menu ..... addformopt(1M)
addmenuopt: add a menu option to a menu ..... addmenuopt(1M)
sysadm: menu interface to do system administration ..... sysadm(1)
a list of the currently marked menu items ..... getitems(1F)
menu: menu processor ..... menu(1M)
menumsg: route user message to menu message queue ..... menumsg(1M)
rmvopt: remove a menu option ..... rmvopt(1M)
addmenuopt: add a menu option to a menu ..... addmenuopt(1M)
menu: menu processor ..... menu(1M)
of printer types supported by the menu system ..... lptypes(4)
tfmenush: assign menu system as user login shell ..... tfmenush(1M)
syskeymap: identify keys to internal menu system functions ..... syskeymap(4)
message queue menumsg: route user message to menu ..... menumsg(1M)
for generating/modifying ASSIST menus or command forms ..... astgen(1)
merge: three-way file merge ..... merge(1)
sort: sort and/or merge files ..... sort(1)
acctmerg: merge or add total accounting files ..... acctmerg(1M)
rcsmerge: merge RCS revisions ..... rcsmerge(1)
subsequent lines of one file paste: merge same lines of several files or ..... paste(1)
merge: three-way file merge ..... merge(1)
mesg: permit or deny messages ..... mesg(1)
t_error: produce error message ..... t_error(3N)
tfnotify: send completion message ..... tfnotify(1M)
msgctl: message control operations ..... msgctl(2)
message: puts its arguments on message line ..... message(1F)
getmsg: get next message off a stream ..... getmsg(2)
putmsg: send a message on a stream ..... putmsg(2)
msgop: message operations ..... msgop(2)
mailx: interactive message processing system ..... mailx(1)
message line message: puts its arguments on ..... message(1F)
menumsg: route user message to menu message queue ..... menumsg(1M)
msgget: get message queue ..... msgget(2)
shared memory ID ipcrm: remove a message queue, semaphore set, or ..... ipcrm(1)
errormsg: error message text file ..... errormsg(4)
menumsg: route user message to menu message queue ..... menumsg(1M)
motd: message-of-the-day file ..... motd(4)
mesg: permit or deny messages ..... mesg(1)
sys_errlist, sys_nerr: system error messages ..... perror(3C)
strace: print STREAMS trace messages ..... strace(1M)
rlog: print log messages and other information about RCS files ..... rlog(1)
dmesg: collect system diagnostic messages to form error log ..... dmesg(1M)
parallel m_fork: execute a subprogram in ..... m_fork(3PPS)
identification m_get_myid: return process ..... m_get_myid(3PPS)

number of child processes — m_get_numprocs: return ..... m_get_numprocs(3PPS)
getusclk, usclk_init: get/init — microsecond clock ..... getusclk(3SEQ)
usclk: 32-bit — microsecond counter ..... usclk(7)
aging, PFF, swapping, or processor — migration disabled ..... noage(1M)
clone: open any — minor device on a STREAMS driver ..... clone(7)
/convert between major/ — minor numbers and tty names ..... dev_to_ttyname(3C)
/acctwtmp: overview of accounting and — miscellaneous accounting commands ..... acct(1M)
intro: introduction to — miscellany ..... intro(5)
— mkdir: make a directory ..... mkdir(2)
— mkdir: make directories ..... mkdir(1)
— mkfifo: make a FIFO special file ..... mkfifo(3C)
— mkfs: construct a filesystem ..... mkfs(1M)
— m_kill_procs: kill child processes ..... m_kill_procs(3PPS)
— mknod: build special file ..... mknod(1M)
special or ordinary file — mknod: make a directory, or a ..... mknod(2)
password lookup — mkpwdbm: make a database for fast ..... mkpwdbm(1M)
— mktemp: make a unique filename ..... mktemp(3C)
/gmtime, asctime, cftime, ascftime, — mktime, tzset: convert date and time/ ..... ctime(3C)
— mkvtoc: populate VTOC on disks ..... mkvtoc(1M)
— m_lock, m_unlock: lock, unlock locks ..... m_lock(3PPS)
processs address space — mmap: map an open file into the ..... mmap(2SEQ)
section m_single, — m_multi: mark single-process code ..... m_single(3PPS)
— m_next: increment global counter ..... m_next(3PPS)
— mnttab: mounted filesystem table ..... mnttab(4)
configuration file — mod_config: format of a module ..... mod_config(4)
chmod: change — mode ..... chmod(1)
umask: set file-creation — mode mask ..... umask(1)
chmod: change — mode of file ..... chmod(2)
cpset: copy file and set — mode, user, and group IDs ..... cpset(1)
getty: set terminal type, — modes, speed, and line discipline ..... getty(1M)
uugetty: set terminal type, — modes, speed, and line discipline ..... uugetty(1M)
frexp, ldexp, — modf: manipulate parts of floating-point/ ..... frexp(3C)
utime: set file access and — modification times ..... utime(2)
touch: update access and — modification times of a file ..... touch(1)
astgen: program for generating/ — modifying ASSIST menus or command/ ..... astgen(1)
printer canonical processing — module ..... lpmod(7)
SSM: System Services — Module ..... ssm(7)
Interface cooperating STREAMS — module ..... timod(7)
read/write interface STREAMS — module ..... tirdwr(7)
mod_config: format of a — module configuration file ..... mod_config(4)
/ckpacct, dodisk, lastlogin, — monacct, nulladm, prctmp, prdaily,/ ..... acctsh(1M)
helplog: — monitor help actions ..... helplog(4)
— monitor: monitor system activity ..... monitor(1M)
— monitor: prepare execution profile ..... monitor(3C)
monitor: — monitor system activity ..... monitor(1M)
a time — more: view a file one full screen at ..... more(1)
— motd: message-of-the-day file ..... motd(4)
mount: — mount a filesystem ..... mount(2)
mount, umount: — mount and unmount filesystems ..... mount(1M)
— mount: mount a filesystem ..... mount(2)
setmnt: establish — mount table ..... setmnt(1M)
filesystems — mount, umount: mount and unmount ..... mount(1M)

mountall, umountall: mount, unmount multiple filesystems.....mountall(1M)
multiple filesystems mountall, umountall: mount, unmount.....mountall(1M)
mnttab: mounted filesystem table.....mnttab(4)
mvdir: move a directory.....mvdir(1M)
cp, ln, mv: copy, link or move files.....cp(1)
lseek: move read/write file pointer.....lseek(2)
start/stop the LP print service and move requests.....lpsched(1M)
and resume child process execution m_park_procs, m_rele_procs: susp/....m_park_procs(3PPS)
ptrace, mptrace: process trace facility.....ptrace(2)
drand48, erand48, lrand48, nrand48, mrand48, jrand48, srand48, seed48,/.....drand48(3C)
child process/ m_park_procs, m_rele_procs: suspend and resume....m_park_procs(3PPS)
processes m_set_procs: set number of child.....m_set_procs(3PPS)
msgctl: message control operations.....msgctl(2)
msgget: get message queue.....msgget(2)
msgop: message operations.....msgop(2)
single-process code section m_single, m_multi: mark.....m_single(3PPS)
m_sync: check in at barrier.....m_sync(3PPS)
program mt: magnetic tape manipulating.....mt(1)
st: Systech MTI-800/1600/1650 terminal multiplexor driver.....st(7)
mtio: UNIX magnetic tape interface.....mtio(7)
mbad: MULITBUS adapter.....mbad(7)
xt: Xylogics 472 MULITBUS magnetic tape interface.....xt(7)
kMBmem, kMWmem: MULITBUS memory.....kmmem(7)
mountall, umountall: mount, unmount multiple filesystems.....mountall(1M)
poll: STREAMS input/output multiplexing.....poll(2)
Systech MTI-800/1600/1650 terminal multiplexor driver.....st(7)
xs: Xylogics 781 terminal multiplexor driver.....xs(7)
tmp_ctl: multi-processor control.....tmp_ctl(2SEQ)
rc2: run commands performed for multi-user environment.....rc2(1M)
m_lock, m_unlock: lock, unlock locks.....m_lock(3PPS)
address space munmap: unmap part of the processs.....munmap(2SEQ)
cp, ln, mv: copy, link or move files.....cp(1)
mvdir: move a directory.....mvdir(1M)
devnm: device name.....devnm(1M)
getenv: return value for environment name.....getenv(3C)
getgeombyname: get disk geometry by name.....getgeombyname(3SEQ)
getlogin: get login name.....getlogin(3C)
logname: get login name.....logname(1)
pwd: working directory name.....pwd(1)
tmpnam, tempnam: create a name for a temporary file.....tmpnam(3S)
table/ ldgetname: retrieve symbol name for common object file symbol.....ldgetname(3X)
getpw: get name from IUD.....getpw(3C)
nlist: get entries from name list.....nlist(3C)
rename: change the name of a file.....rename(2)
ttyname, isatty: find name of a terminal.....ttyname(3C)
uname: get name of current operating system.....uname(2)
uname: print name of current system.....uname(1)
tty: get the name of the terminal.....tty(1)
cuserid: get character login name of the user.....cuserid(3S)
logname: return login name of user.....logname(3X)
/ldnshread: read an indexed named section header of a common object/.....ldshread(3X)
/ldnsseek: seek to an indexed named section of a common object file.....ldsseek(3X)

between major/minor numbers and tty names.....dev_to_ttyname(3C)
id: print user and group IDs and names.....id(1M)
term: conventional names for terminals.....term(5)
isnanf: test for floating-point NaN (Not-A-Number).....isnan(3C)
processing language nawk: pattern scanning and.....nawk(1)
and indirect blocks nblocks: calculate number of blocks.....nblocks(3SEQ)
of filesystems processed by fsck and ncheck.....checklist(4)
i-numbers ncheck: generate pathnames from.....ncheck(1M)
uucp: networking support files.....uucp(4)
file newform: change the format of a text.....newform(1)
newfs: construct a new file system.....newfs(1M)
newgrp: log in to a new group.....newgrp(1M)
news: print news items.....news(1)
news: print news items.....news(1)
getmsg: get next message off a stream.....getmsg(2)
nice: change priority of a process.....nice(2)
nice: run a command at low priority.....nice(1)
nl: line numbering filter.....nl(1)
nlist: get entries from name list.....nlist(3C)
object file nm: print name list of.....see *C and Lang Tools Binder*
command with priority aging, PFF/ noage, nopff, noswap, onproc: run a.....noage(1M)
hangups and quits nohup: run a command immune to.....nohup(1)
setjmp, longjmp: non-local goto.....setjmp(3C)
sigsetjmp, siglongjmp: non-local jumps.....sigsetjmp(3C)
with priority aging, PFF/ noage, nopff, noswap, onproc: run a command.....noage(1M)
priority aging, PFF/ noage, nopff, noswap, onproc: run a command with.....noage(1M)
test for floating-point NaN (Not-A-Number).....isnan(3C)
seed48,/ drand48, erand48, lrand48, nrand48, mrand48, jrand48, srand48,.....drand48(3C)
deroff: remove nroff, tbl, and eqn constructs.....deroff(1)
vax: get processor/ machid: i386, ns32000, pdp11, u3b, u3b2, u3b5,.....machid(1)
null: the null file.....null(7)
null: the null file.....null(7)
/ckpacct, dodisk, lastlogin, monacct, nulladm, prctmp, prdaily, prtacct,/.....acctsh(1M)
obtain the prime factors of a number.....factor(1)
getfrm: returns the current frame number.....getfrm(1F)
test for floating-point NaN (Not-A- Number).....isnan(3C)
convert string to double-precision number.....strtod(3C)
linenum: line number entries in a common object file.....linenum(4)
/ldlinit, ldlitem: manipulate line number entries of a common object/.....ldlread(3X)
ldlseek, ldnlseek: seek to line number entries of a section of a/.....ldlseek(3X)
rand, srand: simple random- number generator.....rand(3C)
nblocks: calculate number of blocks and indirect blocks.....nblocks(3SEQ)
m_get_numprocs: return number of child processes.....m_get_numprocs(3PPS)
m_set_procs: set number of child processes.....m_set_procs(3PPS)
cpus_online: returns the number of CPUs on-line.....cpus_online(3PPS)
df: report number of free disk blocks and inodes.....df(1M)
fcvt, gcvt: convert floating-point number to string.....ecvt(3C)
nl: line numbering filter.....nl(1)
uniformly distributed pseudorandom numbers.....drand48(3C)
manipulate parts of floating-point numbers.....frexp(3C)
install random inode generation numbers.....fsirand(1M)
to system calls and error numbers.....intro(2)

ncheck: generate pathnames from i- numbers.....ncheck(1M)
/convert between major/minor numbers and tty names.....dev_to_ttyname(3C)
cprs: compress an object file.....see *C and Lang Tools Binder*
strstr: find first occurrence of character.....strstr(3C)
od: octal dump.....od(1)
od: octal dump.....od(1)
offline: bring processors online and offline.....online(1M)
online, offline: bring processors online and offline.....online(1M)
processor off line offline_all: take all but one active.....offline_all(3SEQ)
online, offline: bring processors online and offline.....online(1M)
online, offline: bring processors.....online(1M)
aging, PFF,/ noage, nopff, noswap, onproc: run a command with priority.....noage(1M)
ldopen, ldaopen: open a common object file for reading.....ldopen(3X)
fopen, freopen, fdopen: open a stream.....fopen(3S)
clone: open any minor device on a STREAMS driver.....clone(7)
dup: duplicate an open file descriptor.....dup(2)
dup2: duplicate an open file descriptor.....dup2(2)
mmap: map an open file into the processs address space.....mmap(2SEQ)
open: open for reading or writing.....open(2)
open: open for reading or writing.....open(2)
rewinddir, closedir:/ directory: opendir, readdir, telldir, seekdir,.....directory(3X)
a standalone program or bring up the operating system.....boot(8)
run commands performed to stop the operating system.....rc0(1M)
savecore: save a core dump of the operating system.....savecore(1M)
uname: get name of current operating system.....uname(2)
locate: use keywords to identify an operating system command.....locate(1)
assist: assistance using operating system commands.....assist(1)
starter: information about the operating system for beginning users.....starter(1)
help: operating system Help Facility.....help(1)
glossary: definitions of common operating system terms and symbols.....glossary(1)
join: relational database operator.....join(1)
curses: terminal screen handling and optimization package.....curses(3X)
rmvopt: remove a menu option.....rmvopt(1M)
getopt: get option letter from argument vector.....getopt(3C)
addcmdopt: add a command option to a menu.....addcmdopt(1M)
addformopt: add a form/SELECT window option to a menu.....addformopt(1M)
addmenuopt: add a menu option to a menu.....addmenuopt(1M)
object file ldohseek: seek to the optional file header of a common.....ldohseek(3X)
fcntl: file control options.....fcntl(5)
getopt: parse command options.....getopt(1)
getopts, getoptcvt: parse command options.....getopts(1)
stty: set the options for a terminal.....stty(1)
t_optmgmt: manage options for a transport endpoint.....t_optmgmt(3N)
make a directory, or a special or ordinary file.....mknod(2)
architecture information ott: files that hold object.....ott(4)
common assembler and link editor output.....a.out(4)
echo: put string on virtual output.....echo(1F)
fread, fwrite: binary input/ output.....fread(3S)
fprintf, sprintf: print formatted output.....printf(3S)
poll: STREAMS input/ output multiplexing.....poll(2)
/vfprintf, vsprintf: print formatted output of a varargs argument list.....vprintf(3S)
stdio: standard buffered input/ output package.....stdio(3S)

setpgrp: run command in its own process group.....setpgrp(1)
chown: change owner and group of a file.....chown(2)
chown, chgrp: change owner or group.....chown(1)
expand files pack, pcat, unpack: compress and.....pack(1)
makewhatis: generate man page databases.....makealiases(1M)
getpagesize: get system page size.....getpagesize(2SEQ)
4014: paginator for the Tektronix 4014 terminal.....4014(1)
m_fork: execute a subprogram in parallel.....m_fork(3PPS)
lp: Systech parallel line printer interface.....lp(7)
wp: SSM parallel line printer interface.....wp(7)
lpmod: parallel printer canonical processing module.....lpmod(7)
intro: introduction to Parallel Programming Library.....intro(3PPS)
and change virtual memory tuning parameters.....vm_ctl(2SEQ)
tune system-wide virtual-memory parameters.....vmtune(1M)
pmap: display or alter parameters of virtual-to-physical mapping/.....pmap(1M)
get process, process group, and parent process IDs.....getpid(2)
getopt: parse command options.....getopt(1)
getopts, getoptcvt: parse command options.....getopts(1)
cmpt: compatibility partitioning driver.....cmpt(7)
vtoc: VTOC partitioning driver.....vtoc(7)
frexp, ldexp, modf: manipulate parts of floating-point numbers.....frexp(3C)
passmgmt: password files management.....passmgmt(1M)
with information from /etc/passwd.....pwconv(1M)
password attributes passwd: change login password and.....passwd(1)
passwd: password file.....passwd(4)
getpass: read a password.....getpass(3C)
crypt: password and file encryption functions.....crypt(3X)
passwd: change login password and password attributes.....passwd(1)
passwd: change login password and password attributes.....passwd(1)
pwdhd: daemon to oversee password database creation.....pwdhd(1M)
pwfix: remove incorrectly formatted password entries.....pwfix(1M)
passwd: password file.....passwd(4)
shadow: restricted password file.....shadow(4)
setpwent, endpwent, fgetpwent: get password file entry.....getpwent(3C)
lckpwdf, ulckpwdf: get shadow password file entry.....getspent(3X)
putpwent: write password file entry.....putpwent(3C)
putspent: write shadow password file entry.....putspent(3X)
converts from a two to a one password file scheme.....pwunconv(1M)
passmgmt: password files management.....passmgmt(1M)
mkpwdbm: make a database for fast password lookup.....mkpwdbm(1M)
pwck, grpck: password/group file checkers.....pwck(1M)
files or subsequent lines of one/paste: merge same lines of several.....paste(1)
bp: binary patcher.....bp(1M)
pathalias: alias file for FACE.....alias(4)
configurable pathname variables pathconf, fpathconf: get.....pathconf(2)
criteria for filename pathconv: search Interpreter.....pathconv(1F)
getcwd: get pathname of current working directory.....getcwd(3C)
fpathconf: get configurable pathname variables.....pathconf(2)
dirname: deliver portions of pathnames.....basename(1)
ncheck: generate pathnames from i-numbers.....ncheck(1M)
sweepon, sweepoff: enable/disable patrol seeks on DCC disks.....sweepon(1M)
grep: search a file for a pattern.....grep(1)

awk: pattern scanning and processing language ..... awk(1)
nawk: pattern scanning and processing language ..... nawk(1)
egrep: search a file for a pattern using full regular/ ..... egrep(1)
regex: match patterns against a string ..... regex(1F)
pause: suspend process until signal ..... pause(2)
pack, pcat, unpack: compress and expand files ..... pack(1)
popen, pclose: initiate pipe to/from a process ..... popen(3S)
pdbx, dbx: parallel/ ..... see *C and Lang Tools Binder*
machid: i386, ns32000, pdp11, u3b, u3b2, u3b5, vax: get processor/ ..... machid(1)
sigpending: examine pending signals ..... sigpending(2)
check the uucp directories and permissions file ..... uucheck(1M)
acct: per-process accounting file format ..... acct(4)
acctcms: command summary from per-process accounting records ..... acctcms(1M)
sys_nerr: system error messages perror, errno, sys_errlist, ... perror(3C)
run a command with priority aging, PFF, swapping, or processor/ ..... noage(1M)
pg: file perusal filter for CRTs ..... pg(1)
virtual address space pmap: map physical addresses into processs ..... pmap(7)
or alter parameters of virtual-to- physical mapping devices ..... pmap(1M)
pipe: create an interprocess channel ..... pipe(2)
sp: STREAMS pipe device ..... sp(7)
tee: pipe fitting ..... tee(1)
popen, pclose: initiate pipe to/from a process ..... popen(3S)
in memory plock: lock process, text, or data ..... plock(2)
plot: graphics interface ..... plot(4)
plot: graphics interface subroutines ..... plot(3X)
virtual-to-physical mapping devices pmap: display or alter parameters of ..... pmap(1M)
processs virtual address space pmap: map physical addresses into ..... pmap(7)
/fpsetsticky: IEEE floating- point environment control ..... fpgetround(3C)
isnand, isnanf: test for floating- point NaN (Not-A-Number) ..... isnan(3C)
ecvt, fcvt, gcvt: convert floating- point number to string ..... ecvt(3C)
modf: manipulate parts of floating- point numbers ..... frexp(3C)
lseek: move read/write file pointer ..... lseek(2)
rewind, ftell: reposition a file pointer in a stream ..... fseek(3S)
multiplexing a process poll: STREAMS input/output ..... poll(2)
popen, pclose: initiate pipe to/from ..... popen(3S)
banner: make posters ..... banner(1)
power, square root/ exp, log, log10, pow, sqrt: exponential, logarithm, ... exp(3M)
power, square root functions ..... exp(3M)
pr: print files ..... pr(1)
/dodisk, lastlogin, monacct, nulladm, prctmp, prdaily, prtacct, shutacct,/ ..... acctsh(1M)
/lastlogin, monacct, nulladm, prctmp, prdaily, prtacct, shutacct, startup,/ ..... acctsh(1M)
bc: arbitrary- precision arithmetic language ..... bc(1)
atof: convert string to double- precision number ..... strtod(3C)
.environ, .pref, .variables: FACE files ..... env(4)
monitor: prepare execution profile ..... monitor(3C)
cpp: the C language preprocessor ..... see *C and Lang Tools Binder*
factor: obtain the prime factors of a number ..... factor(1)
gps: graphical primitive string, format of graphical files ..... gps(4)
types: primitive system data types ..... types(5)
prs: print an SCCS file ..... prs(1)
date: print and set the date ..... date(1)
cal: print calendar ..... cal(1)

sum: print checksum and block count of a file.....sum(1)
sact: print current SCCS file editing activity.....sact(1)
cat: concatenate and print files.....cat(1)
pr: print files.....pr(1)
printf, fprintf, sprintf: print formatted output.....printf(3S)
vprintf, vfprintf, vsprintf: print formatted output of a varargs/.....vprintf(3S)
of the LP print service lpstat: print information about the status.....lpstat(1)
information about RCS files' rlog: print log messages and other.....rlog(1)
object file nm: print name list of.....see *C and Lang Tools Binder*
uname: print name of current system.....uname(1)
news: print news items.....news(1)
infocmp: compare or print out terminfo descriptions.....infocmp(1M)
acctcom: search and print process accounting files.....acctcom(1)
of object files size: print section sizes.....see *C and Lang Tools Binder*
send/cancel requests to the LP print service.....lp(1)
lpadmin: configure the LP print service.....lpadmin(1M)
administer filters used with the LP print service.....lpfilter(1M)
administer forms used with the LP print service.....lpforms(1M)
about the status of the LP print service.....lpstat(1)
/lpshut, lpmove: start/stop the LP print service and move requests.....lpsched(1M)
strace: print STREAMS trace messages.....strace(1M)
prtvtoc: print the VTOC of a block device.....prtvtoc(1M)
prtvtoc: print the VTOC of a standalone disk device.....prtvtoc(8)
id: print user and group IDs and names.....id(1M)
lpmod: parallel printer canonical processing module.....lpmod(7)
lp: Systech parallel line printer interface.....lp(7)
wp: SSM parallel line printer interface.....wp(7)
lptypes: table of printer types supported by the menu system.....lptypes(4)
enable, disable: enable/disable LP printers.....enable(1)
formatted output printf, fprintf, sprintf: print.....printf(3S)
lpusers: set printing queue priorities.....lpusers(1M)
lpusers: set printing queue priorities.....lpusers(1M)
nice: run a command at low priority.....nice(1)
/noswap, onproc: run a command with priority aging, PFF, swapping, or/.....noage(1M)
nice: change priority of a process.....nice(2)
brk, sbrk: change private data segment size.....brk(3PPS)
process attributes proc_ctl: control over various.....proc_ctl(2SEQ)
codestroy: communicate to a process.....coproc(1F)
exit, _exit: terminate process.....exit(2)
fork: create a new process.....fork(2)
inittab: script for the init process.....inittab(4)
kill: terminate a process.....kill(1)
nice: change priority of a process.....nice(2)
pclose: initiate pipe to/from a process.....popen(3S)
wait: await completion of process.....wait(1)
acct: enable or disable process accounting.....acct(2)
acctprc1, acctprc2: process accounting.....acctprc(1M)
acct: per- process accounting file format.....acct(4)
acctcom: search and print process accounting files.....acctcom(1)
acctcms: command summary from per- process accounting records.....acctcms(1M)
alarm: set a process alarm clock.....alarm(2)
times: get process and child process times.....times(2)

proc_ctl: control over various process attributes.....proc_ctl(2SEQ)
m_single, m_multi: mark single- process code section.....m_single(3PPS)
ipcs: report inter- process communication facilities status.....ipcs(1)
init, telinit: process control initialization.....init(1M)
timex: time a command; report process data and system activity.....timex(1)
suspend and resume child process execution.....m_park_procs(3PPS)
setpgrp: run command in its own process group.....setpgrp(1)
/getpgrp, getppid: get process, process group, and parent process IDs.....getpid(2)
setpgrp: set process group ID.....setpgrp(2)
setsid: create session and set process group ID.....setsid(2)
tcgetpgrp: get foreground process group ID.....tcgetpgrp(2)
tcsetpgrp: set foreground process group ID.....tcsetpgrp(2)
m_get_myid: return process identification.....m_get_myid(3PPS)
process, process group, and parent process IDs.....getpid(2)
vfork: spawn new process in a virtual memory efficient way.....vfork(2)
kill: send a signal to a process or a group of processes.....kill(2)
getpid, getpgrp, getppid: get process, process group, and parent/.....getpid(2)
ps: report process status.....ps(1)
plock: lock process, text, or data in memory.....plock(2)
times: get process and child process times.....times(2)
tmp_affinity: bind a process to a processor.....tmp_affinity(2SEQ)
wait, waitpid: wait for a child process to stop or terminate.....wait(2)
ptrace, mptrace: process trace facility.....ptrace(2)
pause: suspend process until signal.....pause(2)
vsig: synchronize a co- process with the controlling FMLI object.....vsig(1F)
checklist: list of filesystems processed by fsck and ncheck.....checklist(4)
a signal to a process or a group of processes.....kill(2)
killall: kill all active processes.....killall(1M)
return number of child processes.....m_get_numprocs(3PPS)
m_kill_procs: kill child processes.....m_kill_procs(3PPS)
m_set_procs: set number of child processes.....m_set_procs(3PPS)
fuser: identify processes using a file or file structure.....fuser(1M)
m4: macro processor.....m4(1)
menu: menu processor.....menu(1M)
tmp_affinity: bind a process to a processor.....tmp_affinity(2SEQ)
tmp_ctl: multi- processor control.....tmp_ctl(2SEQ)
priority aging, PFF, swapping, or processor migration disabled.....noage(1M)
offline_all: take all but one active processor off line.....offline_all(3SEQ)
pdp11, u3b, u3b2, u3b5, vax: get processor type truth value.....machid(1)
online, offline: bring processors online and offline.....online(1M)
mmap: map an open file into the processs address space.....mmap(2SEQ)
munmap: unmap part of the processs address space.....munmap(2SEQ)
pmap: map physical addresses into processs virtual address space.....pmap(7)
product: installed products file.....product(4)
prof: display profile data.....prof(1)
prof: profile within a function.....prof(5)
profil: execution time profile.....profil(2)
monitor: prepare execution profile.....monitor(3C)
profil: execution time profile.....profil(2)
prof: display profile data.....prof(1)
at login time profile: setting up an environment.....profile(4)
prof: profile within a function.....prof(5)

end, etext, edata: last locations in program ..... end(3C)
mt: magnetic tape manipulating program ..... mt(1)
prompt: prompt program ..... prompt(1M)
sdiff: side-by-side difference program ..... sdiff(1)
STREAMS error logger cleanup program ..... strclean(1M)
units: conversion program ..... units(1)
for the uucp file transport program ..... uusched(1M)
assert: verify program assertion ..... assert(3X)
ASSIST menus or command/ astgen: program for generating/modifying ..... astgen(1)
uucico: file transport program for the uucp system ..... uucico(1M)
installboot: installs a boot program on a disk ..... installboot(1M)
boot: run a standalone program or bring up the operating system ..... boot(8)
a standard/restricted command/ programming language ..... ksh(1)
the standard/restricted command programming language ..... sh(1)
intro: introduction to Parallel Programming Library ..... intro(3PPS)
maintenance commands and application programs ..... intro(1)
update, and regenerate groups of programs ..... make(1)
lex: generate programs for simple lexical tasks ..... lex(1)
setlocale: set programs location ..... setlocale(3C)
prompt: prompt program ..... prompt(1M)
prompt: prompt program ..... prompt(1M)
proto: prototype job file for at ..... proto(4)
t_getinfo: get protocol-specific service information ..... t_getinfo(3N)
proto: prototype job file for at ..... proto(4)
prs: print an SCCS file ..... prs(1)
/monacct, nulladm, prctmp, prdaily, prtacct, shutacct, startup/ ..... acctsh(1M)
device prtvtoc: print the VTOC of a block ..... prtvtoc(1M)
standalone disk device prtvtoc: print the VTOC of a ..... prtvtoc(8)
ps: report process status ..... ps(1)
streampty: STREAMS pseudo terminal driver ..... streampty(7)
getpseudotty: get pseudo terminals ..... getpseudotty(3SEQ)
cn: pseudoconsole driver ..... cn(7)
generate uniformly distributed pseudorandom numbers ..... drand48(3C)
time, ptime: time a command ..... time(1)
facility ptrace, mptrace: process trace ..... ptrace(2)
uuto, uupick: public UNIX-to-UNIX system file copy ..... uuto(1C)
ungetc: push character back into input stream ..... ungetc(3S)
puts, fputs: put a string on a stream ..... puts(3S)
putc, putchar, fputc, putw: put character or word on a stream ..... putc(3S)
getdents: read directory entries and put in a file system independent/ ..... getdents(2)
echo: put string on virtual output ..... echo(1F)
character or word on a stream putc, putchar, fputc, putw: put ..... putc(3S)
or word on a stream putc, putchar, fputc, putw: put character ..... putc(3S)
environment putenv: change or add value to ..... putenv(3C)
putmsg: send a message on a stream ..... putmsg(2)
putpwent: write password file entry ..... putpwent(3C)
stream puts, fputs: put a string on a ..... puts(3S)
message: puts its arguments on message line ..... message(1F)
entry putspent: write shadow password file ..... putspent(3X)
getut: getutent, getutid, getutline, pututline, setutent, endutent/ ..... getut(3C)
putc, putchar, fputc, putw: put character or word on a stream ..... putc(3S)
checkers pwck, grpck: password/group file ..... pwck(1M)

/etc/shadow with information from/ pwconv: installs and updates ..... pwconv(1M)
pwd: working directory name ..... pwd(1)
database creation pwdhd: daemon to oversee password ..... pwdhd(1M)
password entries pwfix: remove incorrectly formatted ..... pwfix(1M)
one password file scheme pwunconv: converts from a two to a ..... pwunconv(1M)
qsort: quicker sort ..... qsort(3C)
tput: initialize a terminal or query terminfo database ..... tput(1)
route user message to menu message queue ..... menumsg(1M)
msgget: get message queue ..... msgget(2)
lpusers: set printing queue priorities ..... lpusers(1M)
ipcrm: remove a message queue, semaphore set, or shared memory ID ..... ipcrm(1)
quotacheck: filesystem quota consistency checker ..... quotacheck(1M)
quota: display disk usage and limits ..... quota(1)
consistency checker quotacheck: filesystem quota ..... quotacheck(1M)
quotactl: manipulate disk quotas ..... quotactl(2SEQ)
quotaon, quotaoff: turn filesystem quotas on and off ..... quotaon(1M)
quotas on and off quotaon, quotaoff: turn filesystem ..... quotaon(1M)
edquota: edit user quotas ..... edquota(1M)
quotactl: manipulate disk quotas ..... quotactl(2SEQ)
repquota: summarize quotas for a filesystem ..... repquota(1M)
quotaon, quotaoff: turn filesystem quotas on and off ..... quotaon(1M)
generator rand, srand: simple random-number ..... rand(3C)
fsirand: install random inode generation numbers ..... fsirand(1M)
rand, srand: simple random-number generator ..... rand(3C)
fsplit: split f77, ratfor, or efl files ..... fsplit(1)
ratfor: rational Fortran dialect ..... ratfor(1)
ratfor: rational Fortran dialect ..... ratfor(1)
scripts rc: reboot and shutdown command ..... rc(4)
the operating system rc0: run commands performed to stop ..... rc0(1M)
multi-user environment rc2: run commands performed for ..... rc2(1M)
rcs: change RCS file attributes ..... rcs(1)
rcsintro: introduction to RCS commands ..... rcsintro(1)
rcsfile: format of RCS file ..... rcsfile(4)
rcs: change RCS file attributes ..... rcs(1)
sccstorcs: build RCS file from SCCS file ..... sccstorcs(1)
messages and other information about RCS files ..... rlog(1)
ci: check in RCS revisions ..... ci(1)
co: check out RCS revisions ..... co(1)
rcsdiff: compare RCS revisions ..... rcsdiff(1)
rcsmerge: merge RCS revisions ..... rcsmerge(1)
rcsdiff: compare RCS revisions ..... rcsdiff(1)
rcsfile: format of RCS file ..... rcsfile(4)
commands rcsintro: introduction to RCS ..... rcsintro(1)
rcsmerge: merge RCS revisions ..... rcsmerge(1)
getpass: read a password ..... getpass(3C)
of a common object file ldtbread: read an indexed symbol table entry ..... ldtbread(3X)
header of a/ ldshread, ldnshread: read an indexednamed section ..... ldshread(3X)
structure read_constab: read constab entries into a ..... read_constab(3SEQ)
file system independent/ getdents: read directory entries and put in a ..... getdents(2)
read: read from file ..... read(2)
mail, rmail: send mail to users or read mail ..... mail(1)
line: read one line ..... line(1)

read: read from file ..... read(2)
of an archive file ldahread: read the archive header of a member..... ldahread(3X)
object file ldfhread: read the file header of a common..... ldfhread(3X)
readlink: read value of a symbolic link ..... readlink(2SEQ)
readv: read vector from file..... readv(2SEQ)
into a structure read_constab: read constab entries..... read_constab(3SEQ)
directory: opendir, readdir, telldir, seekdir, rewinddir/..... directory(3X)
gets longest line readfile, longline: reads file and ..... readfile(1F)
open a common object file for reading..... ldopen(3X)
open: open for reading or writing ..... open(2)
link readlink: read value of a symbolic ..... readlink(2SEQ)
readfile, longline: reads file and gets longest line ..... readfile(1F)
readv: read vector from file..... readv(2SEQ)
lseek: move read/write file pointer ..... lseek(2)
tirdwr: Transport Interface read/write interface STREAMS module..... tirdwr(7)
/get real user, effective user, real group, and effective group IDs..... getuid(2)
/geteuid, getgid, getegid: get real user, effective user, real/..... getuid(2)
malloc, free, realloc, calloc: main memory allocator ..... malloc(3C)
fast main memory/ malloc, free, realloc, calloc, mallopt, mallinfo:..... malloc(3X)
rc: reboot and shutdown command scripts..... rc(4)
t_rcvudata: receive a data unit..... t_rcvudata(3N)
t_rcvuderr: receive a unit data error indication ..... t_rcvuderr(3N)
over a connection t_rcv: receive data or expedited data sent ..... t_rcv(3N)
connect request t_rcvconnect: receive the confirmation from a ..... t_rcvconnect(3N)
lockf: record locking on files..... lockf(3C)
ed, red: text editor ..... ed(1)
setcolor: redefine or create a color..... setcolor(1F)
dltsectbl: refresh security table..... dltsectbl(1M)
regular expression regcmp, regex: compile and execute..... regcmp(3X)
regcmp: regular expression compile ..... regcmp(1)
make: maintain, update, and regenerate groups of programs..... make(1)
expression regcmp, regex: compile and execute regular..... regcmp(3X)
string regex: match patterns against a ..... regex(1F)
and match routines regexp: regular expression compile..... regexp(5)
regcmp, regex: compile and execute regular expression..... regcmp(3X)
regcmp: regular expression compile ..... regcmp(1)
routines regexp: regular expression compile and match..... regexp(5)
a file for a pattern using full regular expressions..... egrep(1)
reinit: run an initialization file..... reinit(1F)
accept, reject: allow or prevent LP requests ..... accept(1M)
comm: select or reject lines common to two sorted files ..... comm(1)
join: relational database operator ..... join(1)
t_sndrel: initiate an orderly release..... t_sndrel(3N)
acknowledge receipt of an orderly release indication..... t_rcvrel(3N)
common object file reloc: relocation information for a..... reloc(4)
ldrseek, ldnrseek: seek to relocation entries of a section of a common/..... ldrseek(3X)
reloc: relocation information for a common object file ..... reloc(4)
/ceil, fmod, fabs: floor, ceiling, remainder, absolute value functions..... floor(3M)
calendar: reminder service..... calendar(1)
uuxqt: execute remote command requests ..... uuxqt(1M)
Uutry: try to contact remote system with debugging on ..... uutry(1M)
ct: spawn getty to a remote terminal..... ct(1C)

rmdel: remove a delta from an SCCS file.....rmdel(1)
rmdir: remove a directory.....rmdir(2)
rmvopt: remove a menu option.....rmvopt(1M)
set, or shared memory ID ipcrm: remove a message queue, semaphore.....ipcrm(1)
unlink: remove directory entry.....unlink(2)
rm, rmdir: remove files or directories.....rm(1)
password entries pwfix: remove incorrectly formatted.....pwfix(1M)
deroff: remove nroff, tbl, and eqn constructs.....deroff(1)
rename: change the name of a file.....rename(2)
consistency check and interactive repair.....fsck(1M)
uniq: report repeated lines in a file.....uniq(1)
clock: report CPU time used.....clock(3C)
fsstat: report filesystem status.....fsstat(1M)
facilities status ipcs: report inter-process communication.....ipcs(1)
df: report number of free disk blocks and inodes.....df(1M)
mailbug: submit a bug report or enhancement request.....mailbug(1)
sar: sa1, sa2, sadc: system activity report package.....sar(1M)
timex: time a command; report process data and system activity.....timex(1)
ps: report process status.....ps(1)
uniq: report repeated lines in a file.....uniq(1)
fseek, rewind, ftell: reposition a file pointer in a stream.....fseek(3S)
filesystem repquota: summarize quotas for a.....repquota(1M)
submit a bug report or enhancement request.....mailbug(1)
t_accept: accept a connect request.....t_accept(3N)
t_listen: listen for a connect request.....t_listen(3N)
the confirmation from a connect request.....t_rcvconnect(3N)
send user-initiated disconnect request.....t_snddis(3N)
accept, reject: allow or prevent LP requests.....accept(1M)
the LP print service and move requests.....lpsched(1M)
uuxqt: execute remote command requests.....uuxqt(1M)
lp, cancel: send/cancel requests to the LP print service.....lp(1)
to its default values reset: reset the current form field.....reset(1F)
default values reset: reset the current form field to its.....reset(1F)
get_process_stats: get resource utilization info.....get_process_stats(2SEQ)
restore: incremental filesystem restore.....restore(1M)
restore restore: incremental filesystem.....restore(1M)
/ksh, rksh – KornShell, a standard/ restricted command and programming/.....ksh(1)
sh, rsh: shell, the standard/ restricted command programming/.....sh(1)
shadow: restricted password file.....shadow(4)
/m_rele_procs: suspend and resume child process execution.....m_park_procs(3PPS)
usage examples usage: retrieve a command description and.....usage(1)
t_rcvdis: retrieve information from disconnect.....t_rcvdis(3N)
object file symbol table/ ldgetname: retrieve symbol name for common.....ldgetname(3X)
abs: return integer absolute value.....abs(3C)
logname: return login name of user.....logname(3X)
m_get_numprocs: return number of child procs.....m_get_numprocs(3PPS)
m_get_myid: return process identification.....m_get_myid(3PPS)
getenv: return value for environment name.....getenv(3C)
stat: data returned by stat system call.....stat(5)
marked menu items getitems: returns a list of the currently.....getitems(1F)
getfrm: returns the current frame number.....getfrm(1F)
cpus_online: returns the number of CPUs on-line.....cpus_online(3PPS)

col: filter reverse line-feeds.....col(1)
ci: check in RCS revisions.....ci(1)
co: check out RCS revisions.....co(1)
rcsdiff: compare RCS revisions.....rcsdiff(1)
rcsmerge: merge RCS revisions.....rcsmerge(1)
pointer in a stream fseek, rewind, ftell: reposition a file.....fseek(3S)
/opendir, readdir, telldir, seekdir, rewinddir, closedir: directory/.....directory(3X)
creat: create a new file or rewrite an existing one.....creat(2)
restricted command/ ksh, rksh: KornShell, a standard/.....ksh(1)
information about RCS files rlog: print log messages and other.....rlog(1)
directories rm, rmdir: remove files or.....rm(1)
mail, rmail: send mail to users or read mail.....mail(1)
file rmdel: remove a delta from an SCCS.....rmdel(1)
rmdir: remove a directory.....rmdir(2)
rm, rmdir: remove files or directories.....rm(1)
rmvopt: remove a menu option.....rmvopt(1M)
chroot: change root directory.....chroot(2)
chroot: change root directory for a command.....chroot(1M)
logarithm, power, square root functions.....exp(3M)
queue menumsg: route user message to menu message.....menumsg(1M)
ldfcn: common object file access routines.....ldfcn(4)
regular expression compile and match routines.....regexp(5)
programming language sh, rsh: shell, standard/restricted command/.....sh(1)
nice: run a command at low priority.....nice(1)
nohup: run a command immune to hangups and quits.....nohup(1)
shell: run a command using shell.....shell(1F)
noage, nopff, noswap, onproc: run a command with priority aging, PFF/.....noage(1M)
the operating system boot: run a standalone program or bring up.....boot(8)
run: run an executable.....run(1F)
edc: run an externally described command.....edc(1M)
reinit: run an initialization file.....reinit(1F)
setpgrp: run command in its own process group.....setpgrp(1)
multi-user environment rc2: run commands performed for.....rc2(1M)
operating system rc0: run commands performed to stop the.....rc0(1M)
runacct: run daily accounting.....runacct(1M)
run: run an executable.....run(1F)
runacct: run daily accounting.....runacct(1M)
Administration SA: devices administered by System.....sa(7)
sar: sa1, sa2, sadc: system activity report package.....sar(1M)
sar: sa1, sa2, sadc: system activity report package.....sar(1M)
editing activity sact: print current SCCS file.....sact(1)
sar: sa1, sa2, sadc: system activity report package.....sar(1M)
report package sar: sa1, sa2, sadc: system activity.....sar(1M)
sar: system activity reporter.....sar(1)
savecore: save a core dump of the operating system.....savecore(1M)
operating system savecore: save a core dump of the.....savecore(1M)
brk, sbrk: change data segment space allocation.....brk(2)
brk, sbrk: change private data segment size.....brk(3PPS)
formatted input scanf, fscanf, sscanf: convert.....scanf(3S)
bfs: big file scanner.....bfs(1)
awk: pattern scanning and processing language.....awk(1)
nawk: pattern scanning and processing language.....nawk(1)

change the delta commentary of an SCCS delta ..... cdc(1)
comb: combine SCCS deltas ..... comb(1)
delta: make a delta (change) to an SCCS file ..... delta(1)
get: get a version of an SCCS file ..... get(1)
prs: print an SCCS file ..... prs(1)
rmdel: remove a delta from an SCCS file ..... rmdel(1)
sccsdiff: compare two versions of an SCCS file ..... sccsdiff(1)
sccsfile: format of SCCS file ..... sccsfile(4)
sccstorcs: build RCS file from SCCS file ..... sccstorcs(1)
unget: undo a previous get of an SCCS file ..... unget(1)
val: validate SCCS file ..... val(1)
sact: print current SCCS file editing activity ..... sact(1)
admin: create and administer SCCS files ..... admin(1)
what: identify SCCS files ..... what(1)
SCCS file sccsdiff: compare two versions of an ..... sccsdiff(1)
sccsfile: format of SCCS file ..... sccsfile(4)
file sccstorcs: build RCS file from SCCS ..... sccstorcs(1)
board SCED: SCSI/Ethernet/Diagnostics ..... sced(7)
ckbupscd: check filesystem backup schedule ..... ckbupscd(1M)
transport program uusched: the scheduler for the uucp file ..... uusched(1M)
unlock locks s_init_lock, s_lock, s_clock, s_unlock: initialize, lock, ... s_lock(3PPS)
object file scnhdr: section header for a common ..... scnhdr(4)
image file scr_dump: format of curses screen ..... scr_dump(4)
more: view a file one full screen at a time ..... more(1)
curses: terminal screen handling and optimization package ..... curses(3X)
scr_dump: format of curses screen image file ..... scr_dump(4)
vi: screen-oriented (visual) display editor based on ex ..... vi(1)
inittab: script for the init process ..... inittab(4)
rc: reboot and shutdown command scripts ..... rc(4)
tg: SCSI 1/2-inch magnetic tape interface ..... tg(7)
sd: SCSI disk controller ..... sd(7)
wd: SSM SCSI disk controller ..... wd(7)
CCSformat: standalone embedded SCSI disk formatter ..... ccsformat(8)
tm: SCSI tape controller ..... tm(7)
ts: SCSI tape controller ..... ts(7)
SCED: SCSI/Ethernet/Diagnostics board ..... sced(7)
getscsiinfo, getscsimatch: get scsiinfo structure ..... getscsiinfo(3SEQ)
sd: SCSI disk controller ..... sd(7)
program sdiff: side-by-side difference ..... sdiff(1)
fgrep: search a file for a character string ..... fgrep(1)
grep: search a file for a pattern ..... grep(1)
full regular expressions egrep: search a file for a pattern using ..... egrep(1)
bsearch: binary search a sorted table ..... bsearch(3C)
acctcom: search and print process accounting files ..... acctcom(1)
lsearch, lfind: linear search and update ..... lsearch(3C)
pathconv: search Interpreter criteria for filename ..... pathconv(1F)
hcreate, hdestroy: manage hash search tables ..... hsearch(3C)
tfind, tdelete, twalk: manage binary search trees ..... tsearch(3C)
manipulate the object file comment section ..... mcs(1)
m_multi: mark single-process code section ..... m_single(3PPS)
scnhdr: section header for a common object file ..... scnhdr(4)
/ldnshread: read an indexednamed section header of a common object/ ..... ldshread(3X)

/seek to line number entries of a section of a common object file ..... ldlseek(3X)
/seek to relocation entries of a section of a common object file ..... ldrseek(3X)
/ldnsseek: seek to an indexednamed section of a common object file ..... ldsseek(3X)
dltsectbl: refresh security table ..... dltsectbl(1M)
task: global security table ..... task(4)
tasktbl: create security table in shared memory ..... tasktbl(1M)
sed: stream editor ..... sed(1)
/nrand48, mrand48, jrand48, srand48, seed48, lcong48: generate uniformly/ ..... drand48(3C)
a common object/ ldsseek, ldnsseek: seek to an indexednamed section of ..... ldsseek(3X)
section of a/ ldlseek, ldnlseek: seek to line number entries of a ..... ldlseek(3X)
ldrseek, ldnrseek: seek to relocation entries of a section of a/ ..... ldrseek(3X)
a common object file ldohseek: seek to the optional file header of ..... ldohseek(3X)
object file ldtbseek: seek to the symbol table of a common ..... ldtbseek(3X)
/opendir, readdir, telldir, seekdir, rewinddir, closedir/ ..... directory(3X)
sweepoff: enable/disable patrol seeks on DCC disks ..... sweepon(1M)
shmget: get shared memory segment identifier ..... shmget(2)
brk, sbrk: change private data segment size ..... brk(3PPS)
shbrk, shsbrk: change shared data segment size ..... shbrk(3PPS)
brk, sbrk: change data segment space allocation ..... brk(2)
sorted files comm: select or reject lines common to two ..... comm(1)
greek: select terminal filter ..... greek(1)
addformopt: add a form/ SELECT window option to a menu ..... addformopt(1M)
semctl: semaphore control operations ..... semctl(2)
semop: semaphore operations ..... semop(2)
ipcrm: remove a message queue, semaphore set, or shared memory ID ..... ipcrm(1)
semget: get set of semaphores ..... semget(2)
semctl: semaphore control operations ..... semctl(2)
semget: get set of semaphores ..... semget(2)
semop: semaphore operations ..... semop(2)
t_sndudata: send a data unit ..... t_sndudata(3N)
putmsg: send a message on a stream ..... putmsg(2)
kill: send a signal to a process or a group of processes ..... kill(2)
tfnotify: send completion message ..... tfnotify(1M)
connection t_snd: send data or expedited data over a ..... t_snd(3N)
mail, rmail: send mail to users or read mail ..... mail(1)
t_snddis: send user-initiated disconnect request ..... t_snddis(3N)
lp, cancel: send/cancel requests to the LP print service ..... lp(1)
receive data or expedited data sent over a connection ..... t_rcv(3N)
intro: introduction to Sequent system calls ..... intro(2SEQ)
dis386: disassemble Series 386 instructions ..... dis386(3SEQ)
SSM: System Services Module ..... ssm(7)
setsid: create session and set process group ID ..... setsid(2)
ascii: map of ASCII character set ..... ascii(5)
alarm: set a process alarm clock ..... alarm(2)
umask: set and get file creation mask ..... umask(2)
stty, gtty: set and get terminal state (defunct) ..... stty(2)
in core or in files set, unset: set and unset environment variables ..... set(1F)
sigstack: set and/or get signal stack context ..... sigstack(2SEQ)
timezone: set default system time zone ..... timezone(4)
backup.info: backup set definitions ..... backup_info(4)
setdtablesize: set descriptor table size ..... setdtablesize(2SEQ)
env: set environment for command execution ..... env(1)

utime: set file access and modification times.....utime(2)
umask: set file-creation mode mask.....umask(1)
tcsetpgrp: set foreground process group ID.....tcsetpgrp(2)
setgroups: set group access list.....setgroups(2SEQ)
cpset: copy file and set mode, user, and group IDs.....cpset(1)
m_set_procs: set number of child processes.....m_set_procs(3PPS)
semget: get set of semaphores.....semget(2)
remove a message queue, semaphore set, or shared memory ID.....ipcrm(1)
lpusers: set printing queue priorities.....lpusers(1M)
setpgrp: set process group ID.....setpgrp(2)
setsid: create session and set process group ID.....setsid(2)
setlocale: set programs location.....setlocale(3C)
tabs: set tabs on a terminal.....tabs(1)
tcgetattr, tcsetattr: get/ set terminal state.....tcgetattr(2)
line discipline getty: set terminal type, modes, speed, and.....getty(1M)
line discipline uugetty: set terminal type, modes, speed, and.....uugetty(1M)
date: print and set the date.....date(1)
stty: set the options for a terminal.....stty(1)
stime: set time.....stime(2)
environment variables in core or in/ set, unset: set and unset.....set(1F)
setuid, setgid: set user and group IDs.....setuid(2)
ulimit: get and set user limits.....ulimit(2)
a stream setbuf, setvbuf: assign buffering to.....setbuf(3S)
/toascci, _tolower, _toupper, setchrclass: character handling.....ctype(3C)
setcolor: redefine or create a color.....setcolor(1F)
size setdtablesize: set descriptor table.....setdtablesize(2SEQ)
setuid, setgid: set user and group IDs.....setuid(2)
getgrent, getgrgid, getgrnam, setgrent, endgrent, fgetgrent: get group/.....getgrent(3C)
setgroups: set group access list.....setgroups(2SEQ)
setjmp, longjmp: non-local goto.....setjmp(3C)
crypt, setkey, encrypt: generate hashing encryption.....crypt(3C)
setlocale: set programs location.....setlocale(3C)
setmnt: establish mount table.....setmnt(1M)
/endmntent, getmntent, hasmntopt, setmntent: get file system/.....getmntent(3SEQ)
setpgid: manipulate signal sets.....setpgid(2)
process group setpgrp: run command in its own.....setpgrp(1)
setpgrp: set process group ID.....setpgrp(2)
getpwent, getpwuid, getpwnam, setpwent, endpwent, fgetpwent: get/.....getpwent(3C)
process group ID setsid: create session and set.....setsid(2)
getspent, getspnam, setspent, endspent, fgetspent, lckpwdf,/.....getspent(3X)
profile: setting up an environment at login time.....profile(4)
gettydefs: speed and terminal settings used by getty.....gettydefs(4)
IDs setuid, setgid: set user and group.....setuid(2)
user setup: initialize system for first.....setup(1)
/getutid, getutline, pututline, setutent, endutent, utmpname,/.....getut(3C)
setbuf, setvbuf: assign buffering to a stream.....setbuf(3S)
machine-independent fashion sputl, sgetl: access long integer data in a.....sputl(3X)
standard/restricted command/ sh, rsh: shell, the.....sh(1)
/fgetspent, lckpwdf, ulckpwdf: get shadow password file entry.....getspent(3X)
putspent: write shadow password file entry.....putspent(3X)
shadow: restricted password file.....shadow(4)
pwconv: installs and updates /etc/ shadow with information from/.....pwconv(1M)

shbrk, shsbrk: change shared data segment size ..... shbrk(3PPS)
tasktbl: create security table in shared memory ..... tasktbl(1M)
shmalloc, shrealloc, shfree: shared memory allocator ..... shmalloc(3PPS)
shmctl: shared memory control operations ..... shmctl(2)
a message queue, semaphore set, or shared memory ID ..... ipcrm(1)
shmop, shmat, shmdt: shared memory operations ..... shmop(2)
shmget: get shared memory segment identifier ..... shmget(2)
segment size shbrk, shsbrk: change shared data ..... shbrk(3PPS)
shell: run a command using shell ..... shell(1F)
assign menu system as user login shell ..... tfmenush(1M)
system: issue a shell command ..... system(3S)
csh: a shell (command interpreter) with C-like syntax ..... csh(1)
/shutacct, startup, turnacct: shell procedures for accounting ..... acctsh(1M)
shell: run a command using shell ..... shell(1F)
shells: shell table ..... shells(4)
command programming/ sh, rsh: shell, the standard/restricted ..... sh(1)
shells: shell table ..... shells(4)
shmalloc, shrealloc, shfree: shared memory allocator ..... shmalloc(3PPS)
memory allocator shmalloc, shrealloc, shfree: shared ..... shmalloc(3PPS)
shmop, shmat, shmdt: shared memory operations ..... shmop(2)
operations shmctl: shared memory control ..... shmctl(2)
shmop, shmat, shmdt: shared memory operations ..... shmop(2)
identifier shmget: get shared memory segment ..... shmget(2)
operations shmop, shmat, shmdt: shared memory ..... shmop(2)
the machine showcfg: show the configuration of ..... showcfg(1M)
allocator shmalloc, shrealloc, shfree: shared memory ..... shmalloc(3PPS)
shbrk, shsbrk: change shared data segment size ..... shbrk(3PPS)
shutdown: shut down system, change system state ..... shutdown(1M)
/nulladm, prctmp, prdaily, prtacct, shutacct, startup, turnacct: shell/ ..... acctsh(1M)
rc: reboot and shutdown command scripts ..... rc(4)
shutdown: system shutdown information file ..... shutdown(4)
system state shutdown: shut down system, change ..... shutdown(1M)
information file shutdown: system shutdown ..... shutdown(4)
actions sigaction: examine and change signal ..... sigaction(2)
manipulate/ sigemptyset, sigfillset, sigaddset, sigdelset, sigismember: ..... sigemptyset(2)
sigcontext: get signal context ..... sigcontext(2SEQ)
sigemptyset, sigfillset, sigaddset, sigdelset, sigismember: manipulate/ ..... sigemptyset(2)
sigemptyset, sigfillset, sigaddset, ... sigemptyset(2)
sigismember/ sigemptyset, sigfillset, sigaddset, sigdelset, ... sigemptyset(2)
sigpause: signal management sighold, sigrelse, sigignore, ... sigset(2)
sigset, sighold, sigrelse, sigignore, sigpause: signal/ ..... sigset(2)
/sigfillset, sigaddset, sigdelset, sigismember: manipulate signal sets ..... sigemptyset(2)
sigsetjmp, siglongjmp: non-local jumps ..... sigsetjmp(3C)
login: sign on ..... login(1)
pause: suspend process until signal ..... pause(2)
specify what to do upon receipt of a signal ..... signal(2)
sigsuspend: wait for a signal ..... sigsuspend(2)
sigaction: examine and change signal actions ..... sigaction(2)
sigcontext: get signal context ..... sigcontext(2SEQ)
sigrelse, sigignore, sigpause: signal management ..... sigset(2)
setpgid: manipulate signal sets ..... setpgid(2)
sigdelset, sigismember: manipulate signal sets ..... sigemptyset(2)

receipt of a signal — signal: specify what to do upon ..... signal(2)
sigstack: set and/or get — signal stack context ..... sigstack(2SEQ)
kill: send a — signal to a process or a group of processes ..... kill(2)
sigpending: examine pending — signals ..... sigpending(2)
examine and change blocked — signals ..... sigprocmask(2)
ssignal, gsignal: software — signals ..... ssignal(3C)
sighold, sigrelse, sigignore, — sigpause: signal management ..... sigset(2)
— sigpending: examine pending signals ..... sigpending(2)
blocked signals — sigprocmask: examine and change ..... sigprocmask(2)
signal management sigset, sighold, — sigrelse, sigignore, sigpause: ..... sigset(2)
sigignore, sigpause: signal/ — sigset, sighold, sigrelse, ... sigset(2)
jumps — sigsetjmp, siglongjmp: non-local ..... sigsetjmp(3C)
stack context — sigstack: set and/or get signal ..... sigstack(2SEQ)
— sigsuspend: wait for a signal ..... sigsuspend(2)
atan2: trigonometric/ trig: — sin, cos, tan, asin, acos, atan, ... trig(3M)
m_single, m_multi: mark — single-process code section ..... m_single(3PPS)
functions — sinh, cosh, tanh: hyperbolic ..... sinh(3M)
initialize barrier, wait at barrier — s_init_barrier, s_wait_barrier: ..... s_wait_barrier(3PPS)
s_unlock: initialize, lock, unlock/ — s_init_lock, s_lock, s_clock, ... s_lock(3PPS)
sbrk: change private data segment — size ..... brk(3PPS)
getdtablesize: get descriptor table — size ..... getdtablesize(2SEQ)
getpagesize: get system page — size ..... getpagesize(2SEQ)
setdtablesize: set descriptor table — size ..... setdtablesize(2SEQ)
shsbrk: change shared data segment — size ..... shbrk(3PPS)
of object files — size: print section sizes ..... see *C and Lang Tools Binder*
interval — sleep: suspend execution for an ..... sleep(1)
interval — sleep: suspend execution for ..... sleep(3C)
s_init_lock, — s_lock, s_clock, s_unlock: initialize, lock/ ..... s_lock(3PPS)
ttyslot: find the — slot in the utmp file of the current user ..... ttyslot(3C)
consmem, — sm, ss: console memory drivers ..... consmem(7)
ssignal, gsignal: — software signals ..... ssignal(3C)
qsort: quicker — sort ..... qsort(3C)
tsort: topological — sort ..... tsort(1)
sort: — sort and/or merge files ..... sort(1)
— sort: sort and/or merge files ..... sort(1)
select or reject lines common to two — sorted files ..... comm(1)
bsearch: binary search a — sorted table ..... bsearch(3C)
an object file list: produce C — source listing from ..... see *C and Lang Tools Binder*
— sp: STREAMS pipe device ..... sp(7)
open file into the processs address — space ..... mmap(2SEQ)
unmap part of the processs address — space ..... munmap(2SEQ)
into processs virtual address — space ..... pmap(7)
brk, sbrk: change data segment — space allocation ..... brk(2)
ct: — spawn getty to a remote terminal ..... ct(1C)
memory efficient way vfork: — spawn new process in a virtual ..... vfork(2)
fspec: format — specification in text files ..... fspec(4)
signal: — specify what to do upon receipt of a signal ..... signal(2)
getty: set terminal type, modes, — speed, and line discipline ..... getty(1M)
uugetty: set terminal type, modes, — speed, and line discipline ..... uugetty(1M)
gettydefs: — speed and terminal settings used by getty ..... gettydefs(4)
find spelling errors — spell, hashmake, spellin, hashcheck: ..... spell(1)
file — spellhist: unmatched spelling words ..... spellhist(4)

spell, hashmake,' spellin, hashcheck: find spelling errors.....spell(1)
hashmake, spellin, hashcheck: find spelling errors.....spell(1)
spellhist: unmatched spelling words file.....spellhist(4)
csplit: context split.....csplit(1)
split: split a file into pieces.....split(1)
fsplit: split f77, ratfor, or efl files.....fsplit(1)
split: split a file into pieces.....split(1)
uucleanup: uucp spool directory clean-up.....uucleanup(1M)
printf, fprintf, sprintf: print formatted output.....printf(3S)
data in a machine-independent/ sputl, sgetl: access long integer.....sputl(3X)
square root/ exp, log, log10, pow, sqrt: exponential, logarithm, power,.....exp(3M)
sqrt: exponential, logarithm, power, square root functions.....exp(3M)
rand, srand: simple random-number generator.....rand(3C)
/lrand48, nrand48, mrand48, jrand48, srand48, seed48, lcong48: generate/.....drand48(3C)
consmem, sm, ss: console memory drivers.....consmem(7)
scanf, fscanf, sscanf: convert formatted input.....scanf(3S)
ssignal, gsignal: software signals.....ssignal(3C)
wp: SSM parallel line printer interface.....wp(7)
wd: SSM SCSI disk controller.....wd(7)
SSM: System Services Module.....ssm(7)
terminal multiplexor driver st: Systech MTI-800/1600/1650.....st(7)
sigstack: set and/or get signal stack context.....sigstack(2SEQ)
copy: perform a standalone data transfer.....copy(8)
prtvtoc: print the VTOC of a standalone disk device.....prtvtoc(8)
dual-channel disk/ zdformat: standalone disk formatter for.....zdformat(8)
CCSformat: standalone embedded SCSI disk formatter.....ccsformat(8)
dump: standalone memory dump.....dump(8)
operating system boot: run a standalone program or bring up the.....boot(8)
intro: introduction to system standalone utilities.....intro(8)
stdio: standard buffered input/output package.....stdio(3S)
package stdipc: ftok: standard interprocess communication.....stdipc(3C)
sh, rsh: shell, the standard/restricted command programming/.....sh(1)
operating system for beginning/ starter: information about the.....starter(1)
lpsched, lpshut, lpmove: start/stop the LP print service and move/.....lpsched(1M)
/prctmp, prdaily, prtacct, shutacct, startup, turnacct: shell procedures/.....acctsh(1M)
call stat: data returned by stat system.....stat(5)
stat, lstat, fstat: get file status.....stat(2)
stat: data returned by stat system call.....stat(5)
information statfs, fstatfs: get filesystem.....statfs(2)
ustat: get filesystem statistics.....ustat(2)
ff: list filenames and statistics for a filesystem.....ff(1M)
fsstat: report filesystem status.....fsstat(1M)
communication facilities status.....ipcs(1)
ps: report process status.....ps(1)
stat, lstat, fstat: get file status.....stat(2)
feof, clearerr, fileno: stream status inquiries.....ferror(3S)
uustat: uucp status inquiry and job control.....uustat(1C)
lpstat: print information about the status of the LP print service.....lpstat(1)
input/output package stdio: standard buffered.....stdio(3S)
communication package stdipc: ftok: standard interprocess.....stdipc(3C)
stime: set time.....stime(2)
waitpid: wait for a child process to stop or terminate.....wait(2)

lpsched, lpshut, lpmove: start/ stop the LP print service and move/ ..... lpsched(1M)
rc0: run commands performed to stop the operating system ..... rc0(1M)
icheck: filesystem storage consistency check ..... icheck(1M)
strace: print STREAMS trace messages ..... strace(1M)
strncmp, strcpy, strncpy,/ string: strcat, strdup, strncat, strcmp, ... string(3C)
/strncmp, strcpy, strncpy, strlen, strchr, strrchr, strpbrk, strspn,/ ..... string(3C)
cleanup program strclean: STREAMS error logger ..... strclean(1M)
string: strcat, strdup, strncat, strcmp, strncmp, strcpy,/ ..... string(3C)
/strdup, strncat, strcmp, strncmp, strcpy, strncpy, strlen, strchr,/ ..... string(3C)
/strchr, strrchr, strpbrk, strspn, strcspn, strtok: string operations ..... string(3C)
strcpy, strncpy,/ string: strcat, strdup, strncat, strcmp, strncmp, ... string(3C)
fclose, fflush: close or flush a stream ..... fclose(3S)
fopen, freopen, fdopen: open a stream ..... fopen(3S)
reposition a file pointer in a stream ..... fseek(3S)
getw: get character or word from a stream ..... getc(3S)
getmsg: get next message off a stream ..... getmsg(2)
gets, fgets: get a string from a stream ..... gets(3S)
putw: put character or word on a stream ..... putc(3S)
putmsg: send a message on a stream ..... putmsg(2)
puts, fputs: put a string on a stream ..... puts(3S)
setvbuf: assign buffering to a stream ..... setbuf(3S)
push character back into input stream ..... ungetc(3S)
sed: stream editor ..... sed(1)
ferror, feof, clearerr, fileno: stream status inquiries ..... ferror(3S)
streamio: STREAMS ioctl commands ..... streamio(7)
driver streampty: STREAMS pseudo terminal ..... streampty(7)
clone: open any minor device on a STREAMS driver ..... clone(7)
strclean: STREAMS error logger cleanup program ..... strclean(1M)
strerr: STREAMS error logger daemon ..... strerr(1M)
log: interface to STREAMS error logging and event tracing ..... log(7)
poll: STREAMS input/output multiplexing ..... poll(2)
streamio: STREAMS ioctl commands ..... streamio(7)
Transport Interface cooperating STREAMS module ..... timod(7)
Interface read/write interface STREAMS module ..... tirdwr(7)
sp: STREAMS pipe device ..... sp(7)
streampty: STREAMS pseudo terminal driver ..... streampty(7)
strace: print STREAMS trace messages ..... strace(1M)
streamtty: Generic STREAMS tty driver interface ..... streamtty(7)
ld0: STREAMS tty line discipline ..... ld0(7)
driver interface streamtty: Generic STREAMS tty ..... streamtty(7)
strerr: STREAMS error logger daemon ..... strerr(1M)
strftime: format time information ..... strftime(3C)
long integer and base-64 ASCII string ..... a64l(3C)
tzset: convert date and time to string ..... ctime(3C)
convert floating-point number to string ..... ecvt(3C)
fgrep: search a file for a character string ..... fgrep(1)
regex: match patterns against a string ..... regex(1F)
gps: graphical primitive string, format of graphical files ..... gps(4)
gets, fgets: get a string from a stream ..... gets(3S)
puts, fputs: put a string on a stream ..... puts(3S)
echo: put string on virtual output ..... echo(1F)
strpbrk, strspn, strcspn, strtok: string operations ..... string(3C)

strcmp, strncmp, strcpy, strncpy/ string: strcat, strdup, strncat, ...string(3C)
strtod, atof: convert string to double-precision number.....strtod(3C)
strtol, atol, atoi: convert string to integer.....strtol(3C)
cftime: language specific strings.....cftime(4)
object file strip: strip an.....see *C and Lang Tools Binder*
/strcmp, strncmp, strcpy, strncpy, strlen, strchr, strrchr, strpbrk,/.....string(3C)
string: strcat, strdup, strncat, strcmp, strncmp, strcpy, strncpy/.....string(3C)
/strcat, strdup, strncat, strcmp, strncmp, strcpy, strncpy, strlen,/.....string(3C)
/strncat, strcmp, strncmp, strcpy, strncpy, strlen, strchr, strrchr,/.....string(3C)
/strncpy, strlen, strchr, strrchr, strpbrk, strspn, strcspn, strtok:/.....string(3C)
/strcpy, strncpy, strlen, strchr, strrchr, strpbrk, strspn, strcspn,/.....string(3C)
/strlen, strchr, strrchr, strpbrk, strspn, strcspn, strtok: string/.....string(3C)
character strstr: find first occurrence of.....strstr(3C)
double-precision number strtod, atof: convert string to.....strtod(3C)
strrchr, strpbrk, strspn, strcspn, strtok: string operations.....string(3C)
to integer strtol, atol, atoi: convert string.....strtol(3C)
processes using a file or file structure.....fuser(1M)
getscsimatch: get scsiinfo structure.....getscsiinfo(3SEQ)
getzdinfobyname: get zdinfo structure.....getzdinfo(3SEQ)
read constab entries into a structure.....read_constab(3SEQ)
t_alloc: allocate a library structure.....t_alloc(3N)
t_free: free a library structure.....t_free(3N)
state (defunct) stty, gtty: set and get terminal.....stty(2)
stty: set the options for a terminal.....stty(1)
su: become superuser or another user.....su(1M)
backup.codes1: backup subcode definitions.....backup_codes1(4)
m_fork: execute a subprogram in parallel.....m_fork(3PPS)
dbm_clearerr, dmb_blkflush: database subroutines.....ndbm(3X)
plot: graphics interface subroutines.....plot(3X)
sulog: log of substitute user events.....sulog(4)
sulog: log of substitute user events.....sulog(4)
of a file sum: print checksum and block count.....sum(1)
du: summarize disk usage.....du(1M)
repquota: summarize quotas for a filesystem.....repquota(1M)
records acctcms: command summary from per-process accounting.....acctcms(1M)
s_init_lock, s_lock, s_clock, s_unlock: initialize, lock, unlock locks.....s_lock(3PPS)
sync: update the superblock.....sync(1M)
sync: update superblock.....sync(2)
su: become superuser or another user.....su(1M)
getgroups: get supplementary group IDs.....getgroups(2)
uucp: networking support files.....uucp(4)
lptypes: table of printer types supported by the menu system.....lptypes(4)
m_park_procs, m_rele_procs: suspend and resume child process/.....m_park_procs(3PPS)
sleep: suspend execution for an interval.....sleep(1)
sleep: suspend execution for interval.....sleep(3C)
pause: suspend process until signal.....pause(2)
swab: swap bytes.....swab(3C)
wait at barrier s_init_barrier, s_wait_barrier: initialize barrier,.....s_wait_barrier(3PPS)
swap: swap administrative interface.....swap(1M)
swab: swap bytes.....swab(3C)
swap: swap administrative interface.....swap(1M)
/a command with priority aging, PFF, swapping, or processor migration/.....noage(1M)

seeks on DCC disks sweepon, sweepoff: enable/disable patrol ..... sweepon(1M)
patrol seeks on DCC disks sweepon, sweepoff: enable/disable ..... sweepon(1M)
symbol table/ ldgetname: retrieve symbol name for common object file ..... ldgetname(3X)
ldtbindex: compute the index of a symbol table entry ..... ldgetname(3X)
ldtbread: read an indexed symbol table entry of a common/ ..... ldtbindex(3X)
syms: common object file symbol table entry of a common object/ ..... ldtbread(3X)
ldtbseek: seek to the symbol table format ..... syms(4)
unistd: file header for symbol table of a common object file ..... ldtbseek(3X)
readlink: read value of a symbolic constants ..... unistd(4)
symlink: make symbolic link ..... readlink(2SEQ)
of common operating system terms and symbolic link to a file ..... symlink(2SEQ)
file symbols ..... glossary(1)
table format symlink: make symbolic link to a ..... symlink(2SEQ)
syms: common object file symbol ..... syms(4)
sync: update superblock ..... sync(2)
sync: update the superblock ..... sync(1M)
controlling FMLI object vsig: synchronize a co-process with the ..... vsig(1F)
t_sync: synchronize transport library ..... t_sync(3N)
check documents for correct eqn syntax ..... checkeq(1)
(command interpreter) with C-like syntax ..... csh(1)
administration sysadm: menu interface to do system ..... sysadm(1)
syscall: indirect system call ..... syscall(2)
variables sysconf: get configurable system ..... sysconf(2)
configuration file sys_config: format of a system ..... sys_config(4)
perror, errno, sys_errlist, sys_nerr: system error messages ..... perror(3C)
sysi86: machine specific functions ..... sysi86(2)
menu system functions syskeymap: identify keys to internal ..... syskeymap(4)
perror, errno, sys_errlist, sys_nerr: system error messages ..... perror(3C)
multiplexor driver st: Systech MTI-800/1600/1650 terminal ..... st(7)
lp: Systech parallel line printer interface ..... lp(7)
bsearch: binary search a sorted table ..... bsearch(3C)
constab: constructed devices table ..... constab(4)
devtab: virtual device table ..... devtab(4)
dltsectbl: refresh security table ..... dltsectbl(1M)
kernel auto-configuration device table ..... dumpconf(1M)
fstab: file-system- table ..... fstab(4)
mnttab: mounted filesystem table ..... mnttab(4)
setmnt: establish mount table ..... setmnt(1M)
shells: shell table ..... shells(4)
task: global security table ..... task(4)
name for common object file symbol table entry ..... ldgetname(3X)
/compute the index of a symbol table entry of a common object file ..... ldtbindex(3X)
ldtbread: read an indexed symbol table entry of a common object file ..... ldtbread(3X)
syms: common object file symbol table format ..... syms(4)
tasktbl: create security table in shared memory ..... tasktbl(1M)
get maximum in-use descriptor table index ..... getdtablemax(2SEQ)
ldtbseek: seek to the symbol table of a common object file ..... ldtbseek(3X)
the menu system lptypes: table of printer types supported by ..... lptypes(4)
getdtablesize: get descriptor table size ..... getdtablesize(2SEQ)
setdtablesize: set descriptor table size ..... setdtablesize(2SEQ)
classification and conversion tables ..... chrtbl(1M)
hdestroy: manage hash search tables ..... hsearch(3C)

tabs: set — tabs on a terminal ..... tabs(1)
tabs: set tabs on a terminal ..... tabs(1)
t_accept: accept a connect request ..... t_accept(3N)
ctags: create a — tags file ..... ctags(1)
file — tail: deliver the last part of a ..... tail(1)
structure — t_alloc: allocate a library ..... t_alloc(3N)
trig: sin, cos, — tan, asin, acos, atan, atan2: trigonometric/ ..... trig(3M)
sinh, cosh, — tanh: hyperbolic functions ..... sinh(3M)
tm: SCSI — tape controller ..... tm(7)
ts: SCSI — tape controller ..... ts(7)
hpio: Hewlett-Packard 2645A terminal — tape file archiver ..... hpio(1)
tar: — tape file archiver ..... tar(1)
mtio: UNIX magnetic — tape interface ..... mtio(7)
tg: SCSI 1/2-inch magnetic — tape interface ..... tg(7)
xt: Xylogics 472 MULITBUS magnetic — tape interface ..... xt(7)
mt: magnetic — tape manipulating program ..... mt(1)
tar: tape file archiver ..... tar(1)
task: global security table ..... task(4)
shared memory — tasktbl: create security table in ..... tasktbl(1M)
transport endpoint — t_bind: bind an address to a ..... t_bind(3N)
deroff: remove nroff, — tbl, and eqn constructs ..... deroff(1)
terminal state — tcgetattr, tcsetattr: get/set ..... tcgetattr(2)
group ID — tcgetpgrp: get foreground process ..... tcgetpgrp(2)
t_close: close a transport endpoint ..... t_close(3N)
with another transport user — t_connect: establish a connection ..... t_connect(3N)
tcgetattr, — tcsetattr: get/set terminal state ..... tcgetattr(2)
group ID — tcsetpgrp: set foreground process ..... tcsetpgrp(2)
tsearch, tfind, — tdelete, twalk: manage binary search trees ..... tsearch(3C)
tee: pipe fitting ..... tee(1)
4014: paginator for the — Tektronix 4014 terminal ..... 4014(1)
init, — telinit: process control initialization ..... init(1M)
directory: opendir, readdir, — telldir, seekdir, rewinddir/ ..... directory(3X)
temporary file — tempnam, — tempnam: create a name for a ..... tmpnam(3S)
tmpfile: create a — temporary file ..... tmpfile(3S)
tmpnam, tempnam: create a name for a — temporary file ..... tmpnam(3S)
terminals — term: conventional names for ..... term(5)
term: format of compiled — term file ..... term(4)
term: format of compiled term file ..... term(4)
description — captoinfo: convert a — termcap description into a terminfo ..... captoinfo(1M)
base — termcap: terminal capability data ..... termcap(4)
paginator for the Tektronix 4014 — terminal ..... 4014(1)
special functions of the DASI 450 — terminal ..... 450(1)
ct: spawn getty to a remote — terminal ..... ct(1C)
ctermid: generate filename for — terminal ..... ctermid(3S)
hangup the specified controlling — terminal ..... fvhangup(2SEQ)
stty: set the options for a — terminal ..... stty(1)
tabs: set tabs on a — terminal ..... tabs(1)
tty: get the name of the — terminal ..... tty(1)
ttyname, isatty: find name of a — terminal ..... ttyname(3C)
/associated with a terminal or the — terminal associated with a device ..... ttydevname(1)
termcap: — terminal capability data base ..... termcap(4)
terminfo: — terminal capability database ..... terminfo(4)

streampty: STREAMS pseudo terminal driver.....streampty(7)
greek: select terminal filter.....greek(1)
tty: controlling terminal (gentty) interface.....tty(7)
termio: general terminal interface.....termio(7)
dial: establish an out-going terminal line connection.....dial(3C)
st: Systech MTI-800/1600/1650 terminal multiplexor driver.....st(7)
xs: Xylogics 781 terminal multiplexor driver.....xs(7)
tput: initialize a terminal or query terminfo database.....tput(1)
/get the device associated with a terminal or the terminal associated/.....ttydevname(1)
optimization package curses: terminal screen handling and.....curses(3X)
gettydefs: speed and terminal settings used by getty.....gettydefs(4)
tcgetattr, tcsetattr: get/set terminal state.....tcgetattr(2)
stty, gtty: set and get terminal state (defunct).....stty(2)
hpio: Hewlett-Packard 2645A terminal tape file archiver.....hpio(1)
line discipline getty: set terminal type, modes, speed, and.....getty(1M)
line discipline uugetty: set terminal type, modes, speed, and.....uugetty(1M)
functions of DASI 300 and 300s terminals.....300(1)
getpseudotty: get pseudo terminals.....getpseudotty(3SEQ)
special functions of Hewlett-Packard terminals.....hp(1)
term: conventional names for terminals.....term(5)
wait for a child process to stop or terminate.....wait(2)
kill: terminate a process.....kill(1)
exit, _exit: terminate process.....exit(2)
tic: terminfo compiler.....tic(1M)
tput: initialize a terminal or query terminfo database.....tput(1)
convert a termcap description into a terminfo description.....captoinfo(1M)
infocmp: compare or print out terminfo descriptions.....infocmp(1M)
database terminfo: terminal capability.....terminfo(4)
termio: general terminal interface.....termio(7)
of common operating system terms and symbols.....glossary(1)
t_error: produce error message.....t_error(3N)
test: condition evaluation command.....test(1)
isnan: isnand, isnanf: test for floating-point NaN/.....isnan(3C)
ed, red: text editor.....ed(1)
ex: text editor.....ex(1)
edit: text editor (variant of ex for casual users).....edit(1)
errormsg: error message text file.....errormsg(4)
newform: change the format of a text file.....newform(1)
fspec: format specification in text files.....fspec(4)
isismgr: ISIS file manager, updates text files in ISIS file format.....isismgr(1M)
plock: lock process, text, or data in memory.....plock(2)
search trees tsearch, tfind, tdelete, twalk: manage binary.....tsearch(3C)
reporter tfipc: interprocess communication.....tfipc(1M)
· login shell tfmenush: assign menu system as user.....tfmenush(1M)
tfnotify: send completion message.....tfnotify(1M)
t_free: free a library structure.....t_free(3N)
interface tg: SCSI 1/2-inch magnetic tape.....tg(7)
service information t_getinfo: get protocol-specific.....t_getinfo(3N)
t_getstate: get the current state.....t_getstate(3N)
merge: three-way file merge.....merge(1)
tic: terminfo compiler.....tic(1M)
batch: execute commands at a later time.....at(1)

view a file one full screen at a — time ..... more(1)
setting up an environment at login — time ..... profile(4)
stime: set — time ..... stime(2)
time: get — time ..... time(2)
time, ptime: — time a command ..... time(1)
and system activity timex: — time a command; report process data ..... timex(1)
acctcon1, acctcon2; connect- — time accounting ..... acctcon(1M)
— time: get time ..... time(2)
strftime: format — time information ..... strftime(3C)
profil: execution — time profile ..... profil(2)
— time, ptime: time a command ..... time(1)
mktime, tzset: convert date and — time to string ..... ctime(3C)
clock: report CPU — time used ..... clock(3C)
timezone: set default system — time zone ..... timezone(4)
times: get process and child process — times ..... times(2)
set file access and modification — times ..... utime(2)
times — times: get process and child process ..... times(2)
update access and modification — times of a file ..... touch(1)
process data and system activity — timex: time a command; report ..... timex(1)
zone — timezone: set default system time ..... timezone(4)
cooperating STREAMS module — timod: Transport Interface ..... timod(7)
read/write interface STREAMS module — tirdwr: Transport Interface ..... tirdwr(7)
request — t_listen: listen for a connect ..... t_listen(3N)
a transport endpoint — t_look: look at the current event on ..... t_look(3N)
— tm: SCSI tape controller ..... tm(7)
processor — tmp_affinity: bind a process to a ..... tmp_affinity(2SEQ)
— tmp_ctl: multi-processor control ..... tmp_ctl(2SEQ)
— tmpfile: create a temporary file ..... tmpfile(3S)
temporary file — tmpnam, tempnam: create a name for a ..... tmpnam(3S)
/isgraph, isascii, tolower, toupper, — toascci, _tolower, _toupper/ ..... ctype(3C)
/tolower, _toupper, _tolower, — toascii: translate characters ..... conv(3C)
conv: toupper, tolower, _toupper, — _tolower, toascii: translate/ ..... conv(3C)
/isascii, tolower, toupper, toascci, — _tolower, _toupper, setchrclass:/ ..... ctype(3C)
/ispunct, isprint, isgraph, isascii, — tolower, toupper, toascci, _tolower/ ..... ctype(3C)
conv: toupper, — tolower, _toupper, _tolower, toascii: translate/ ..... conv(3C)
endpoint — t_open: establish a transport ..... t_open(3N)
tsort: — topological sort ..... tsort(1)
transport endpoint — t_optmgmt: manage options for a ..... t_optmgmt(3N)
acctmerg: merge or add — total accounting files ..... acctmerg(1M)
modification times of a file — touch: update access and ..... touch(1)
/tolower, toupper, toascci, _tolower, — _toupper, setchrclass: character/ ..... ctype(3C)
/isprint, isgraph, isascii, tolower, — toupper, toascci, _tolower/ ..... ctype(3C)
conv: toupper, tolower, — _toupper, _tolower, toascii: translate/ ..... conv(3C)
_tolower, toascii: translate/ conv: — toupper, tolower, _toupper, ... conv(3C)
terminfo database — tput: initialize a terminal or query ..... tput(1)
— tr: translate characters ..... tr(1)
ptrace, mptrace: process — trace facility ..... ptrace(2)
strace: print STREAMS — trace messages ..... strace(1M)
to STREAMS error logging and event — tracing ..... log(7)
copy: perform a standalone data — transfer ..... copy(8)
init.d: system level — transition files directory ..... init.d(4)
_toupper, _tolower, toascii: — translate characters ..... conv(3C)

tr: translate characters ..... tr(1)
t_bind: bind an address to a transport endpoint ..... t_bind(3N)
t_close: close a transport endpoint ..... t_close(3N)
look at the current event on a transport endpoint ..... t_look(3N)
t_open: establish a transport endpoint ..... t_open(3N)
t_optmgmt: manage options for a transport endpoint ..... t_optmgmt(3N)
t_unbind: disable a transport endpoint ..... t_unbind(3N)
STREAMS module timod: Transport Interface cooperating ..... timod(7)
interface STREAMS module tirdwr: Transport Interface read/write ..... tirdwr(7)
t_sync: synchronize transport library ..... t_sync(3N)
the scheduler for the uucp file transport program ..... uusched(1M)
uucico: file transport program for the uucp system ..... uucico(1M)
establish a connection with another transport user ..... t_connect(3N)
data sent over a connection t_rcv: receive data or expedited ..... t_rcv(3N)
confirmation from a connect request t_rcvconnect: receive the ..... t_rcvconnect(3N)
disconnect t_rcvdis: retrieve information from ..... t_rcvdis(3N)
orderly release indication t_rcvrel: acknowledge receipt of an ..... t_rcvrel(3N)
t_rcvudata: receive a data unit ..... t_rcvudata(3N)
error indication t_rcvuderr: receive a unit data ..... t_rcvuderr(3N)
ftw: walk a file tree ..... ftw(3C)
tdelete, twalk: manage binary search trees ..... tsearch(3C)
atan, atan2: trigonometric/ trig: sin, cos, tan, asin, acos, ... trig(3M)
cos, tan, asin, acos, atan, atan2: trigonometric functions ..... trig(3M)
true, false: provide truth values ..... true(1)
truncate, ftruncate: truncate a file to a specified length ..... truncate(2SEQ)
to a specified length truncate, ftruncate: truncate a file ..... truncate(2SEQ)
u3b2, u3b5, vax: get processor type truth value ..... machid(1)
true, false: provide truth values ..... true(1)
ts: SCSI tape controller ..... ts(7)
manage binary search trees tsearch, tfind, tdelete, twalk: ..... tsearch(3C)
over a connection t_snd: send data or expedited data ..... t_snd(3N)
disconnect request t_snddis: send user-initiated ..... t_snddis(3N)
release t_sndrel: initiate an orderly ..... t_sndrel(3N)
t_sndudata: send a data unit ..... t_sndudata(3N)
tsort: topological sort ..... tsort(1)
library t_sync: synchronize transport ..... t_sync(3N)
interface tty: controlling terminal (gentty) ..... tty(7)
streamtty: Generic STREAMS tty driver interface ..... streamtty(7)
tty: get the name of the terminal ..... tty(1)
ld0: STREAMS tty line discipline ..... ld0(7)
between major/minor numbers and tty names ..... dev_to_ttyname(3C)
associated with a terminal or the/ ttydevname: get the device ..... ttydevname(1)
terminal ttyname, isatty: find name of a ..... ttyname(3C)
major/minor/ dev_to_ttyname, ttyname_to_dev: convert between ..... dev_to_ttyname(3C)
file of the current user ttyslot: find the slot in the utmp ..... ttyslot(3C)
endpoint t_unbind: disable a transport ..... t_unbind(3N)
parameters vmtune: tune system-wide virtual-memory ..... vmtune(1M)
tunefs: tune up an existing filesystem ..... tunefs(1M)
filesystem tunefs: tune up an existing ..... tunefs(1M)
examine and change virtual memory tuning parameters ..... vm_ctl(2SEQ)
/prdaily, prtacct, shutacct, startup, turnacct: shell procedures for/ ..... acctsh(1M)
tsearch, tfind, tdelete, twalk: manage binary search trees ..... tsearch(3C)

types: primitive system data — types.....types(5)
— types: primitive system data types.....types(5)
lptypes: table of printer — types supported by the menu system.....lptypes(4)
/asctime, cftime, ascftime, mktime, — tzset: convert date and time to/.....ctime(3C)
machid: i386, ns32000, pdp11, — u3b, u3b2, u3b5, vax: get processor type/.....machid(1)
machid: i386, ns32000, pdp11, u3b, — u3b2, u3b5, vax: get processor type/.....machid(1)
/i386, ns32000, pdp11, u3b, u3b2, — u3b5, vax: get processor type truth/.....machid(1)
— uadmin: administrative control.....uadmin(1M)
— uadmin: administrative control.....uadmin(2)
/endspent, fgetspent, lckpwdf, — ulckpwdf: get shadow password file entry.....getspent(3X)
— ulimit: get and set user limits.....ulimit(2)
mask — umask: set and get file creation.....umask(2)
— umask: set file-creation mode mask.....umask(1)
mount, — umount: mount and unmount filesystems.....mount(1M)
— umount: unmount a filesystem.....umount(2)
filesystems mountall, — umountall: mount, unmount multiple.....mountall(1M)
system — uname: get name of current operating.....uname(2)
— uname: print name of current system.....uname(1)
expand data compress, — uncompress, zcat: compress and.....compress(1)
unget: — undo a previous get of an SCCS file.....unget(1)
SCCS file — unget: undo a previous get of an.....unget(1)
input stream — ungetc: push character back into.....ungetc(3S)
file — uniq: report repeated lines in a.....uniq(1)
mktemp: make a — unique filename.....mktemp(3C)
constants — unistd: file header for symbolic.....unistd(4)
t_rcvudata: receive a data — unit.....t_rcvudata(3N)
t_sndudata: send a data — unit.....t_sndudata(3N)
t_rcvuderr: receive a — unit data error indication.....t_rcvuderr(3N)
— units: conversion program.....units(1)
mtio: — UNIX magnetic tape interface.....mtio(7)
cu: call another — UNIX system.....cu(1C)
link, unlink: link and — unlink files and directories.....link(1M)
link, — unlink: link and unlink files and directories.....link(1M)
— unlink: remove directory entry.....unlink(2)
m_lock, m_unlock: lock, — unlock locks.....m_lock(3PPS)
s_clock, s_unlock: initialize, lock, — unlock locks.....s_lock(3PPS)
space munmap: — unmap part of the processs address.....munmap(2SEQ)
spellhist: — unmatched spelling words file.....spellhist(4)
umount: — unmount a filesystem.....umount(2)
mount, umount: mount and — unmount filesystems.....mount(1M)
mountall, umountall: mount, — unmount multiple filesystems.....mountall(1M)
pack, pcat, — unpack: compress and expand files.....pack(1)
set, unset: set and — unset environment variables in core or in files.....set(1F)
variables in core or in files set, — unset: set and unset environment.....set(1F)
lsearch, lfind: linear search and — update.....lsearch(3C)
touch: — update access and modification times of a file.....touch(1)
make: maintain, — update, and regenerate groups of programs.....make(1)
bdflush: — update daemon.....bdflush(1M)
bdflush: — update daemon.....bdflush(2SEQ)
sync: — update superblock.....sync(2)
sync: — update the superblock.....sync(1M)
from/ pwconv: installs and — updates /etc/shadow with information.....pwconv(1M)

isismgr: ISIS file manager, updates text files in ISIS file format ..... isismgr(1M)
been up uptime: show how long system has..... uptime(1)
du: summarize disk usage ..... du(1M)
quota: display disk usage and limits..... quota(1)
retrieve a command description and usage examples..... usage(1)
description and usage examples usage: retrieve a command..... usage(1)
usclk: 32-bit microsecond counter..... usclk(7)
getusclk, usclk_init: get/init microsecond clock ..... getusclk(3SEQ)
get character login name of the user..... cuserid(3S)
logname: return login name of user ..... logname(3X)
setup: initialize system for first user ..... setup(1)
su: become superuser or another user ..... su(1M)
a connection with another transport user ..... t_connect(3N)
slot in the utmp file of the current user..... ttyslot(3C)
write: write to another user ..... write(1)
cpset: copy file and set mode, user, and group IDs..... cpset(1)
setuid, setgid: set user and group IDs..... setuid(2)
id: print user and group IDs and names..... id(1M)
crontab: user crontab file..... crontab(1)
addusr.def: add user defaults..... addusr_def(4)
/geteuid, getgid, getegid: get real user, effective user, real group/..... getuid(2)
environ: user environment ..... environ(5)
run commands performed for multi- user environment..... rc2(1M)
sulog: log of substitute user events ..... sulog(4)
generate disk accounting data by user ID..... diskusg(1M)
ulimit: get and set user limits..... ulimit(2)
tfmenush: assign menu system as user login shell..... tfmenush(1M)
menumsg: route user message to menu message queue..... menumsg(1M)
edquota: edit user quotas..... edquota(1M)
/getegid: get real user, effective user, real group, and effective/..... getuid(2)
t_snddis: send user-initiated disconnect request ..... t_snddis(3N)
editor (variant of ex for casual users)..... edit(1)
the operating system for beginning users..... starter(1)
wall: write to all users ..... wall(1)
ut_delete_user: add/delete users in utmp file..... utadduser(3X)
mail, rmail: send mail to users or read mail..... mail(1)
login attempts /usr/adm/loginlog: log of failed/..... loginlog(4)
ustat: get filesystem statistics..... ustat(2)
add/delete users in utmp file ut_add_user, ut_delete_user: ..... utadduser(3X)
utmp file ut_delete_user: add/delete users in ..... utadduser(3X)
introduction to system standalone utilities..... intro(8)
get_process_stats: get resource utilization information..... get_process_stats(2SEQ)
modification times utime: set file access and ..... utime(2)
utmp, wtmp: utmp and wtmp entry formats..... utmp(4)
ut_delete_user: add/delete users in utmp file..... utadduser(3X)
endutent, utmpname, getutpid: access utmp file entry..... getut(3C)
ttyslot: find the slot in the utmp file of the current user..... ttyslot(3C)
formats utmp, wtmp: utmp and wtmp entry ..... utmp(4)
/pututline, setutent, endutent, utmpname, getutpid: access utmp file/..... getut(3C)
and permissions file uucheck: check the uucp directories..... uucheck(1M)
the uucp system uucico: file transport program for..... uucico(1M)
clean-up uucleanup: uucp spool directory ..... uucleanup(1M)

uucheck: check the — uucp directories and permissions file.....uucheck(1M)
uusched: the scheduler for the — uucp file transport program.....uusched(1M)
— uucp: networking support files.....uucp(4)
uucleanup: — uucp spool directory clean-up.....uucleanup(1M)
uustat: — uucp status inquiry and job control.....uustat(1C)
file transport program for the — uucp system.....uucico(1M)
UNIX-to-UNIX system/ — uucp, uulog, uuname:.....uucp(1C)
speed, and line discipline — uugetty: set terminal type, modes,.....uugetty(1M)
uucp, — uulog, uuname: UNIX-to-UNIX/.....uucp(1C)
uucp, uulog, — uuname: UNIX-to-UNIX system copy.....uucp(1C)
uuto, — uupick: public UNIX-to-UNIX/.....uuto(1C)
file transport program — uusched: the scheduler for the uucp.....uusched(1M)
control — uustat: uucp status inquiry and job.....uustat(1C)
UNIX-to-UNIX system/ — uuto, uupick: public.....uuto(1C)
with debugging on — Uutry: try to contact remote system.....uutry(1M)
system command execution — uux: UNIX-to-UNIX.....uux(1C)
requests — uuxqt: execute remote command.....uuxqt(1M)
— val: validate SCCS file.....val(1)
val: — validate SCCS file.....val(1)
— valloc: aligned memory allocator.....valloc(3SEQ)
abs: return integer absolute — value.....abs(3C)
u3b5, vax: get processor type truth — value.....machid(1)
getenv: return — value for environment name.....getenv(3C)
floor, ceiling, remainder, absolute — value functions.....floor(3M)
readlink: read — value of a symbolic link.....readlink(2SEQ)
putenv: change or add — value to environment.....putenv(3C)
current form field to its default — values.....reset(1F)
true, false: provide truth — values.....true(1)
values: machine-dependent — values.....values(5)
— values: machine-dependent values.....values(5)
print formatted output of a — varargs argument list.....vprintf(3S)
list — varargs: handle variable argument.....varargs(5)
varargs: handle — variable argument list.....varargs(5)
fpathconf: get configurable pathname — variables.....pathconf(2)
sysconf: get configurable system — variables.....sysconf(2)
.environ, .pref, — .variables: FACE files.....env(4)
unset: set and unset environment — variables in core or in files.....set(1F)
edit: text editor (— variant of ex for casual users).....edit(1)
proc_ctl: control over — various process attributes.....proc_ctl(2SEQ)
/ns32000, pdp11, u3b, u3b2, u3b5, — vax: get processor type truth value.....machid(1)
— vc: version control.....vc(1)
get option letter from argument — vector.....getopt(3C)
readv: read — vector from file.....readv(2SEQ)
writev: write — vector on a file.....writev(2SEQ)
assert: — verify program assertion.....assert(3X)
vc: — version control.....vc(1)
get: get a — version of an SCCS file.....get(1)
sccsdiff: compare two — versions of an SCCS file.....sccsdiff(1)
virtual memory efficient way — vfork: spawn new process in a.....vfork(2)
output of a varargs/ vprintf, — vfprintf, vsprintf: print formatted.....vprintf(3S)
editor based on ex — vi: screen-oriented (visual) display.....vi(1)
more: — view a file one full screen at a time.....more(1)

map physical addresses into processs virtual address space.....pmap(7)
devtab: virtual device table.....devtab(4)
devdestroy: build and destroy virtual devices.....devbuild(1M)
vfork: spawn new process in a virtual memory efficient way.....vfork(2)
vm_ctl: examine and change virtual memory tuning parameters.....vm_ctl(2SEQ)
echo: put string on virtual output.....echo(1F)
controlling terminal fvhangup: virtually hangup the specified.....fvhangup(2SEQ)
vmtune: tune system-wide virtual-memory parameters.....vmtune(1M)
pmap: display or alter parameters of virtual-to-physical mapping devices.....pmap(1M)
vi: screen-oriented visual display editor based on ex.....vi(1)
memory tuning parameters vm_ctl: examine and change virtual.....vm_ctl(2SEQ)
virtual-memory parameters vmtune: tune system-wide.....vmtune(1M)
filesystem volcopy: make literal copy of.....volcopy(1M)
fs, inode: format of file system volume.....fs(4)
formatted output of a varargs/ vprintf, vfprintf, vsprintf: print.....vprintf(3S)
the controlling FMLI object vsig: synchronize a co-process with.....vsig(1F)
a varargs/ vprintf, vfprintf, vsprintf: print formatted output of.....vprintf(3S)
prtvtoc: print the VTOC of a block device.....prtvtoc(1M)
prtvtoc: print the VTOC of a standalone disk device.....prtvtoc(8)
mkvtoc: populate VTOC on disks.....mkvtoc(1M)
vtoc: VTOC partitioning driver.....vtoc(7)
vtoc: VTOC partitioning driver.....vtoc(7)
s_wait_barrier: initialize barrier, wait at barrier.....s_wait_barrier(3PPS)
wait: await completion of process.....wait(1)
wait, waitpid: wait for a child process to stop or terminate.....wait(2)
sigsuspend: wait for a signal.....sigsuspend(2)
process to stop or terminate wait, waitpid: wait for a child.....wait(2)
stop or terminate wait, waitpid: wait for a child process to.....wait(2)
ftw: walk a file tree.....ftw(3C)
wall: write to all users.....wall(1)
wc: word count.....wc(1)
wd: SSM SCSI disk controller.....wd(7)
what: identify SCCS files.....what(1)
whatis: describe what a command is.....whatis(1)
whodo: who is doing what.....whodo(1M)
who: who is on the system.....who(1)
who: who is on the system.....who(1)
whodo: who is doing what.....whodo(1M)
vmtune: tune system- wide virtual-memory parameters.....vmtune(1M)
addformopt: add a form/SELECT window option to a menu.....addformopt(1M)
wc: word count.....wc(1)
fgetc, getw: get character or word from a stream.....getc(3S)
fputc, putw: put character or word on a stream.....putc(3S)
spellhist: unmatched spelling words file.....spellhist(4)
cd: change working directory.....cd(1)
chdir: change working directory.....chdir(2)
getcwd: get pathname of current working directory.....getcwd(3C)
pwd: working directory name.....pwd(1)
specific alarms and/or the working indicator.....indicator(1F)
interface wp: SSM parallel line printer.....wp(7)
lseek: move read/ write file pointer.....lseek(2)
tirdwr: Transport Interface read/ write interface STREAMS module.....tirdwr(7)

write: write on a file ..... write(2)
putpwent: write password file entry ..... putpwent(3C)
putspent: write shadow password file entry ..... putspent(3X)
wall: write to all users ..... wall(1)
write: write to another user ..... write(1)
writev: write vector on a file ..... writev(2SEQ)
write: write on a file ..... write(2)
write: write to another user ..... write(1)
writev: write vector on a file ..... writev(2SEQ)
open: open for reading or writing ..... open(2)
utmp, wtmp: utmp and wtmp entry formats ..... utmp(4)
utmp, wtmp: utmp and wtmp entry formats ..... utmp(4)
accounting records fwtmp, wtmpfix: manipulate connect ..... fwtmp(1M)
and execute command xargs: construct argument list(s) ..... xargs(1)
multiplexor driver xs: Xylogics 781 terminal ..... xs(7)
tape interface xt: Xylogics 472 MULITBUS magnetic ..... xt(7)
xt: Xylogics 472 MULITBUS magnetic tape interface ..... xt(7)
xs: Xylogics 781 terminal multiplexor driver ..... xs(7)
bessel: j0, j1, jn, y0, y1, yn: Bessel functions ..... bessel(3M)
bessel: j0, j1, jn, y0, y1, yn: Bessel functions ..... bessel(3M)
yacc: yet another compiler-compiler ..... yacc(1)
compress, uncompress, zcat: compress and expand data ..... compress(1)
zd: dual-channel disk controller ..... zd(7)
for dual-channel disk controller zdformat: standalone disk formatter ..... zdformat(8)
getzdinfo, getzdinfobyname: get zdinfo structure ..... getzdinfo(3SEQ)

NAME

intro – introduction to maintenance commands and application programs

DESCRIPTION

This section describes, in alphabetical order, commands used for programming, system maintenance and administration, and general use. Certain distinctions of purpose are made in the headings:

(1) Commands of general utility

(1C) Commands for communication with other systems

(1F) Commands for use with FMLI (Forms and Menu Language Interpreter)

(1G) Commands for graphics and computer-aided design

(1M) Commands for system maintenance and administration

COMMAND SYNTAX

Unless otherwise noted, commands described in this section accept options and other arguments according to the following syntax:

name [*option*(*s*)] [*cmdarg*(*s*)]

The elements of the syntax take these values:

name	The name of an executable file.
option	– *noargletter*(*s*) or – *argletter* <>*optarg* where <> is optional white space.
noargletter	A single letter representing an option without an argument.
argletter	A single letter representing an option requiring an argument.
optarg	Argument (character string) satisfying preceding *argletter*.
cmdarg	Pathname (or other command argument) *not* beginning with –, or – by itself indicating the standard input.

Command Syntax Standard: Rules

These command syntax rules are not followed by all current commands, but all new commands will obey them. **getopts**(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 lowercase 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 preceded 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

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

DIAGNOSTICS

On 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 system byte is 0 for normal termination; the program byte is customarily zero for successful execution and non-zero to indicate troubles such as erroneous parameters, bad or inaccessible data, or other inability to cope with the task at hand. The return status 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.

BUGS

Not all commands adhere to the syntax standard.

NAME

300, 300s – handle special functions of DASI 300 and 300s terminals

SYNOPSIS

300 [**+12**] [**–n**] [**–dt,l,c**]

300s [**+12**] [**–n**] [**–dt,l,c**]

DESCRIPTION

The **300** command supports special functions and optimizes the use of the DASI 300 (GSI 300 or DTC 300) terminal; **300s** performs the same functions for the DASI 300s (GSI 300s or DTC 300s) terminal. It converts half-line forward, half-line reverse, and full-line reverse motions to the correct vertical motions. It also attempts to draw Greek letters and other special symbols. It permits convenient use of 12-pitch text. It also reduces printing time 5 to 70%. The **300** command can be used to print equations neatly. Use this sequence:

neqn *file* ... | **nroff** | **300**

WARNING: if your terminal has a PLOT switch, make sure it is turned on before **300** is used.

The behavior of **300** can be modified by the optional flag arguments to handle 12-pitch text, fractional line spacings, messages, and delays.

+12 Permits use of 12-pitch, 6 lines/inch text. DASI 300 terminals normally allow only two combinations: 10-pitch, 6 lines/inch, or 12-pitch, 8 lines/inch. To obtain the 12-pitch, 6 lines per inch combination, the user should turn the PITCH switch to 12, and use the **+12** option.

–n Controls the size of half-line spacing. A half-line is, by default, equal to 4 vertical plot increments. Because each increment equals 1/48 of an inch, a 10-pitch line-feed requires 8 increments, while a 12-pitch line-feed needs only 6. The first digit of *n* overrides the default value, thus allowing for individual taste in the appearance of subscripts and superscripts. For example, **nroff** half-lines could be made to act as quarter-lines by using **–2**. The user could also obtain appropriate half-lines for 12-pitch, 8 lines/inch mode by using the option **–3** alone, having set the PITCH switch to 12-pitch.

–d *t,l,c* Controls delay factors. The default setting is **–d3,90,30**. DASI 300 terminals sometimes produce peculiar output when faced with very long lines, too many tab characters, or long strings of blankless, non-identical characters. One null (delay) character is inserted in a line for every set of *t* tabs, and for every contiguous string of *c* non-blank, non-tab characters. If a line is longer than *l* bytes, 1+(total length)/20 nulls are inserted at the end of that line. Items can be omitted from the end of the list, implying use of the default values. Also, a value of zero for *t* (*c*) results in two null bytes per tab (character). The former may be needed for C programs, the latter for files like */etc/passwd*. Because terminal behavior varies according to the specific characters printed and the load on a system, the user may have to experiment with these values to get correct output. The **–d** option exists only as a last resort for those few cases that do not otherwise print properly. For example, the file */etc/passwd* may

be printed using **–d3,30,5**. The value **–d0,1** is a good one to use for C programs that have many levels of indentation.

Note that the delay control interacts heavily with the prevailing carriage return and line-feed delays. The **stty**(1) modes **nl0 cr2** or **nl0 cr3** are recommended for most uses.

The **300** command can be used with the **nroff –s** flag or **.rd** requests when it is necessary to insert paper manually or change fonts in the middle of a document. Instead of hitting the return key in these cases, you must use the line-feed key to get any response.

In many (but not all) cases, the following sequences are equivalent:

nroff –T300 *files* ... and **nroff** *files* ... | **300**
nroff –T300–12*files* ... and **nroff** *files* ... | **300 +12**

The use of **300** can thus often be avoided unless special delays or options are required. In a few cases, however, the additional movement optimization of **300** may produce better-aligned output.

SEE ALSO

450(1), mesg(1), graph(1G), stty(1), tabs(1), tplot(1G)

eqn(1), nroff(1), tbl(1) in the *DOCUMENTER'S WORKBENCH Technical Discussion and Reference Manual*

BUGS

Some special characters cannot print correctly in column 1 because the printhead cannot be moved to the left from there.

If your output contains Greek and/or reverse line-feeds, use a friction-feed platen instead of a forms tractor. Although good enough for drafts, the latter has a tendency to slip when reversing direction, distorting Greek characters and misaligning the first line of text after one or more reverse line-feeds.

NAME

4014 – paginator for the Tektronix 4014 terminal

SYNOPSIS

4014 [**–t**] [**–n**] [**–c***N*] [**–p**L] [file]

DESCRIPTION

The output of **4014** is intended for a Tektronix 4014 terminal; **4014** arranges for 66 lines to fit on the screen, divides the screen into *N* columns, and contributes an eight-space page offset in the (default) single-column case. Tabs, spaces, and backspaces are collected and plotted when necessary. Teletype Model 37 half- and reverse-line sequences are interpreted and plotted. At the end of each page, **4014** waits for a new-line (empty line) from the keyboard before continuing on to the next page. In this wait state, the command **!***cmd* will send the *cmd* to the shell.

The command line options are as follows:

–t Do not wait between pages (useful for directing output into a file).

–n Start printing at the current cursor position and never erase the screen.

–c*N* Divide the screen into *N* columns and wait after the last column.

–p*L* Set page length to *L*; *L* accepts the scale factors **i** (inches) and **l** (lines); default is lines.

SEE ALSO

pr(1)

NAME

450 – handle special functions of the DASI 450 terminal

SYNOPSIS

450

DESCRIPTION

The **450** command supports the special functions and optimizes the use of the DASI 450 terminal or any terminal that is functionally identical, such as the Diablo 1620 or Xerox 1700. It converts half-line forward, half-line reverse, and full-line reverse motions to the correct vertical motions. It also attempts to draw Greek letters and other special symbols in the same manner as **300**(1). Use **450** to print equations neatly. Use this sequence:

neqn *file* ... | **nroff** | **450**

WARNING: make sure that the PLOT switch on your terminal is ON before **450** is used. The SPACING switch should be put in the desired position (either 10- or 12-pitch). In either case, vertical spacing is 6 lines/inch, unless dynamically changed to 8 lines per inch by an appropriate escape sequence.

Use **450** with the **nroff –s** flag or **.rd** requests when it is necessary to insert paper manually or change fonts in the middle of a document. Instead of hitting the return key in these cases, you must use the line-feed key to get any response.

In many (but not all) cases, the use of **450** can be eliminated in favor of one of the following:

nroff –T450 *files* ...

or

nroff –T450–12 *files* ...

The use of **450** can thus often be avoided unless special delays or options are required. In a few cases, however, the additional movement optimization of **450** may produce better-aligned output.

SEE ALSO

300(1), mesg(1), stty(1), tabs(1), graph(1G), tplot(1G)

eqn(1), nroff(1), tbl(1) in the *DOCUMENTER'S WORKBENCH Technical Discussion and Reference Manual*

BUGS

Some special characters cannot print correctly in column 1 because the printhead cannot be moved to the left from there.

If your output contains Greek and/or reverse line-feeds, use a friction-feed platen instead of a forms tractor. Although good enough for drafts, the latter has a tendency to slip when reversing direction, distorting Greek characters and misaligning the first line of text after one or more reverse line-feeds.

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**.

–r[*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

enable(1), lp(1), lpstat(1), lpadmin(1M), lpsched(1M)

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 **acctcon**(1M) convert this file into session and charging records, which are then summarized by **acctmerg**(1M).

Process accounting is performed by the operating 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; **acctcms**(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)). **prtacct** (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:

```
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
/usr/adm/pacct	Current process accounting file
/etc/wtmp	Login/logoff history file

SEE ALSO

acctcom(1), acctcms(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), diskusg(1M), fwtmp(1M), runacct(1M), acct(2), acct(4), 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 as follows:

–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. **–t** combines the prime and non-prime time parts into a single field that is the total of both and provides upward compatibility with old 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.

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

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

SEE ALSO

acctcom(1), acct(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M), acct(2), acct(4), 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 files

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 pound sign (#) 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 question mark (?) 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 as follows:

–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 expression 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), acctcms(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), ps(1), runacct(1M), su(1), acct(2), acct(4), 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.

The memory size of a process is not maintained by the system and is reported as 0.

NAME

acctcon1, acctcon2 – connect-time accounting

SYNOPSIS

/usr/lib/acct/acctcon1 [options]

/usr/lib/acct/acctcon2

DESCRIPTION

acctcon1 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 as follows:

–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 line 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 to four 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 `tacct` 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 +1n +2 > ctmp
acctcon2 <ctmp | acctmerg > ctacct
```

FILES

/etc/wtmp

SEE ALSO

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

–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:

acctmerg –v <file1 > file2

Edit *file2* as desired...

acctmerg –i <file2 > file1

SEE ALSO

acctcom(1), acct(1M), acctcms(1M), acctcon(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M), acct(2), acct(4), 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

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

NOTES

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.

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 **acctwtmp** program in **acct**(1M).

The mean memory size for a process is not maintained by the system and is reported as 0.

NAME

chargefee, ckpacct, dodisk, lastlogin, monacct, nulladm, prctmp, prdaily, prtacct, 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] [mmdd]

/usr/lib/acct/prtacct file ["heading"]

/usr/lib/acct/runacct [mmdd] [mmdd 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* filesystem 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*. Disk accounting is limited to filesystems of type "4.2". If the **–o** flag is used, it will do a slower version of disk accounting by login directory. *files* specifies 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 mountpoints of mounted filesystems. If omitted, they should be the special filenames of mountable filesystems.

lastlogin is invoked by **runacct** to update */usr/adm/acct/sum/loginlog*, which shows the last date 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 be 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/nite/ctmp* created by **acctcon**(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 mmdd* where *mmdd* 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 *mmdd* option and specifying the exact report date desired. The **–l** flag prints a report of exceptional usage by login ID for the specifed 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/wtmp*.

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 **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

acctcom(1), acct(1M), acctcms(1M), acctcon(1M), acctmerg(1M), acctprc(1M),

cron(1M), diskusg(1M), fwtmp(1M), runacct(1M), acct(2), acct(4), utmp(4)

NAME

addcmdopt – add a command option to a menu

SYNOPSIS

addcmdopt –M *menu-name* **–t** *task* **–d** *desc* [**–S SHELL | NOSHELL**] **–C CLEAR | NOCLEAR –H HOLD | NOHOLD** [**–g** *grps*] [**–u** *users*] [**–p**] **–c** *command* [**–h** *helpfile*] [**–k** *keywords*] [**–F**]

DESCRIPTION

This command adds a command option to a menu on the ptx/ADMIN menu system. This option calls any command that can normally run from the operating system shell.

–M *parentname* The name of an existing menu that will display the new option. For example, **clusr**.

–t *taskname* The name of task that option calls. You can select an existing task or enter a unique name and the system will create a new task for you.

–d *taskdesc* Description of the new task, for example, Display Outstanding Calls. The menu system displays this description on the menu (*parentmenu*) for the command option.

–S *shellflag* The *shellflag* controls how the command is executed. Use **SHELL** when the menu system must spawn the Bourne shell to execute the command. Use the **NOSHELL** value when the menu system executes the command without invoking the shell.

–C *clearflag* The *clearflag* controls how the menu is cleared from the screen when the user selects the command option. Use the **CLEAR** value when the menu system clears the menu from the screen before calling the command. Use the **NOCLEAR** value when the command clears the screen.

–H *holdflag* The *holdflag* controls how the menu is redisplayed after the user exits the command. Use the **NOHOLD** value to redisplay the menu when the user exists the command. Use the **HOLD** value when you want the user to press the Return key to redisplay the menu.

–p Use this option to make the command available to all users.

–g *grplist* Use this option to limit the groups that can use the command.

–u *usrlist* Use this option to limit the users that can use the command.

–c *cmdstring* Use this option to name the program that the command option will run. Enter the full pathname of the command and include any arguments, for example, **/usr/bin/sales_report –d 02/11/89** .

–h *helpfile* This option identifies the file that contains the new command's helpfile. Must be the full pathname of the file. The system copies this file and it becomes the help text displayed for the command.

-k *keywords* This option identifies the keywords for the new command. The keywords must appear in the option's helpfile and source file.

-F Use this option to overwrite existing files and tasks for the command. Primarily used for updates.

Environment Variables

TFADMFILE Identifies the directory where the ptx/ADMIN administrative data files are located.

LANG Identifies the language used for internationalized versions of the product. Will default to *english_usa*.

FILES

${TFADMFILE}/etc/task	Task table
${TFADMFILE}/menus/chap1/*parentmenu*	Menu displaying the option
${TFADMFILE}/doc/{LANG}/add_ons	Directory for help text

SEE ALSO

addmenuopt(1M), addformopt(1M), rmvopt(1M)

NAME

addformopt – add a form/SELECT window option to a menu

SYNOPSIS

addformopt –M *menu-name* **–t** *task-name* **–d** *desc* **–f** *form/SELECT-window* [**–C CLEAR** | **NOCLEAR**] [**–H HOLD** | **NOHOLD**] [**–g** *grps*] [**–u** *users*] [**–p**] [**–h** *helpfile*] [**–k** *keywords*] [**–F**]

DESCRIPTION

This command adds a form or SELECT window option to a menu on the ptx/ADMIN menu system.

–M *parentmenu* The name of an existing menu that displays the option. For example, **c1main**.

–t *taskname* The name of task that the option calls. You can select an existing task, or enter a unique name and the system will create a new task for you.

–d *taskdesc* Description of the new command, such as Display Outstanding Calls. The menu system displays this description on the menu (*parentmenu*) for the form/SELECT window option.

–f *filename* The full pathname of the file that contains the new command.

–C *clearflag* The *clearflag* controls how the menu is cleared from the screen when the user selects the command option. Use the **CLEAR** value when the menu system clears the menu from the screen before calling the command. Use the **NOCLEAR** value when the command clears the screen.

–H *holdflag* The *holdflag* controls how the menu is redisplayed after the user exits the command. Use the **NOHOLD** value to redisplay the menu when the user exists the command. Use the **HOLD** value when you want the user to press Return to redisplay the menu.

–p Use this option to make the command available to all users.

–g *grplist* Use this option to limit the groups that can use the command.

–u *usrlist* Use this option to limit the users that can use the command.

–h *helpfile* This option identifies the file that contains the new command's helpfile. Must be the full pathname of the file. The system copies this file and it becomes the help text displayed for the command.

–k *keywords* This option identifies the keywords for the new command. The keywords must appear in the option's helpfile and source file.

–F Use the this option to overwrite existing files and tasks for the command. Primarily used for updates.

Environment Variables

TFADMFILE Identifies the directory where the ptx/ADMIN administrative data files are located.

LANG Identifies the language used for internationalized versions of the product. Will default to *english_usa*.

FILES

${TFADMFILE}/etc/task	Task table
${TFADMFILE}/menus/chap1/*parentmenu*	Menu displaying the option
${TFADMFILE}/edc/add_ons	Directory for compiled file for form/SELECT window (see **ec**(1M))

SEE ALSO

addmenuopt(1M), addcmdopt(1M), rmvopt(1M)

NAME

addmenuopt – add a menu option to a menu

SYNOPSIS

addmenuopt –M *menu-name* **–m** *new-menu-name* **–d** *desc* [**–g** *grps*] [**–u** *users*] [**–p**] [**–f** *menufile*] [**–h** *helpfile*] [**–k** *keywords*] [**–F**]

DESCRIPTION

This command adds a menu option to the ptx/ADMIN menu system.

–M *parentmenu* The name of an existing menu that will display the menu as an option. For example, **clusr**

–m *menuname* The name of the menu that you are adding to the menu system. For example, **clprint**.

–d *menudesc* Description of the new menu, such as Printing Files. The menu system displays this description on the parent menu for the new menu.

–p Use this option to make the menu available to all users.

–g *grplist* Use this option to limit the groups that can use the menu.

–u *usrlist* Use this option to limit the users who can use the menu.

–f *filename* The full pathname of the file containing the new menu.

–h *helpfile* This option identifies the file containing the new menu's helpfile. You must enter the full pathname of the file. The system copies this file and it becomes the help text displayed for the command.

–k *keywords* This option identifies the keywords for the new menu. The keywords must appear in the menus's helpfile and source file.

–F Use the –F option to overwrite existing files and tasks for the menu. Primarily used for updates.

Environment Variables

TFADMFILE Identifies the directory where the ptx/ADMIN administrative data files are located.

FILES

${TFADMFILE}/etc/task
Task table

${TFADMFILE}/menus/chap1/*parentmenu*
Menu that displays the menu

{TFADMFILE}/menus/add_ons/parentmenu/menuname
Copied or newly created menu file

SEE ALSO

addformopt(1M), addcmdopt(1M), rmvopt(1M)

NAME

admin – create and administer SCCS files

SYNOPSIS

admin [**–n**] [**–i**[name]] [**–r**rel] [**–t**[name]] [**–f**flag[flag-val]] [**–d**flag[flag-val]] [**–a**login] [**–e**login] [**–m**[mrlist]] [**–y**[comment]] [**–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 filenames must begin with the characters *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 pathname 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 filename 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 *rel*ease 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 are also used), the descriptive text filename must also be supplied. In the case of existing SCCS files: (1) a **–t** keyletter without a filename causes removal of descriptive text (if any) currently in the SCCS file, and (2) a **–t** keyletter with a filename 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 as follows:

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

c*ceil* 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.

f*floor* 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.

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

i*[str]* Causes the `No ID keywords (ge6)` 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 match the given string exactly. 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.

l*list* 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 (for example, 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.

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

m*mod* *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.

t*type* *Type* of module in the SCCS file substituted for all occurrences of %Y% keyword in SCCS file text retrieved by **get**(1).

v*pgm* 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.

l*list* A *list* of releases to be "unlocked". See the **–f** keyletter for a description of the **l** flag and the syntax of a *list*.

–a*login* A *login* name, or numerical 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 *login*s, 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 this form:

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 (that is, 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 checksum (the sum of all characters in the SCCS file except those in the first line) with the checksum 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 checksum 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 filenames must be of the form *s.filename*. 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.filename*, [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 checksum. Another **admin –h** is recommended to ensure the SCCS file is valid.

admin also makes use of a transient lock file (called *z.filename*), 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

apropos – locate commands by keyword lookup

SYNOPSIS

apropos keyword ...

DESCRIPTION

apropos shows which *Reference Manual* sections contain instances of any of the given keywords in their title. Each word is considered separately and case is ignored. Words which are part of other words are considered; for example looking for *compile* also finds all instances of *compiler*. Try

apropos password

and

apropos editor

If the line starts `name(section) ...`, enter **man section name** to get the documentation for it. Enter **apropos format** and then **man 3s printf** to get the manual page on the subroutine **printf.**

apropos is equivalent to the **man –k** command.

FILES

/usr/catman/whatis data base

SEE ALSO

man(1), whatis(1), makewhatis(1M)

NAME

asa – interpret ASA carriage control characters

SYNOPSIS

asa [files]

DESCRIPTION

asa interprets the output of FORTRAN programs that utilize ASA carriage control characters. It processes either the *files* whose names are given as arguments or the standard input if no file names are supplied. The first character of each line is assumed to be a control character with the following meanings:

' '	(blank) single new line before printing
0	double new line before printing
1	new page before printing
+	overprint previous line.

Lines beginning with other than the above characters are treated as if they began with ' '. The first character of a line is not printed. If any such lines appear, an appropriate diagnostic appears on standard error. This program forces the first line of each input file to start on a new page.

To view correctly the output of FORTRAN programs which use ASA carriage control characters, **asa** can be used as a filter as follows:

```
a.out | asa | lp
```

In this example, the output, properly formatted and paginated, is directed to the line printer. FORTRAN output sent to a file could be viewed by:

```
asa file
```

SEE ALSO

efl(1), f77(1), fsplit(1), ratfor(1)

NAME

assist – assistance using operating system commands

SYNOPSIS

assist [name]
assist [**–s**]
assist [**–c** name]

DESCRIPTION

The **assist** command invokes the ASSIST menu interface software for the operating system. The ASSIST menus categorize operating system commands in a hierarchy according to function. The menus lead to full-screen forms (called command forms) that aid in the execution of a syntactically correct command line. The menus also lead to interactive simulations of operating system commands or concepts (called walkthrus).

If you type **assist** without options, you enter at the top of the menu interface hierarchy. New users may wish to use the **–s** option to select an introductory tutorial explaining how to use the ASSIST software. Options are as follows:

name	Invoke an ASSIST-supported operating system command form or walkthru for *name*
–c *name*	Invoke the version of *name* that is in your current directory
–s	Reinvoke the ASSIST setup module to check or modify your terminal variable or to access the introductory information about ASSIST

When you invoke **assist**, you perform operations within the program by using **assist** commands. To see a list of the **assist** commands, type Ctrl-A or **f8** (function key 8) when you are in **assist**. A list of commands then prints on the terminal screen.

EXAMPLES

This example illustrates how to invoke a particular command form directly. In this case, **mkdir** is the desired command form.

```
assist mkdir
```

FILES

$HOME/.assistrc	Information needed by **assist** (for example, about the terminal you are using)
/usr/lib/assist	Default directory containing assist command forms, walkthrus, and executable programs
/usr/lib/readme/assist	Basic information about installing **assist** and **astgen**

SEE ALSO

astgen(1)

NOTES

The first time you invoke **assist**, it will ignore any options you give and ask for information about the terminal you are using. Once it has saved this information in a file named *.assistrc* in your home directory, it will show you a list of basic **assist** commands and offer you an introduction to ASSIST.

NAME

astgen – program for generating/modifying ASSIST menus or command forms

SYNOPSIS

astgen name[.fs]

DESCRIPTION

astgen is an interactive program to generate information files (ASCII text data files) that define a menu or command form used by the **assist**(1) program.

Both the **astgen** and **assist**(1) programs recognize and process information files whose names are suffixed with three characters: *.fs* . If no *.fs* file exists for the specified name, **astgen** assumes that a new menu or command form is to be created. If *name* is given without *.fs*, **astgen** will automatically create the file *name.fs*.

SEE ALSO

assist(1)

NAME

at, batch – execute commands at a later time

SYNOPSIS

at time [date] [+ increment]
at –r job ...
at –l [job ...]
at –q [queuename ...]

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** 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.

–q Allows an alternate queue to be used.

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 remove only your own jobs unless you are the superuser.

at –q allows alternate queues to be used. A letter immediately following the letter q specifies the prototype file to be used. Normally the file **/usr/lib/cron/.proto** is used as a template to set up a shell environment. **batch** uses the file **.proto.b** if it exists. A system administrator can set up other files such that invoking **at** with the argument **-qc** will use the file **.proto.c**. Additional information is given by **proto(4)**.

batch submits a batch job. It is almost equivalent to **at now**, except that it goes into a different queue, and **at now** responds with the error message `too late`.

at and **batch** write the job number and schedule time to standard error. 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 can use **at** if their name appears in the */usr/lib/cron/at.allow* file. If that file does not exist, the */usr/lib/cron/at.deny* file 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 be modified only 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 digit numbers 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 24-hour clock time is understood. The suffix **zulu** may be used to indicate UTC (Universal Time Code, formerly Greenwich Mean Time). 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.

The following commands are all legitimate:

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

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*
Ctrl-D (hold down Control and depress D)

The following 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/atjobs	Spool area

SEE ALSO

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

DIAGNOSTICS

Complains about various syntax errors and times out of range.

NAME

awk – pattern scanning and processing language

SYNOPSIS

awk [**–Fc**] [prog] [parameters] [files]

DESCRIPTION

awk 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 **awk.**

Files are read in order; if there are no files, the standard input is read. The filename – 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 this 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**(3S)].

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**(3S) 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

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 the following line or by using the **-F***c* option.

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

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 %.6g).

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: **awk –f program n=5 input**

SEE ALSO

grep(1), lex(1), nawk(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 an expression to be treated as a string, concatenate the null string ("") to it.

NAME

banner – make posters

SYNOPSIS

banner strings

DESCRIPTION

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

SEE ALSO

echo(1)

NAME

basename, dirname – deliver portions of pathnames

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 pathname 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 next example sets 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 C-like language that provides unlimited precision arithmetic. It takes input from files, and then reads **stdin**. **bc** is a preprocessor for **dc**(1), which it invokes automatically unless the **–c** option is specified. In this case, the **dc** input is sent to the **stdout**. The options are:

–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 –l math library:

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

All function arguments are passed by value.

The value of an expression is printed unless the main operator is an assignment. Either semicolons or newlines can 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 can be used simultaneously as an array, a function, and a simple variable. All variables are global to the program. "Auto" variables are pushed down during function calls. Used as function arguments or automatic variables, array names must be followed by empty square brackets.

EXAMPLES

This example defines a function to compute an approximate value of the exponential function:

```
scale = 20
define e(x){
        auto a, b, c, i, s
        a = 1
        b = 1
        s = 1
        for(i=1; 1==1; i++){
                a = a*x
                b = b*i
                c = a/b
                if(c == 0) return(s)
                s = s+c
        }
}
```

The next example prints approximate values of the exponential function of the first 10 integers:

```
for(i=1; i<=10; i++) e(i)
```

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

bcheckrc – system initialization procedures

SYNOPSIS

/etc/bcheckrc

DESCRIPTION

This shell procedure is executed via an entry in */etc/inittab* by **init**(1M) whenever the system is booted (or rebooted).

First, **bcheckrc** uses **fsstat** to check the status of the root filesystem. If the root filesystem is found to be bad, **bcheckrc** repairs it by running **fsck**. If **fsck** fails, an interative shell is issued to allow manual intervention. When the shell exits, the system is automatically rebooted to restore the operating system view of the corrected root filesystem.

Then, **bcheckrc** clears the mounted filesystem table (*/etc/mnttab*) and puts the entry for the root filesystem into the mount table using **setmnt**.

After this procedure completes, **init** continues to process other *sysinit* level procedures in */etc/inittab*. When all *sysinit* procedures are complete, **init** places the system in its initial default state. Since *initdefault* is initially set to 2, the system is placed in the multiuser state via the **/etc/rc2** procedure.

Note that **bcheckrc** should always be executed before any other *sysinit* procedure to insure a clean root filesystem before doing any writes.

SEE ALSO

sh(1), fsck(1M), fsstat (1M), init(1M), rc2(1M), setmnt (1M), shutdown(1M)

NAME

bdflush – update daemon

SYNOPSIS

/etc/bdflush [autoup [bdflushrate]]

DESCRIPTION

bdflush is the update daemon which periodically flushes modified disk buffers to disk. *autoup* is the maximum age of delayed write buffers before they get written out. *bdflushrate* is the rate at which bdflush checks for delayed write buffers.

bdflush should normally be run in the background since the system call it makes never returns. To change the rate at which the system flushes disk buffers, it is sufficient to kill the current invocation of **bdflush** and restart it with different arguments.

SEE ALSO

bdflush(2SEQ)

NAME

bdiff – big diff

SYNOPSIS

bdiff file1 file2 [n] [**–s**]

DESCRIPTION

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

The parameters to **bdiff** are as follows:

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 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 newline) per line (255 for 16-bit machines). **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 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 entering another **P** and carriage return. 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 prints only the name of the file being scanned; there is no *remembered* filename. 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 these 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 following command is an unconditional jump:

xb/^/ *label*

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**, all of the following commands will print the first 100 lines.

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

The following command would globally search for the characters **100** and print each line containing a match.

```
g/%5/p
```

To escape the special meaning of %, a \ must precede it. The following command could be used to match and list lines containing **printf** of characters, decimal integers, or strings.

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

Another feature of the **xv** command is that the first line of output from an operating system command can be stored in a variable. The only requirement is that the first character of *value* be an **!**. For example, the following commands would put the current line into variable **5**, print it, and increment the variable **6** by one.

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

To escape the special meaning of ! as the first character of *value*, precede it with a \. The following command stores the value **!date** in variable **7**.

```
xv7\!date
```

xbz *label*
xbn *label*

These two commands will test the last saved return code from the execution of an operating system command (**!***command*) or non-zero value, respectively, to the specified label. The two examples below both search for the next five lines containing the string **size**.

```
xv55
:l
/size/
xv5!expr %5 - 1
!if 0%5 != 0 exit 2
xbn l
xv45
:l
/size/
xv4!expr %4 - 1
!if 0%4 = 0 exit 2
xbz l
```

xc [*switch*]

If *switch* is **1**, output from the **p** and null commands is crunched; if *switch* is **0** the output is not crunched. Without an argument, **xc** reverses *switch*. Initially, *switch* is set for no crunching. Crunched output reduces strings of tabs and blanks to one blank and suppresses blank lines.

SEE ALSO

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

DIAGNOSTICS

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

NAME

bomverify – perform an installation consistency check based on the BOM

SYNOPSIS

/etc/bomverify [**–l**|**v**] product [product product...]

DESCRIPTION

The **bomverify** program compares the Bill of Materials (BOM) of the specified product(s) to the actual status of each file listed in the BOM as it currently exists on the system. By default, it attempts to change the status of inconsistent files to match the BOM entry for that file. The information listed in the BOM includes the mode, owner, and group, size and checksum, and hard and symbolic link data. **bomverify** attempts to correct mode, owner, and group status, and also links when possible. Correction attempts which fail are reported, as well as files which don't match in size or checksum, since these can't be corrected.

The consistency check should be done while the system is inactive or in single user mode. If the */etc/group* or */etc/passwd* files are changed during the check, false errors are possible.

product indicates the application to be verified, and must correspond to a directory in */usr/options*. This directory contains a subdirectory for each distinct product installed on the system, and each subdirectory contains a Bill of Materials for that product. For example, to verify the base installation, you would specify *base* as the product, which corresponds to the directory */usr/options/base*.

The **–l** option specifies *listonly*. When this option is used, no attempts are made to correct inconsistent files. Inconsistencies are merely reported, with the expected and actual file status.

The **–v** option specifies *verbose* mode. In this mode, all changes made to the file system to bring it into line with the BOM are reported. Otherwise, changes which are successful are silent and only those files which can't be corrected are reported.

FILES

/usr/options/*

DIAGNOSTICS

The diagnostics produced by **bomverify** are intended to be self-explanatory.

NAME

bootflags – display/change system boot flags

SYNOPSIS

bootflags [-p] [-c] [-v] [flag=value] ...

DESCRIPTION

bootflags displays or changes the flags that control system boot and reboot actions. If the **–p** option is not specified, only the temporary flags are affected.

–p Display or change the permanent boot flags rather than the temporary boot flags.

–c Copy temporary boot flags to the permanent boot flags. If the **–p** flag is given, copy the permanent boot flags into the temporary boot flags. The copy is done after all the changes to flags have been made.

–v Display the flags after they have been changed.

The assignments are as follows. All numeric values (f, ra0, ra1), are assumed to be in decimal unless they start with "0x", in which case they are taken in hexadecimal. String values (n0, n1) are taken literally as character strings.

Assignment	Effect
n0=*string*	Change primary boot name to *string*.
n1=*string*	Change secondary boot name to *string*.
f=*n*	Set boot flags to *n*.
ra0=*n*	Change primary reboot reconfiguration structure address to *n*.
ra1=*n*	Change secondary reboot reconfiguration structure address to *n*.

For example, the following command changes the permanent boot flags so that the secondary boot name is "sd(48,0)stand/dump sd(48,1) 4000 /dev/dsk/sd0s1", and the secondary reboot reconfiguration structure address is 0x30000:

bootflags -p 'n1=sd(48,0)stand/dump sd(48,1) 4000 /dev/dsk/sd0s1' 'ra1=0x30000'

Note the use of quotes to collect the arguments of the assignments together.

FILES

/dev/consmem Console memory

NAME

bp – binary patcher

SYNOPSIS

/etc/bp [**–qdh**] [**–s symfile**] *filename symbol*[**+offset**] [**new_value**]

DESCRIPTION

bp is a binary file patch program. When *new_value* is omitted, filename will be opened readonly and the value at symbol displayed. When *new_value* is specified, filename is opened for reading and writing, the old value at *symbol* is read, *new_value* is written, and both new and old values are displayed.

The **–q** option makes **bp** quiet about displaying anything. The **–d** and **–h** options set the display radix to decimal and hex respectively. *new_value* and *offset* are assumed to be decimal numbers unless they have a leading '0x' which is recognized as indicating a hex number.

Following the **–s** option is the name of a file to look up the value of *symbol*. *symfile* can be identical to *filename*. A leading underscore will be optionally added to *symbol* when looking for a match.

It is possible to specify an absolute address by omiting *symbol* and just specifying *offset*.

SEE ALSO

nm(1)

BUGS

bp reads and writes only in 4 byte quantities.

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 prints. If neither is specified, a calendar for the present month prints. The *month* is a number between 1 and 12; *year* can be between 1 and 9999. 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 this command:

cal 9 1752

NOTES

Note that **cal 83** refers to the early Christian era, not the 20th century.

BUGS

The year is always considered to start in January even though this is historically naive.

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, and 8/24 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 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

captoinfo – convert a termcap description into a terminfo description

SYNOPSIS

captoinfo [–v ...] [–V] [–1] [–w width] file ...

DESCRIPTION

captoinfo 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) is reduced to the minimum superset before being output.

If no *file* is given, the environment variable **TERMCAP** is used for the filename or entry. If **TERMCAP** 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 **TERMCAP** is not set, 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

SEE ALSO

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

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 that 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.

terminfo 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.

WARNINGS

Certain *termcap* defaults are assumed to be true. For example, the bell character (*terminfo bel*) is assumed to be Ctrl-G. The linefeed capability (*termcap nl*) is assumed to be the same for both *cursor_down* and *scroll_forward* (*terminfo cud1* 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 cm*, *terminfo cup*) sometimes produces a string which, though technically correct, may not be optimal. In particular, the rarely used *termcap* operation *%n* produces strings that are especially long. Most occurrences of these non-optimal strings are 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 has been removed.

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. The following command prints *file* on your terminal:

cat file

The next command concatenates *file1* and *file2*, and writes the results in *file3*:

cat file1 file2 >file3

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. ASCII control characters (octal 000 - 037) are printed as ^*n*, where *n* is the corresponding ASCII character in the range octal 100 - 137 (@, A, B, C, . . ., X, Y, Z, [, \,], ^, and _); the DEL character (octal 0177) is printed ^?. Other non-printable characters are printed as **M-***x*, where *x* is the ASCII character specified by the low-order seven bits.

When **cat** is invoked with the **–v** option, the following options can also be used:

–t Causes tabs to be printed as ^I's and formfeeds to be printed as ^L's.

–e Causes a **$** character to be printed at the end of each line (prior to the newline).

The **–t** and **–e** options are ignored if the **–v** option is not specified.

WARNINGS

Redirecting the output of **cat** onto one of the files being read will cause the loss of the data originally in the file being read. For example, typing the following command will cause the original data in *file1* to be lost.

cat file1 file2 >file1

SEE ALSO

cp(1), pg(1), pr(1)

NAME

cd – change working directory

SYNOPSIS

cd [directory]

DESCRIPTION

If *directory* is not specified, the value of shell parameter **$HOME** is used as the new working directory. If *directory* specifies a complete path starting with /, ., or .., *directory* becomes the new working directory. If neither case applies and the shell is **sh** or **ksh** , **cd** tries to find the designated directory relative to one of the paths specified by the **$CDPATH** shell variable. **csh** uses the variable **$cdpath** for this purpose. **$CDPATH** has the same syntax 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

pwd(1), sh(1), chdir(2)

NAME

cdc – change the delta commentary of an SCCS delta

SYNOPSIS

cdc –r*SID* [–**m**[*mrlist*]] [–**y**[*comment*]] *files*

DESCRIPTION

For each named file, **cdc** changes the delta commentary for the SID (SCCS **I**Dentification string) specified by the **–r** keyletter.

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 pathname does not begin with *s.*) and unreadable files are silently ignored. If a name of – 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 filenames. All the described keyletter arguments apply independently to each named file:

–r*SID* 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 newline 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

This example 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.

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

The next example does the same thing.

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

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.

WARNINGS

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

NAME

checkeq – check documents for correct eqn syntax.

SYNOPSIS

checkeq [*–dn*] [file] ...

DESCRIPTION

checkeq is used to check and report about missing or unbalanced delimters and .EQ/.EN pairs.

If no files are specified, **checkeq** reads from standard input. A line beginning with .EQ marks the start of an equation; the end of an equation is marked by a line beginning with .EN. It is also possible to set a character as a delimiter; subsequent text between delimiters is also treated as *eqn* input.

The **–d** option is used to specify a character to be used as a delimiter.

SEE ALSO

eqn(1), neq(1)

NAME

checkfsys – check a filesystem

DESCRIPTION

The **checkfsys** command allows the user to check for and optionally repair a damaged filesystem.

The user is asked to select one of the following three functions:

Check the filesystem:
No repairs are attempted.

Repair it interactively:
The user is informed about each instance of damage and asked if it should be repaired.

Repair it automatically:
The program applies a standard repair to each instance of damage.

The identical function is available under the **sysadm** menu:

sysadm checkfsys

The command may be assigned a password. See **sysadm**(1), the **admpasswd** sub-command.

SEE ALSO

sysadm(1), fsck(1M)

WARNINGS

While automatic and interactive checks are generally successful, they can occasionally lose a file or a file's name. Files with content but without names are put in the */filesystem/lost+found* directory.

NAME

chmod – change mode

SYNOPSIS

chmod mode file ...

chmod mode directory ...

DESCRIPTION

The permissions of the named files or directories are **c**hanged according to **mode**, which may be symbolic or absolute. Absolute changes to permissions are stated using octal numbers where *n* is a number from 0 to 7:

chmod *nnn file(s)*

Symbolic changes are stated using mnemonic characters:

chmod *a operator b file(s)*

a is one or more characters corresponding to user, group, or other; *operator* is +, –, or =, signifying assignment of permissions; and *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
20#0	Set group ID on execution if # is **7**, **5**, **3**, or **1** Enable mandatory locking if # is **6**, **4**, **2**, or **0**
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 **u**ser, **g**roup, and **o**thers all have **r**eading, **w**riting, and **e**xecution 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 permissions
g	group permissions
o	other 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 the set-group-ID and enable a file to be locked on execution at the same time. Therefore, the following examples are illegal usages and will elicit error messages.

chmod g+x,+l *file*

chmod g+s,+l *file*

Only the owner of a file or directory (or the superuser) can change a file's mode. Only the superuser can set the sticky bit on a non-directory file. If you are not superuser, **chmod** will mask the sticky-bit but will not return an error. 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.

Shell scripts cannot be suid or sgid.

EXAMPLES

The first example denies execution permission to all. The second (octal) example permits only reading permissions.

chmod a–x *file*

chmod 444 *file*

These commands make a file readable and writable by the group and others:

chmod go+rw *file*

chmod 666 *file*

The next command causes a file to be locked during access:

chmod +l *file*

The last two examples enable all to read, write, and execute the file and they turn on the set group-ID.

chmod =rwx,g+s *file*

chmod 2777 *file*

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 **chgrp** is invoked by other than the superuser, the set-user-ID and set-group-ID bits of the file mode will be cleared. The user invoking **chgrp** must be a member of the group to which the file is being changed and be the owner of that file, or be the superuser.

Only the superuser can change the owner of that file.

FILES

/etc/passwd
/etc/group

SEE ALSO

chmod(1), chown(2), group(4), passwd(4)

NAME

chroot – change root directory for a command

SYNOPSIS

/etc/chroot newroot command

DESCRIPTION

chroot causes the given command to be executed relative to the new root. The meaning of any initial slashes (/) in the pathnames 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, as shown below, the command will create the file *x* relative to the original root of the command, not the new one.

chroot newroot command >x

The new root pathname is always relative to the current root. Even if a **chroot** 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 superuser.

SEE ALSO

cd(1), chroot(2)

BUGS

One should exercise extreme caution when referencing device files in the new root filesystem.

NAME

chrtbl – generate character classification and conversion tables

SYNOPSIS

chrtbl [file]

DESCRIPTION

The **chrtbl** command creates a character classification table and an upper/lowercase conversion table. The tables are contained in a byte-sized array encoded such that a table lookup can be used to determine the character classification of a character or to convert a character (see **ctype**(3C)). The size of the array is 257*2 bytes: 257 bytes are required for the 8-bit code set character classification table and 257 bytes for the upper- to lowercase and lower- to uppercase conversion table.

chrtbl reads the user-defined character classification and conversion information from *file* and creates two output files in the current directory. One output file, *ctype.c* (a C-language source file), contains the 257*2-byte array generated from processing the information from *file*. You should review the content of *ctype.c* to verify that the array is set up as you had planned. (In addition, an application program could use *ctype.c .)* The first 257 bytes of the array in *ctype.c* are used for character classification. The characters used for initializing these bytes of the array represent character classifications that are defined in */usr/include/ctype.h*; for example, **_L** means a character is lower case and **_S| _B** means the character is both a spacing character and a blank. The last 257 bytes of the array are used for character conversion. These bytes of the array are initialized so that characters for which you do not provide conversion information will be converted to themselves. When you do provide conversion information, the first value of the pair is stored where the second one would be stored normally, and vice versa. For example, if you provide **<0x41 0x61>**, then **0x61** is stored where **0x41** would be stored normally, and **0x61** is stored where **0x41** would be stored normally.

The second output file (a data file) contains the same information, but is structured for efficient use by the character classification and conversion routines (see **ctype**(3C)). The name of this output file is the value of the character classification *chrclass* read in from *file*. This output file must be installed in the */lib/chrclass* directory under this name by someone who is superuser or a member of group **bin**. This file must be readable by user, group, and other; no other permissions should be set. To use the character classification and conversion tables on this file, set the environmental variable **CHRCLASS** (see **environ**(5)) to the name of this file and export the variable. For example, if the name of this file (and character class) is *xyz*, you should issue these commands: **CHRCLASS=xyz ; export CHRCLASS**

If no input file is given, or if the argument **–** is encountered, **chrtbl** reads from the standard input file.

The syntax of *file* allows the user to define the name of the data file created by **chrtbl**, the assignment of characters to character classifications, and the relationship between upper- and lowercase letters. The character classifications recognized by **chrtbl** are as follows:

chrclass Name of the data file to be created by **chrtbl**.

isupper Character codes to be classified as uppercase letters.

islower Character codes to be classified as lowercase letters.

isdigit Character codes to be classified as numeric.

isspace Character codes to be classified as a spacing (delimiter) character.

ispunct Character codes to be classified as a punctuation character.

iscntrl Character codes to be classified as a control character.

isblank Character code for the space character.

isxdigit Character codes to be classified as hexadecimal digits.

ul Relationship between upper- and lowercase characters.

Any lines with the number sign (#) in the first column are treated as comments and are ignored. Blank lines are also ignored.

A character can be represented as a hexadecimal or octal constant (for example, the letter **a** can be represented as 0x61 in hexadecimal or 0141 in octal). Hexadecimal and octal constants may be separated by one or more space and tab characters.

The hyphen (-) may be used to indicate a range of consecutive numbers. Zero or more space characters may be used for separating the dash character from the numbers.

The backslash character (\) is used for line continuation. Only a carriage return is permitted after the backslash character.

The relationship between upper- and lowercase letters (**ul**) is expressed as ordered pairs of octal or hexadecimal constants: *<uppercase_character lowercase_character>*. These two constants may be separated by one or more space characters. Zero or more space characters may be used for separating the angle brackets (< >) from the numbers.

EXAMPLES

The following is an example of an input file used to create the ASCII code set definition table on a file named *ascii*.

```
chrclass  ascii
isupper   0x41 - 0x5a
islower   0x61 - 0x7a
isdigit   0x30 - 0x39
isspace   0x20 0x9 - 0xd
ispunct   0x21 - 0x2f 0x3a - 0x40    \
          0x5b - 0x60 0x7b - 0x7e
iscntrl   0x0 - 0x1f  0x7f
isblank   0x20
isxdigit  0x30 - 0x39 0x61 - 0x66    \
          0x41 - 0x46
ul        <0x41 0x61> <0x42 0x62> <0x43 0x63>  \
          <0x44 0x64> <0x45 0x65> <0x46 0x66>  \
          <0x47 0x67> <0x48 0x68> <0x49 0x69>  \
          <0x4a 0x6a> <0x4b 0x6b> <0x4c 0x6c>  \
          <0x4d 0x6d> <0x4e 0x6e> <0x4f 0x6f>  \
```

```
<0x50 0x70> <0x51 0x71> <0x52 0x72>  \
<0x53 0x73> <0x54 0x74> <0x55 0x75>  \
<0x56 0x76> <0x57 0x77> <0x58 0x78>  \
<0x59 0x79> <0x5a 0x7a>
```

FILES

/lib/chrclass/* Data file containing character classification and conversion tables created by **chrtbl**

/usr/include/ctype.h
Header file containing information used by character classification and conversion routines

SEE ALSO

ctype(3C), environ(5)

DIAGNOSTICS

The error messages produced by **chrtbl** are intended to be self-explanatory. They indicate errors in the command line or syntactic errors encountered within the input file.

NAME

ci – check in RCS revisions

SYNOPSIS

ci [options] file ...

DESCRIPTION

ci stores new revisions into RCS files. Each file name ending in *,v* is taken to be an RCS file, all others are assumed to be working files containing new revisions. **ci** deposits the contents of each working file into the corresponding RCS file.

Pairs of RCS files and working files may be specified in three ways (see the example section of **co**(1)):

1) Both the RCS file and the working file are given. The RCS filename is of the form *path1/workfile,v* and the working filename is of the form *path2/workfile*, where *path1/* and *path2/* are (possibly different or empty) paths and *workfile* is a filename.

2) Only the RCS file is given. The working file is assumed to be in the current directory and its name is derived from the name of the RCS file by removing *path1/* and the suffix *,v*.

3) Only the working file is given. The name of the RCS file is derived from the name of the working file by removing *path2/* and appending the suffix *,v*.

If the RCS file is omitted or specified without a path, **ci** looks for the RCS file first in the directory *./RCS* , then in the current directory.

For **ci** to work, the caller's login must be on the access list, except if the access list is empty or the caller is the superuser or the owner of the file. To append a new revision to an existing branch, the tip revision on that branch must be locked by the caller. Otherwise, only a new branch can be created. This restriction is not enforced for the owner of the file, unless locking is set to *strict* (see **rcs**(1)). A lock held by someone else may be broken with the **rcs** command.

Normally, **ci** checks whether the revision to be deposited is different from the preceding one. If it is not different, **ci** either aborts the deposit (if **–q** is given) or asks whether to abort (if **–q** is omitted). A deposit can be forced with the **–f** option.

For each revision deposited, **ci** prompts for a log message. The log message should summarize the change and must be terminated with a line containing a single '.' or a Ctrl-D. If several files are checked in, **ci** asks whether to reuse the previous log message. If the standard input is not a terminal, **ci** suppresses the prompt and uses the same log message for all files. See also **–m**.

The number of the deposited revision can be given by any of the options **–r**, **–f**, **–k**, **–l**, **–u**, or **–q** (see **–r**).

If the RCS file does not exist, **ci** creates it and deposits the contents of the working file as the initial revision (default number: 1.1). The access list is initialized to empty. Instead of the log message, **ci** requests descriptive text (see **–t** below).

-r[*rev*] Assigns the revision number *rev* to the checked-in revision, releases the corresponding lock, and deletes the working file. This is also the default.

If *rev* is omitted, **ci** derives the new revision number from the caller's last lock. If the caller has locked the tip revision of a branch, the new revision is appended to that branch. The new revision number is obtained by incrementing the tip revision number. If the caller locked a non-tip revision, a new branch is started at that revision by incrementing the highest branch number at that revision. The default initial branch and level numbers are 1. If the caller holds no lock, but he is the owner of the file and locking is not set to *strict*, then the revision is appended to the trunk.

If *rev* indicates a revision number, it must be higher than the latest one on the branch to which *rev* belongs, or it must start a new branch.

If *rev* indicates a branch instead of a revision, the new revision is appended to that branch. The level number is obtained by incrementing the tip revision number of that branch. If *rev* indicates a non-existing branch, that branch is created with the initial revision numbered *rev.1*.

Exception: On the trunk, revisions can be appended to the end, but not inserted.

-f[*rev*] Forces a deposit; the new revision is deposited even it is not different from the preceding one.

-k[*rev*] Searches the working file for keyword values to determine its revision number, creation date, author, and state (see **co**(1)), and assigns these values to the deposited revision, rather than computing them locally. A revision number given by a command option overrides the number in the working file. This option is useful for software distribution. A revision that is sent to several sites should be checked in with the **-k** option at these sites to preserve its original number, date, author, and state.

-l[*rev*] Works like **-r**, except it performs an additional **co -l** for the deposited revision. Thus, the deposited revision is immediately checked out again and locked. This is useful for saving a revision although one wants to continue editing it after the check-in.

-u[*rev*] Works like **-l**, except that the deposited revision is not locked. This is useful if one wants to process (e.g., compile) the revision immediately after check-in.

-q[*rev*] Quiet mode; diagnostic output is not printed. A revision that is not different from the preceding one is not deposited, unless **-f** is given.

-m*msg* Uses the string *msg* as the log message for all revisions checked in.

-n*name* Assigns the symbolic name *name* to the number of the checked-in revision. **ci** prints an error message if *name* is already assigned to another number.

-N*name* Same as **-n**, except that it overrides a previous assignment of *name*.

-s*state* Sets the state of the checked-in revision to the identifier *state*. The default is *Exp*.

-t[*txtfile*] Writes descriptive text into the RCS file (deletes the existing text). If *txtfile* is omitted, **ci** prompts the user for text supplied from the standard input, terminated with a line containing a single '.' or Ctrl-D. Otherwise, the descriptive text is copied from the file *txtfile*. During initialization, descriptive text is requested even if **-t** is not given. The prompt is suppressed if standard input is not a terminal.

File Modes

An RCS file created by **ci** inherits the read and execute permissions from the working file. If the RCS file exists already, **ci** preserves its read and execute permissions. **ci** always turns off all write permissions of RCS files.

FILES

The caller of the command must have read/write permission for the directories containing the RCS file and the working file, and read permission for the RCS file itself. A number of temporary files are created. A semaphore file is created in the directory containing the RCS file. **ci** always creates a new RCS file and unlinks the old one. This strategy makes links to RCS files useless.

SEE ALSO

co(1), ident(1), rcs(1), rcsdiff(1), rcsintro(1), rcsmerge(1), rlog(1), sccstorcs(1), rcsfile(4)

NOTES

Copyright 1982 by Walter F. Tichy.

DIAGNOSTICS

For each revision, **ci** prints the RCS file, the working file, and the number of both the deposited and the preceding revision. The exit status always refers to the last file checked in and is 0 if the operation was successful, 1 otherwise.

NAME

ckbupscd – check filesystem backup schedule

SYNOPSIS

/etc/ckbupscd [**–m**]

DESCRIPTION

ckbupscd consults the file */etc/bupsched* and prints the filesystem lists from lines with date and time specifications matching the current time. If the **–m** flag is present, an introductory message in the output is suppressed so that only the filesystem lists are printed. Entries in the */etc/bupsched* file are printed under the control of **cron.**

The commands **bupsched/schedcheck** (part of the **sysadm**(1M) package) are provided to review and edit the */etc/bupsched* file.

The file */etc/bupsched* should contain lines of 4 or more fields, separated by spaces or tabs. The first 3 fields (the schedule fields) specify a range of dates and times. The rest of the fields constitute a list of names of filesystems to be printed if **ckbupscd** is run at some time within the range given by the schedule fields. The general format is as follows:

time1[,time] day[,day] month[,month] fsyslist

where:

time Specifies an hour of the day (**0** through **23**), matching any time within that hour or an exact time of day (**0:00** through **23:59**).

day Specifies a day of the week (**sun** through **sat**) or day of the month (**1** through **31**).

month Specifies the month in which the time and day fields are valid. Legal values are the month numbers (**1** through **12**).

fsyslist The rest of the line is taken to be a filesystem list to print.

Multiple time, day, and month specifications may be separated by commas, in which case they are evaluated left to right.

An asterisk (*) always matches the current value for that field.

A line beginning with a pound sign (#) is interpreted as a comment and ignored.

The longest line allowed (including continuations) is 1024 characters.

EXAMPLES

The following are examples of lines which could appear in the */etc/bupsched* file.

```
06:00-09:00 fri 1,2,3,4,5,6,7,8,9,10,11 /applic
```

Prints the filesystem name */applic* if **ckbupscd** is run between 6:00am and 9:00am any Friday during any month except December.

```
00:00-06:00,16:00-23:59 1,2,3,4,5,6,7 1,8 /
```

Prints a reminder to back up the root (/) filesystem if **ckbupscd** is run between the times of 4:00pm and 6:00am during the first week of August or January.

FILES

/etc/bupsched Specification file containing times and filesystem to back up

SEE ALSO

sysadm(1), cron(1M) echo(1), sh(1)

BUGS

ckbupscd will report filesystems due for backup if invoked any time in the window. It does not know that backups may have just been taken.

NAME

clri – clear inode

SYNOPSIS

/etc/clri filesystem i-number ...

DESCRIPTION

N.B.: clri is obsoleted for normal filesystem repair work by **fsck**(1M).

clri writes zeros on the inodes with the decimal *i-numbers* on the *filesystem*. After **clri**, any blocks in the affected file will show up as "missing" in an **icheck**(1M) of the *filesystem*.

Read and write permission is required on the specified filesystem device. The inode becomes allocatable.

The primary purpose of this routine is to remove a file which for some reason appears in no directory. If it is used to zap an inode which does appear in a directory, care should be taken to track down the entry and remove it. Otherwise, when the inode is reallocated to some new file, the old entry will still point to that file. At that point, removing the old entry will destroy the new file. The new entry will again point to an unallocated inode, so the whole cycle is likely to be repeated again and again.

SEE ALSO

icheck(1M)

BUGS

If the file is open, **clri** is likely to be ineffective.

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, **cmp** 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.

The following options are available:

–l 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

co – check out RCS revisions

SYNOPSIS

co [options] file ...

DESCRIPTION

co retrieves revisions from RCS files. Each filename ending in *,v* is taken to be an RCS file. All other files are assumed to be working files. **co** retrieves a revision from each RCS file and stores it in the corresponding working file.

Pairs of RCS files and working files may be specified in three ways (see the **EXAMPLE** section).

1) Both the RCS file and the working file are given. The RCS filename is of the form *path1/workfile,v* and the working filename is of the form *path2/workfile*, where *path1/* and *path2/* are (possibly different or empty) paths and *workfile* is a filename.

2) Only the RCS file is given. The working file is created in the current directory and its name is derived from the name of the RCS file by removing *path1/* and the suffix *,v*.

3) Only the working file is given. The name of the RCS file is derived from the name of the working file by removing *path2/* and appending the suffix *,v*.

If the RCS file is omitted or specified without a path, **co** looks for the RCS file first in the directory *./RCS* and then in the current directory.

Revisions of an RCS file may be checked out locked or unlocked. Locking a revision prevents overlapping updates. A revision checked out for reading or processing (e.g., compiling) need not be locked. A revision checked out for editing and later check-in must normally be locked. Locking a revision currently locked by another user fails. (A lock may be broken with the **rcs**(1) command.) **co** with locking requires the caller to be on the access list of the RCS file, unless he is the owner of the file or the superuser, or the access list is empty. **co** without locking is not subject to accesslist restrictions.

A revision is selected by number, checkin date/time, author, or state. If none of these options are specified, the latest revision on the trunk is retrieved. When the options are applied in combination, the latest revision that satisfies all of them is retrieved. The options for date/time, author, and state retrieve a revision on the *selected branch*. The selected branch is either derived from the revision number (if given), or is the highest branch on the trunk. A revision number may be attached to one of the options **–l**, **–p**, **–q**, or **–r**.

A **co** command applied to an RCS file with no revisions creates a zero-length file. **co** always performs keyword substitution (see below).

–l[*rev*] Locks the checked-out revision for the caller. If omitted, the checked-out revision is not locked. See option **–r** for handling of the revision number *rev*.

–p[*rev*] Prints the retrieved revision on the standard output rather than storing it in the working file. This option is useful when **co** is part of a pipe.

–q[*rev*] Quiet mode; diagnostics are not printed.

–d*date* Retrieves the latest revision on the selected branch whose check-in date/time is less than or equal to *date*. The date and time may be given in free format and are converted to local time. Following are examples of formats for *date*:

```
22-April-1982, 17:20-CDT,
2:25 AM, Dec. 29, 1983,
Tue-PDT, 1981, 4pm Jul 21      (free format),
Fri, April 16 15:52:25 EST 1982 (output of ctime).
```

Most fields in the date and time may be defaulted. **co** determines the defaults in this order: year, month, day, hour, minute, and second (most to least significant). At least one of these fields must be provided. For omitted fields that are of higher significance than the highest provided field, the current values are assumed. For all other omitted fields, the lowest possible values are assumed. For example, the date `20, 10:30` defaults to 10:30:00 of the 20th of the current month and current year. The date/time must be quoted if it contains spaces.

–r[*rev*] Retrieves the latest revision whose number is less than or equal to *rev*. If *rev* indicates a branch rather than a revision, the latest revision on that branch is retrieved. *rev* is composed of one or more numeric or symbolic fields separated by a dot (.). The numeric equivalent of a symbolic field is specified with the **–n** option of the commands **ci** and **rcs**.

–s*state* Retrieves the latest revision on the selected branch whose state is set to *state*.

–w[*login*] Retrieves the latest revision on the selected branch that was checked in by the user with login name *login*. If the argument *login* is omitted, the caller's login is assumed.

–j*joinlist* Generates a new revision that is the join of the revisions on *joinlist*. *joinlist* is a comma-separated list of pairs of the form *rev2:rev3*, where *rev2* and *rev3* are (symbolic or numeric) revision numbers. For the initial such pair, *rev1* denotes the revision selected by the options **–l**, ..., **–w**. For all other pairs, *rev1* denotes the revision generated by the previous pair. (Thus, the output of one join becomes the input to the next.)

For each pair, **co** joins revisions *rev1* and *rev3* with respect to *rev2*. This means that all changes that transform *rev2* into *rev1* are applied to a copy of *rev3*. This is particularly useful if *rev1* and *rev3* are the ends of two branches that have *rev2* as a common ancestor. If *rev1* < *rev2* < *rev3* on the same branch, joining generates a new revision that is like *rev3*, but with all changes that lead from *rev1* to *rev2* undone. If changes from *rev2* to *rev1* overlap with changes from *rev2* to *rev3*, **co** prints a warning and includes the overlapping sections, delimited by the lines <<<<<<< *rev1*, =======, and >>>>>>> *rev3*.

For the initial pair, *rev2* may be omitted. The default is the common ancestor. If any of the arguments indicate branches, the latest revisions on those branches are assumed. If the option **–l** is present, the initial *rev1* is locked.

Keyword Substitution

Strings of the form *$keyword$* and *$keyword:...$* embedded in the text are replaced with strings of the form *$keyword: value $*, where *keyword* and *value* are pairs listed below. Keywords may be embedded in literal strings or comments to identify a revision.

Initially, the user enters strings of the form *$keyword$*. On checkout, **co** replaces these strings with strings of the form *$keyword: value $*. If a revision containing strings of the latter form is checked back in, the value fields will be replaced during the next checkout. Thus, the keyword values are automatically updated on checkout.

Following are the keywords and their corresponding values:

$Author$	The login name of the user who checked in the revision.
$Date$	The date and time the revision was checked in.
$Header$	A standard header containing the RCS filename, the revision number, the date, the author, and the state.
$Locker$	The login name of the user who locked the revision (empty if not locked).
Log	The log message supplied during check-in, preceded by a header containing the RCS filename, the revision number, the author, and the date. Existing log messages are NOT replaced. Instead, the new log message is inserted after *$Log:...$*. This is useful for accumulating a complete change log in a source file.
$Revision$	The revision number assigned to the revision.
$Source$	The full pathname of the RCS file.
$State$	The state assigned to the revision with **rcs –s** or **ci –s**.

File Modes

The working file inherits the read and execute permissions from the RCS file. In addition, the owner write permission is turned on unless the file is checked out unlocked and locking is set to *strict* (see **rcs**(1)).

If a file with the name of the working file exists already and has write permission, **co** aborts the checkout if **–q** is given, or asks whether to abort if **–q** is not given. If the existing working file is not writable, it is deleted before the checkout.

EXAMPLES

Suppose the current directory contains a subdirectory *RCS* with an RCS file *io.c,v*. Then all of the following commands retrieve the latest revision from *RCS/io.c,v* and store it in *io.c*.

```
co io.c;  co RCS/io.c,v;  co io.c,v;
co io.c RCS/io.c,v;  co io.c io.c,v;
co RCS/io.c,v io.c;  co io.c,v io.c;
```

FILES

The caller of the command must have write permission in the working directory, read permission for the RCS file, and either read permission (for reading) or read/write permission (for locking) in the directory containing the RCS file.

A number of temporary files are created. A semaphore file is created in the directory of the RCS file to prevent simultaneous update.

SEE ALSO

ci(1), ident(1), rcs(1), rcsdiff(1), rcsintro(1), rcsmerge(1), rlog(1), sccstorcs(1), rcsfile(4)

NOTES

Copyright 1982 by Walter F. Tichy.

DIAGNOSTICS

The RCS filename, the working filename, and the revision number retrieved are written to the diagnostic output. The exit status always refers to the last file checked out and is 0 if the operation was successful, 1 otherwise.

WARNINGS

The option **–d** gets confused in some circumstances, and accepts no date before 1970.

There is no way to suppress the expansion of keywords, except by writing them differently. In **nroff**, this is done by embedding the null-character '\&' into the keyword.

BUGS

The option **–j** does not work for files that contain lines with a single '.'.

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) in the *DOCUMENTER'S WORKBENCH Technical Discussion and Reference Manual*

NOTES

The input format accepted by **col** matches the output produced by **nroff** with either the **–T37** or **–Tlp** 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 **–Tlp** 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

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 pathname 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 reconstructed 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 filename, size (in blocks) after combining, original size (also in blocks), and percentage change computed as follows:

100 * (original – combined) / original

Before any SCCS files are actually combined, it is recommended that you use this option to determine exactly how much space is saved by the combining process.

–p SID The SCCS identification string (SID) of the oldest delta to be preserved. All older deltas are discarded in the reconstructed file.

–c *list* 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 might 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 three columns of output: lines only in *file1*, lines only in *file2*, and lines in both files. The filename – 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 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 forces compression of *name*, even if it does not actually shrink or the corresponding *name*.Z file already exists. If **–f** is not given, the user is prompted as to whether an existing *name*.Z file should be overwritten, except when **compress** is run in the background under */bin/sh*.

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

config – build system configuration files

SYNOPSIS

/etc/config [**–c** configuration-type] [**–m** machine-type]

/etc/config –v

DESCRIPTION

config builds a set of kernel files from configuration files that describes the system being configured. The configuration files include *system configuration files*, *system module files*, and *module configuration files* (see the **FILES** section below). The kernel files produced are as follows:

conf.c Contains a description of the I/O devices attached to the system, as well as a description of the block and character device table entry points.

Makefile A file used by **make**(1) in building the system.

config.h A header file compiled with *conf.c* and any *module space files*.

config should be run either from the *uts* subdirectory of the system source (usually */usr/src/uts*) or from the *uts* subdirectory of the binary distribution (usually */etc/conf/uts*).

The **–c** option specifies the configuration type; the default is **std**. The **–m** option specifies which machine type to use. The *system configuration files* and *system modules files* are found in the subdirectory whose name corresponds to the *machine-type*. The kernel files: *conf.c*, *config.h*, and *Makefile* created by **config** are placed in the subdirectory named *machine-type.configuration-type*.

config –v displays the version levels supported by **config(1m)**. Refer to the description of the *version* section in **sys_config**(4).

If you receive any error messages from **config**, you should fix the problems in your configuration file and try again. If you try to compile a system that had configuration errors, you will likely meet with failure.

The *Makefile* produced can be used to generate a new kernel by taking the *conf.c* and *config.h* files, the modules *.os* and *.a*, and any *module space files* and *module stub files*.

FILES

The following files are in the source tree as */usr/src/uts/** and in the binary distribution tree as */etc/conf/uts/**:

system configuration files	These take the form {machine}/*.{configuration}
system module files	These take the form {machine}/modules and *.mod
module configuration files	This takes the form {module}/{submodule}/config
module stub files	This takes the form {module}/{submodule}/*_stubs.c
module space files	These take the form {module}/{submodule}/*_space.c and are specified in the *module configuration files*

The following files are standard:

symmetry/system.std	The standard symmetry configuration file

symmetry/modules The base symmetry modules list

SEE ALSO

sys_config(4), mod_config(4)

SYNOPSIS section of each device in section 7

NAME

cocreate, cosend, cocheck, coreceive, codestroy – communicate to a process

SYNOPSIS

cocreate [**–r** rpath] [**–w** wpath] [**–i** id] [**–R** refname] [**–s** send_string] [**–e** expect_string] command
cosend [**–n**] id string
cocheck id
coreceive id
codestroy [**–R** refname] id

DESCRIPTION

The **cocreate** built-in initializes communication to a process using named-pipes. This means that the process expects strings on its input and sends information on its output.

The **cosend** built-in works two ways. If **–n** is specified, **cosend** does not wait for a response. The process should use **vsig** to force the strings into the pipe and then signal that it wishes to send. This causes a reread to occur in the current frame. The **vsig** executable is described in detail on the **vsig**(1F) manual page.

The **cocheck** built-in should be called from a `reread` descriptor. The default value of one of the fields in the form should include the **coreceive**.

If **–n** is not specified, *send_string* and *expect_string* are used to tell when input and output are completed on the pipe. In other words, during a **cosend** the Interpreter outputs all the strings given as arguments followed by *send_string*, to say that it is done giving information. Then it reads all the output from the process until it sees *expect_string*. By default, the Interpreter sends no *send_string* and expects no *expect_string* (it expects only one line of output from the process). Read the warning below if you use **cosend** without the **–n** option.

The **codestroy** built-in should usually be given the **–R** option, as you may have more than one process with the same name, and you don't want to kill the wrong one. It keeps track of the number of *refnames* you have assigned and when the last one is killed, kills the process (*id*) for you.

The *id* is used to refer to the process identification number. If none is specified, the process name is used.

The *refname* is a local name for a process. This is useful when multiple objects reference the same process (i.e., when multiple objects perform a **cocreate** on the same process). Thus, when a **codestroy** operation is performed, you usually want to destroy only the local reference to the process rather than the entire pipe.

The *rpath* tells **cocreate** what file to use to read information from. The *wpath* tells **cocreate** what file to use to write information to. These files are usually used for processes that naturally write to a certain pipe or for having one process talk to many different Interpreters. If *rpath* and *wpath* are not specified, paths are picked in *$HOME/tmp*.

The *command* argument specifies a program followed by its arguments.

Here is some advice for writing these programs. If this program is to be written in C, make sure to flush output after writing to the pipe (a good way to check this is to run **cat | prog | cat** from shell). As of this writing, **awk**(1) and **sed**(1) cannot be used because they do not flush after lines of output. Shell scripts are well-mannered, but slow. C language is recommended. If possible, use the default *send_string*, *rpath* and *wpath*. In most cases, the *expect_string* must be specified. (Note: the *expect_string* need only be the initial part of the line and there must be a newline at the end of the output). The *id* is typically used when the same process is used with different options and different meanings.

The **codestroy** built-in usually works best in **close=** lines in menus and forms. The **close=** is guaranteed to be evaluated when a window is closed.

EXAMPLES

```
    .
    .
    .
init='cocreate -i BIGPROCESS initialize'
close='codestroy BIGPROCESS'
    .
    .
    .
reread='cocheck BIGPROCESS'

name='cosend -n BIGPROCESS field1'
    .
    .
    .
name="Receive field"
inactive=TRUE
value='coreceive BIGPROCESS'
```

WARNINGS

If **cosend** is used without the **-n** option, a coprocess that does not answer causes the Interpreter to hang permanently

SEE ALSO

awk(1), cat(1), sed(1)

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 **sh**(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 permissable; if not, the command exits. For **mv**, when the parent directory of file1 is writeable and has the sticky bit set, one or more of the following conditions must be true:

The user must own the file
The user must own the directory
The file must be writeable by the user
The user must be the superuser

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 to **ln** will cause a symbolic link to be created rather than the default of a hard 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 superuser. 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, or 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 link across filesystems when creating hard links. This restriction is necessary because filesystems can be added and removed.

BUGS

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

NAME

cpio – copy file archives in and out

SYNOPSIS

cpio –o [**acBvVx**] [**–C** *bufsize*] [[**–O** *file*] [**–M** *message*]]

cpio –i [**BcdLmrtuvVfsSb6kx**] [**–C** *bufsize*] [[**–I** *file*] [**–M** *message*]] [*pattern ...*]

cpio –p [**adlmuvVx**] *directory*

DESCRIPTION

cpio –o (copy out) reads the standard input to obtain a list of pathnames and copies those files onto the standard output together with pathname and status information. Output is padded to a 512-byte boundary by default.

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 filename-generating notation of **sh**(1). In *patterns*, metacharacters **?**, *****, and **[...]** match the slash (/) character, and backslash (\) is an escape character. A **!** metacharacter means *not*. (For example, the **!abc*** pattern would exclude all files that begin with **abc**.) Multiple *patterns* may be specified and if no *patterns* are specified, the default for *patterns* is * (i.e., select all files). Each *pattern* must be enclosed in double quotes, otherwise the name of a file in the current directory is used. Extracted files are conditionally created and copied into the current directory tree based upon the options described below.

POSIX compliance requires that the users's **umask** value shall modify file creation modes unless the root user is executing the program. The AT&T semantics were such that **cpio** would set the **umask** value to 0 before creating files. This caused the permissions of the files to match those of the previous **cpio –o**. This is now done only when the root user executes the program. Non-root users wishing to have files extracted with the same modes as appear on the input file must set the **umask** value to 0 before executing the **cpio** program.

The owner and group of the files will be that of the current user unless the user is superuser, which causes **cpio** to retain the owner and group of the files of the previous **cpio –o**.

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 pathnames 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 as follows:

–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 Reverse the order of the *bytes* within each word. Use only with the **–i** option.

–B Input/output is to be blocked 5,120 bytes to the record. The default buffer size is 512 bytes when this and the **C** options are not used. (**–B** does not apply to the pass option; **–B** is meaningful only with data

directed to or from a character special device such as */dev/rmt/xt0*.)

–c Write header information in ASCII *character* form for portability. Always use this option when origin and destination machines are different types.

–C *bufsize*
Input/output is to be blocked *bufsize* bytes to the record, where *bufsize* is replaced by a positive integer. The default buffer size is 512 bytes when this option and the **B** options are not used. (**–C** does not apply to the pass option; **–C** is meaningful only with data directed to or from a character special device such as */dev/rmt/xt0*.)

–d *directories* are to be created as needed.

–f Copy in all *files* except those in *patterns*. (See the paragraph on **cpio –i** for a description of *patterns*.)

–I *file* Read the contents of *file* as input. If *file* is a character special device, when the first medium is full, replace the medium and type a carriage return to continue to the next medium. Use only with the **–i** option.

–k Attempt to skip corrupted file headers and I/O errors that may be encountered. If you want to copy files from a medium that is corrupted or out of sequence, this option lets you read only those files with good headers. (For **cpio** archives that contain other **cpio** archives, if an error is encountered, **cpio** may terminate prematurely. **cpio** will find the next good header, which may be one for a smaller archive, and terminate when the smaller archive's trailer is encountered.) Used only with the **–i** option.

–L Follow symbolic links rather than copying them to tape. Usable only with the **–i** option.

–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.

–M *message*
Define a message to use when switching media. When you use the **–O** or **–I** options and specify a character special device, you can use this option to define the message that is printed when you reach the end of the medium. One **%d** can be placed in the message to print the sequence number of the next medium needed to continue.

–O *file* Direct the output of cpio to *file*. If *file* is a character special device, when the first medium is full, replace the medium and type a carriage return to continue to the next medium. Use only with the **–o** option.

–r Interactively *rename* files. If the user types a null line, the file is skipped. If the user types a "." the original pathname will be copied. (Not available with **cpio –p**.)

–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.

-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 filenames 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)).

-V *Special*Verbose: print a dot for each file seen. Useful to assure the user that **cpio** is working without printing out all filenames.

-x Ignore block special and character special files. Special files do not have any meaning to the operating system when reading a tape from a foreign system. Therefore, it is reasonable to ignore them and not restore them. Similarily, the device names and major/minor numbers of the operating system do not have any meaning for a foreign system.

-6 Process an old (i.e., UNIX System *Sixth* Edition format) file. Use only with the **-i** option.

NOTE: **cpio** assumes four-byte words.

If **cpio** reaches end of medium (end of a tape for example), when writing to (**-o**) or reading from (**-i**) a character special device, and **-O** and **-I** aren't used, **cpio** will print the message:

```
If you want to go on, type device/file name when ready.
```

To continue, you must replace the medium. Then type the character special device name and press carriage return. You may want to continue by directing **cpio** to use a different device. For example, if you have two tape drives, you may want to switch between them so **cpio** can proceed while you are changing the tapes. (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*). The **c** option ensures that the file will be portable to other machines. Instead of **ls**(1), you could use **find**(1), **echo**(1), **cat**(1), etc. to pipe a list of names to **cpio**. You could direct the output to a device instead of a file.

ls | cpio -oc > *../newfile*

cpio -i uses the output file of **cpio -o** (directed through a pipe with **cat** in the example), extracts 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. The **c** option is used when the file is created with a portable header. If no patterns were given, all files from *newfile* would be placed in the directory.

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

cpio -p takes the filenames piped to it and copies or links (**-l** option) those files to another directory on your machine (*newdir* in the example). The **-d** option says to create directories as needed. The **-m** option says retain the modification time. (It is important to use the **-depth** option of **find**(1) to generate pathnames 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), cat(1), echo(1), find(1), ls(1), tar(1), cpio(4)

NOTES

1) Pathnames are restricted to 256 characters.
2) Only the superuser can copy special files.
3) Blocks are reported in 512-byte quantities.
4) If a file has 000 permissions, contains more than 0 characters of data, and the user is not **root**, the file is not saved or restored.
5) The operating system uses 32 bits to represent a device. The standard **cpio** header does not provide space for devices represented by more than 16 bits. The operating system uses the header field "c_filesize" to hold the extra bits of a device when special files are placed on **cpio** files. This is allowed by the POSIX specification but may not be compatible with some other implementations. For this reason, use the **-x** option to prevent block and character files from being placed on the **cpio** files when generating files that are to be portable to other systems.

NAME

cpset – copy file and set mode, user, and group IDs

SYNOPSIS

/usr/bin/cpset [**–o**] filename destination [mode [owner [group]]]

DESCRIPTION

The **cpset** command is most commonly used in makefiles to copy a file (update a target file) to a specific place within a filesystem. The file is copied into the specified directory and if the mode, owner, and group are not specified, **cpset** sets them to some default values.

The options and arguments are as follows:

–o If the file already exists in the destination directory, the existing file is saved by copying it to *OLDfile* in the destination directory. This option is useful when re-installing a frequently used file such as */bin/sh* or */etc/getty*, where the existing file cannot be removed.

mode Sets the mode of the new file to *mode*. If this option is not specified, the default mode is 755.

user Sets the owner of the new file to *user*. This option is available only to the superuser. If the effective uid is less than 100, the default value for user is **bin**, otherwise the effective uid is used.

group Sets the group of the new file to *group*. This option is available only to the superuser. If the effective uid is less than 100, the default value for group is **bin**, otherwise the effective gid is used.

SEE ALSO

install(1)

NOTES

Because AT&T may not support **cpset** in their next release, the use of this command in portable makefiles should be avoided.

NAME

crash – examine system images

SYNOPSIS

/etc/crash [system_image] [system_namelist]

DESCRIPTION

crash is used to examine system memory images interactively. The **help** command lists available commands.

help lists the following commands:

Name	Long Name or Option	Description
(D)	Debug	change debug settings
(dis)	disassemble	disassemble addr count
(buffer)	buffer	buffer data
(hdr)	bufhdr	buffer headers
(dba)	dballoc	display dballoc table
	value(s)	dballoc modifier (give class)
(dbf)	dbfree	display free data blocks
	value(s)	dbfree modifier (give class)
(dbl)	dblock	display allocated stream dblocks
	–e	dblock modifier (all dblocks)
	–c *value*(s)	dblock modifier (give class)
	value(s)	dblock identifier (give slot # or addr)
(dev)	dev	display devsw
(dp)	dp	dump physical i.e., dp addr [*count*] [**dhoxDHOXcs**]
(dv)	dv	dump virtual i.e., dv addr [*count*] [**dhoxDHOXcs**]
(file)	file	file table
(inode)	inode	inode table
(kfp)	kfp	frame pointer for kernel stack trace start
(every)	every	repeat periodically
(ex)	expr	evaluate expressions
(e)	engine	display engine tables
(link)	linkblk	display linkblk table
	–e	linkblk modifier (all entries)
	value(s)	linkblk identifier (give slot # or addr)
(mbf)	mbfree	display free stream msgs
(mbl)	mblock	display allocated stream mblocks
	–e	mblock modifier (all mblocks)
	value(s)	mblock identifier (give slot # or addr)
(p)	proc	process table (one per line)
	–l	process table (long listing)
	–r	RUN & ONPROC procs only

	-o	ONPROC procs only
(qrun)	qrun	display servicable stream queues
(Q)	Queue	display allocated stream queues
	-e	queue modifier (all queues)
	values	queue identifier (give slot # or addr)
(strm)	stream	display allocated streams
	-e	stream modifier (all queues)
	-f	stream modifier (full display)
	value(s)	stream identifier (give slot # or addr)
(strst)	strstat	display streams statistics
(set)	set var=expr	set variable equal to expression value
(s)	stack	kernel stack trace (all procs)
	-r	kernel stack modifier (display registers)
	-f	kernel stack modifier (use kfp value)
	-e *value*	kernel stack trace (give engine #)
	value	kernel stack trace (give proc #)
(S)	session	session information
	value(s)	session information (give proc #)
(t)	trace	turn on debug tracing
(upte)	upte	dump user pte's (give proc #)
(u)	user	display user area (give proc #)
(vnode)	vnode	dump a vnode table (sortof)
(vtop)	vtop	virtual to physical
	!	escape to shell
	!!	repeat last command
	.*<file>*	source a file
(?)	?	print this list of available commands
(h)	help	print this list of available commands
(w)	what	interpret addresses, symbols or expressions

FILES

/dev/kmem	default system image
/unix	default system namelist

WARNINGS

This command is intended for system debugging and is not supported software. It is not compatible with the AT&T System V **crash** command.

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 examines *crontab* files and **at** command files only 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)

DIAGNOSTICS

A history of all actions taken by **cron** are recorded in */usr/lib/cron/log*.

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' *crontab* files. The **–r** option removes a user's *crontab* from the crontab directory. **crontab –l** lists 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 fields 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 can 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 hyphen (meaning an inclusive range). Note that the specification of days can 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 newline 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/lbin).

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	Main cron directory
/usr/spool/cron/crontabs	Spool area
/usr/lib/cron/log	Accounting information
/usr/lib/cron/cron.allow	List of allowed users
/usr/lib/cron/cron.deny	List of denied users

SEE ALSO

cron(1M), sh(1)

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 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. 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, and "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 (that is, to take a substantial fraction of a second to compute). However, if keys are restricted to (say) three lowercase 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)

WARNINGS

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.

NOTES

This command is available only in the United States.

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**), and a C-like syntax.

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. Login shells will also source commands from the file **/etc/csh.login** *if this file exists.*

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 these exceptions. The following characters form separate words:

& | ; < > ( )

If doubled as shown below, these pairs form single words.

&& << >>

These parser metacharacters can be made part of other words, or their special meaning can be removed by preceding them with a backslash (\). A newline preceded by a backslash (\) is equivalent to a blank.

In addition, strings enclosed in matched pairs of quotations (such as ', `, 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 single or double quotes (' or "), a newline preceded by a backslash (\) gives a true newline character.

If the shell input is not a terminal, the pound character (#) introduces a comment that continues to the end of the input line. It is prevented this special meaning when preceded by a backslash (\) or enclosed in quotes (`, ', or ").

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 the pipe character (|) forms a pipeline. The output of each command in a pipeline is connected to the input of the next. Sequences of pipelines can be separated by a semicolon (;) and are then executed sequentially. A sequence of pipelines can be executed without immediately waiting for it to

terminate by following it with an ampersand (&).

Any of the above can be placed in parentheses ( ) 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 that looks like this:

```
[1] 1234
```

The message indicates the job started asynchronously was job number 1 and had one (top-level) process, whose process ID was `1234`.

If you are running a job and want to do something else, you can suspend the job, typically by pressing Ctrl-Z, which sends a STOP signal to the current job. The shell then normally indicates that the job has been `Stopped` and prints 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 Ctrl-Z takes effect immediately and is like an interrupt in that pending output and unread input are discarded when it is typed.

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 can 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.

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 an exclamation mark (!) and can begin anywhere in the input stream (with the proviso that they do not nest.) The exclamation mark (!) can be preceded by a backslash (\) to prevent its special meaning. For convenience, an exclamation mark (!) is passed unchanged when it is followed by a blank, tab, newline, equal sign (=), or left parenthesis. (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:

```
09  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 exclamation mark (!) in the prompt string.

With the current event 13 we can refer to previous events by event number **!11**, relatively as in **!-2** (referring to the same event), by a prefix of a command word as in **!d** for event 12 or **!wri** for event 9, or by a string contained in a word in the command as in **!?mic?**, which also refers 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 colon (:) and a designator for the desired words. The words of an input line are numbered from 0, the first (usually command) word being 0, the second word (first

argument) being 1, etc. The basic word designators are as follows:

0	First (command) word
n	*n*'th argument
↑	First argument, i.e. '1'
$	Last argument
%	Word matched by (immediately preceding) ?*s*? search
x–*y*	Range of words
–*y*	Abbreviates 0–*y*
*	Abbreviates ↑–$, or nothing if only 1 word in event
*x**	Abbreviates *x*–$
x–	Like *x** but omitting word $

The colon (:) 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 colon (:). 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, for example **g&**.
p	Print the new command line but do not execute it.
q	Quote the substituted words, preventing further substitutions.
x	Like q, but break into words at blanks, tabs and newlines.

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 can be used as the delimiter in place of a slash (/); a backslash (\) 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 backslash (\) quotes **&** also. A null *l* uses the previous string either from a *l* or from a contextual scan string *s* in **!?***s***?**. The trailing delimiter in the substitution may be omitted if a newline follows immediately as may the trailing **?** in a contextual scan.

A history reference can be given without an event specification, for example **!$**. In this case the reference is to the previous command unless a previous history reference occurred on the same line in which case this form repeats the previous reference. Thus **!?foo?↑ !$** 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 **↑lb↑lib** fixes the spelling of **lib** in the previous command. Finally, a history substitution can be surrounded with brackets ({ and }) if necessary to insulate it from the characters which follow. Thus, after **ls –ld ~paul**, we might type **!{l}a** to perform **ls –ld ~paula**, while **!la** 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 single quotes (´) are prevented any further interpretation. Strings enclosed in double quotes (") can 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 Substitition** below) does a double-quoted string yield parts of more than one word; single-quoted strings never do.

Alias substitution

The shell maintains a list of aliases that 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 \!* | 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. Many of the variables referred to by the shell are toggles; the shell does not care what their value is, only whether they are set. For instance, the **verbose** variable is a toggle that 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 keyed by **$** characters. This expansion can be prevented by preceding the dollar sign ($) with a backslash (\), except within double quotes ("), where it always occurs, and within single

quotes (‘), where it never occurs. Strings quoted by backquotes (`) are interpreted later (see **Command substitution** below) so $ substitution does not occur there until later, if at all. A dollar sign ($) is passed unchanged if followed by a blank, tab, or end-of-line.

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

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

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

$name
${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).

$name[selector]
${name[selector]}
: May be used to select only some of the words from the value of *name*. The selector is subjected to $ substitution and may consist of a single number or two numbers separated by a hyphen (–). 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 `$#`*name*. The asterisk selector (*) selects all words. It is not an error for a range to be empty if the second argument is omitted or in range.

$#name
${#name}
: Gives the number of words in the variable. This is useful for later use in a [selector].

$0
: Substitutes the name of the file from which command input is being read. An error occurs if the name is not known.

$number
${number}
: Equivalent to $argv[number].

$*

Equivalent to $argv[*].

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

The following substitutions may not be modified with colon (:) modifiers:

$?name
${?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.

$$

Substitutes 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.

Command and filename substitution

The remaining substitutions, command and filename substitution, are applied selectively to the arguments of builtin 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.

Command substitution

Command substitution is indicated by a command enclosed in backquotes (`). 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 double quotes, 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 is replaced with an alphabetically sorted list of filenames that match the pattern. In a list of words specifying filename substitution, it is an error for no pattern to match an existing filename, but it is not required for each pattern to match. Only the metacharacters *, ? and [imply pattern matching. The characters ~ and { are more akin to abbreviations.

In matching filenames, a dot (.) character at the beginning of a filename or immediately following a slash (/), as well as the /, must be matched explicitly. The asterisk (*) matches any string of characters, including the null string.

The question mark (?) matches any single character. The sequence [...] matches any one of the characters enclosed. Within [...], a pair of characters separated by a hyphen (–) matches any character lexically between the two.

The tilde (~) at the beginning of a filename is used to refer to home directories. Standing alone, (that is, ~), it expands to the invoker's home directory as reflected in the value of the variable **home.** When followed by a name consisting of letters, digits and hyphens, 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 tilde (~) is followed by a character other than a letter or a slash (/) or appears not at the beginning of a word, it is left undisturbed.

The metanotation **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 can be nested. Thus, *~source/s1/{oldls,ls}.c* expands to */usr/source/s1/oldls.c /usr/source/s1/ls.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
>! name
>& name
>&! name

The file *name* is used as standard output. If the file doesn't exist, it is created; if the file exists, it is truncated and its previous contents are lost.

If the variable **noclobber** is set, then the file must not exist or be a character special file (such as a terminal or */dev/null*) or an error results. This helps prevent accidental destruction of files. In this case, the exclamation mark (!) 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.

>> name
>>& name
>>! name
>>&! name
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 exclamation mark (!) 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 | & rather than just |.

Expressions

A number of the built-in 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 % being 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, for example, *s, ?s and instances of [...]) against which the left-hand operand is matched. This reduces the need to use the *switch* statement in shell scripts when all that is really needed is pattern matching.

Strings beginning with zero (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 (& | < > ( )), they should be surrounded by spaces.

Also available in expressions as primitive operands are command executions enclosed in braces ({ and }) and file enquiries of the form *-l name* where *l* is one of the following:

r	Read access
w	Write access
x	Execute access
e	Existence
o	Ownership
z	Zero size
f	Plain file
d	Directory

The specified name is command- and 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 (that is, 0). Command executions succeed, returning true (that is, 1), if the command exits with status 0, otherwise they fail, returning false (that is, 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.

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 **cdpath** 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.

exec 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 executed. (Both *foreach* and *end* must appear alone on separate lines.)

The built-in command **continue** may be used to continue the loop prematurely and the builtin command **break** to terminate it prematurely. When this command 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 backslash (\) 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 continues after the specified line.

hashstat
Print a statistics line indicating how effective the internal hash table has been at locating commands (and avoiding **exec**'s). An **exec** is attempted for each component of the **path** where the hash function indicates a possible hit, and in each component which does not begin with a slash (/).

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 sourceing using the –h option to **source**.

if (expr) command
If the specified expression evaluates true, then the single *command* with arguments 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** executed (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/sys/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*

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.

The only controllable resource is *filesize* (the largest single file that can be created). Only the superuser may increase this limit.

The *maximum-use* may be given as a (floating-point or integer) number followed by a scale factor. The default scale is **k** or kilobytes (1024 bytes); a scale factor of **m** or megabytes may also 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.

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 variables TERM, and PATH) are automatically imported to and exported from the **csh** variables **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 can 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 beginning of a line. The command **breaksw** causes execution to continue after the **endsw.** Otherwise control may fall through case labels and default labels as in C. If no label matches and there is no default, execution continues after the **endsw.**

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 disabled.

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 superuser 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 **env**(1).

wait
All background jobs are waited for. It 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 *=, +=, and so on 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 postfix ++ and -- operators increment and decrement *name* respectively, that is, @ 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 **TERM** into the variable **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 **csh** 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, (for example, **$1** is replaced by **$argv[1]** and so on).

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-builtin commands all expansions occur before echoing. Builtin commands are echoed before command and filename substitution, since these substitutions are then done selectively.

filec Enable filename completion.

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 Ctrl-D.

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. If the file exists with an access time not greater than its modify time, the shell says `You have new mail.`

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 (that is, **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, only full pathnames will execute. The usual search path is ., */bin*, and */usr/bin*, but this may vary from system to system. For the superuser, 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* 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 an exclamation mark (!) appears in the string, it will be replaced by the current event number unless a preceding backslash (\) is given. Default is % for regular users; # for the superuser.

savehist This variable 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 number of events is saved. During start up, the shell sources ~/*.history* into the history list, enabling history to be saved across logins. Too large values of **savehist** 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 that 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 builtin 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 slash (/), the shell concatenates with the given command name to form a pathname 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 pathname of the shell (such as *$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 a hyphen (–), 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 backslash (\) can 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 #, (that is, 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, including hangups. Other signals have the values which the shell inherited from its parent. The shells handling of interrupts and terminate signals in shell scripts can be controlled by **onintr.** Login shells catch the *terminate* signal; otherwise this signal is passed on to children from the state in the shell's parent. In no case are 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, Laxenburg, Austria, with different syntax than that used now.

FILES

~/.cshrc	Read at beginning of execution by each shell
~/.login	Read by login shell, after *.cshrc* at login
~/.logout	Read by login shell, at logout
/etc/csh.login	Read by login shell before files in home directory.
/bin/sh	Standard shell, for shell scripts not starting with a #
/tmp/sh*	Temporary file for <<
/etc/passwd	Source of home directories for ~name

NOTES

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(1), access(2), execve(2), fork(2), kill(2), pipe(2), signal(2), ulimit(2), umask(2), wait(2), termio(7), a.out(r), environ(5)

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 (that is, 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 immediately executes **c**. This is especially noticeable if this expansion results from an **alias**. It suffices to place the sequence of commands in parentheses to force it to a subshell, such as **(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 procedures 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.

The way the **filec** facility is implemented is ugly and expensive.

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 . . . xx*n* (*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 hyphen (–), 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 *prefix***00** . . . *prefix*x*n*. The default is **xx00** . . . **xx***n*.

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 (for example, **/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 arguments. 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 newlines. **csplit** does not affect the original file; it is the users responsibility to remove it.

EXAMPLES

This example creates four files, *cobol00 . . . cobol03*:

csplit -f cobol *file* **'/procedure division/' /par5./ /par16./**

After editing the "split" files, they can be recombined as follows. Note that this example overwrites the original file.

cat cobol0[0–3] > *file*

The next example splits 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 print.

csplit -k *file* **100 {99}**

Assuming that *prog.c* follows the normal C coding convention of ending routines with a } at the beginning of the line, the next example creates a file containing each separate C routine (up to 21) in *prog.c*.

csplit -k prog.c '%main(%' '/^}/+1' {20}

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

ct – spawn getty to a remote terminal

SYNOPSIS

ct [**-wn**] [**-xn**] [**-h**] [**-v**] [**-s**speed] telno ...

DESCRIPTION

ct 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, **ct** will try each in succession until one answers. This is useful for specifying alternate dialing paths.

ct will try each line listed in the file */usr/lib/uucp/Devices* until it either finds an available line with appropriate attributes or runs out of entries. If there are no free lines, **ct** will ask if it should wait for one, and if so, the number of minutes it should wait before it gives up. **ct** 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 **-w***n* option, where *n* is the maximum number of minutes that **ct** is to wait for a line.

The **-x***n* option is used for debugging; it produces a detailed output of the program execution on *stderr*. The debugging level, *n*, is a single digit; **-x**9 is the most useful value.

Normally, **ct** 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 **ct** process before returning control to the user's terminal. If the **-v** option is used, **ct** will send a running narrative to the standard error output stream.

The data rate can 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 can occur depending on the type of getty on the line (**getty** or **uugetty**). For the first case, **ct** prompts `Reconnect?` If the response begins with the letter **n**, the line will be dropped; otherwise, **getty** will start again and the `login:` prompt will print. 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 Ctrl-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), login(1), uucp(1C), getty(1M), 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 [**–auwxm**] name ...

DESCRIPTION

ctags makes a tags file for **ex**(1) from the specified C sources. A tags file gives the locations of specified objects (in this case functions) in a group of files. Each line of the tags file contains the object name, the file in which it is defined, and an address specification for the object definition. Functions are searched with a pattern. Specifiers are given in separate fields on the line, separated by blanks or tabs. Using the *tags* file, **ex** can quickly find these objects definitions.

If the **–x** flag is given, **ctags** produces a list of object names, the line number and filename 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 that can be printed out as an off-line readable function index.

Files whose names end in *.c* or *.h* are assumed to be C source files and are searched for C routine and macro definitions.

Other options are as follows:

–a Append to tags file.

–w Suppress warning diagnostics.

–u Cause 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.)

–m Create multiple entries for tags that occur more than once.

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 using **ctags** practical in directories with more than one program.

FILES

tags	Output tags file

SEE ALSO

ex(1), vi(1)

AUTHOR

ctags was developed by The University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

BUGS

Does not know about *#ifdefs*.

NAME

cu – call another UNIX system

SYNOPSIS

cu [**–s**speed] [**–l**line] [**–h**] [**–t**] [**–d**] [**–o** | **–e**] [**–n**] telno
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.

–lline 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 **–l** option is used without the **–s** option, the speed of a line is taken from the *Devices* file. When the **–l** and **–s** options are 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 print and the call will not be made. The specified device is generally a directly-connected asynchronous line (for example, */dev/ttyGA/GAAB*), 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 can be used rather than a telephone number. In this case, **cu** will obtain an appropriate direct line or telephone number from */usr/lib/uucp/Systems.*

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 system. 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 the ~**%take** and the ~**%put** command, as each block of the file is transferred, consecutive single digits are printed to the terminal.
~^Z	Stop cu (available only with job control).
~~ *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 if the remote system 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 log into a system connected by a direct line, use either of these commands:

cu –l */dev/ttyGA/GAAE*

or

cu –l *ttyGA/GAAE*

To dial a system with a specific line and a specific speed:

cu –s1200 –l *ttyGA/GAAE*

To dial a system using a specific line associated with an auto dialer:

cu –l *ttyGA/GAAE* **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 interconnection 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.

NAME

cut – cut out selected fields of each line of a file

SYNOPSIS

cut –c*list* [file ...]
cut –f*list* [–d*char*] [–s] [file ...]

DESCRIPTION

Use **cut** to cut out columns from a table or fields from each line of a file. In database parlance, **cut** implements the projection of a relation. The fields as specified by *list* can be fixed length, such as character positions on a punched card (–c option), or the length can vary from line to line and be marked with a field delimiter character like *tab* (–f option). **cut** can be used as a filter; if no files are given, the standard input is used. In addition, a filename of "–" explicitly refers to standard input.

The meanings of the options are as follows:

list A comma-separated list of integer field numbers (in increasing order), with optional hyphens (–) to indicate ranges [for example, **1,4,7**; **1–3,8**; **–5,10** (short for **1–5,10**); or **3–** (short for third through last field)].

–c*list* The *list* following –c (no space) specifies character positions. For example, **–c1–72** would pass the first 72 characters of each line.

–f*list* The *list* following –f is a list of fields assumed to be separated in the file by a delimiter character (see **–d**); for example, **–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.

–d*char* The character following –d is the field delimiter (–f option only). Default is *tab*. Space or other characters with special meaning to the shell must be quoted.

–s 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 **grep**(1) to make horizontal "cuts" (by context) through a file, or **paste**(1) to put files together column-wise (that is, horizontally). To reorder columns in a table, use **cut** and **paste**.

EXAMPLES

cut –d: –f1,5 /etc/passwd	Mapping of user IDs to names
name=`who am i | cut –f1 –d" "`	Set **name** to current login name

DIAGNOSTICS

ERROR: line too long
Either a line has more than 1023 characters or fields, or there is no new-line character.

ERROR: bad list for c / f option
Missing –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
The *list* is empty. .

ERROR: no delimeter
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 it 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 [+format]
date [mmddhhmm[[yy] | [ccyy]]]

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 (only the superuser can set the date). 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, *cc* is the century minus one and is optional, *yy* is the last 2 digits of the year number and is optional. For example, the command **date 10080045** sets the date to Oct 8, 12:45 AM.

If no year is mentioned, the current year is the default. The system operates in UTC (Universal Time Code, formerly Greenwich Mean Time). **date** takes care of the conversion to and from local standard and daylight time. Only the superuser can change the date. When the date and time are set successfully, **date** displays the new date according to the format defined in the environment variable **CFTIME** (see **environ**(5)).

Specifications of native language translations of month and weekday names are supported. The language used depends on the value of the environment variable **LANGUAGE** (see **environ**(5)). The month and weekday names used for a language are taken from strings in the file for that language in the */lib/cftime* directory (see **cftime**(4)).

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 newline character. If the argument contains embedded blanks, it must be quoted (see the **EXAMPLE** section). The Field Descriptors (which must be preceded by a %) are as follows:

a Abbreviated weekday name

A Full weekday name

b Abbreviated month name

B Full month name

d Day of month – 01 to 31

D Date as mm/dd/yy

e Day of month – 1 to 31 (single digits are preceded by a blank)

h Abbreviated month name (alias for %b)

H Hour – 00 to 23

I Hour – 01 to 12

j Day of year – 001 to 366

m Month of year – 01 to 12

M Minute – 00 to 59

n Insert a new-line character

p String containing ante-meridiem or post-meridiem indicator (by default, AM or PM)

r Time as *hh:mm:ss pp* where *pp* is the ante-meridiem or post-meridiem indicator (by default, AM or PM)

R Time as hh:mm

S Second – 00 to 59

t Insert a tab character

T Time as *hh:mm:ss*

U Week number of year (Sunday as the first day of the week) – 01 to 52

w Day of week – Sunday = 0

W Week number of year (Monday as the first day of the week) – 01 to 52

x Country-specific date format

X Country-specific time format

y Year within century – 00 to 99

Y Year as *ccyy* (4 digits)

Z Timezone name

EXAMPLES

```
date '+DATE: %m/%d/%y%nTIME: %H:%M:%S'
```

The command shown above generates the following output:

```
DATE: 08/01/76
TIME: 14:45:05
```

FILES

/dev/kmem

SEE ALSO

sysadm(1), cftime(4), environ(5)

DIAGNOSTICS

No permission You are not the superuser and you tried to change the date

bad conversion The date set is syntactically incorrect

WARNINGS

Should you need to change the date while the system is running multiuser, use the command **sysadm datetime**, or **menu cladmsetup**.

NOTES

If you attempt to set the current date to a date that the standard and alternate time zones change (such as the date that daylight time starts or ends) and you attempt to set the time to a time in the interval between the end of standard time and the beginning of the alternate time (or the end of the alternate time and the beginning of standard time), the results are unpredictable.

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*x* 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*x* 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 an operating 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.

EXAMPLES

This example prints the first ten values of n!:

```
[la1+dsa*pla10>y]sy
0sa1
lyx
```

SEE ALSO

bc(1)

DIAGNOSTICS

x is unimplemented	*x* is an octal number
stack empty	Not enough elements on the stack to do what was asked
Out of space	The free list is exhausted (too many digits)
Out of headers	Too many numbers being kept around
Out of pushdown	Too many items on the stack
Nesting Depth	Too many levels of nested execution

NAME

dcheck – filesystem directory consistency check

SYNOPSIS

/etc/dcheck [**–i** numbers] [filesystem]

DESCRIPTION

dcheck reads the directories in a filesystem and compares the link count in each inode with the number of directory entries by which it is referenced. If the filesystem is not specified, a set of default filesystems is checked.

The **–i** flag is followed by a list of inode numbers. When one of those inode numbers occurs in a directory, the number, the inode number of the directory, and the name of the entry are reported.

Because the inode list is read in large chunks, the program is fastest if the raw version of the special file is used.

FILES

Default filesystems vary with installation.

NOTES

dcheck is obsoleted by **fsck**(1M) and remains for historical reasons.

SEE ALSO

clri(1M), fs(4), fsck(1M), icheck(1M), ncheck(1M)

DIAGNOSTICS

If the link count and the number of directory entries disagree in a file, the relevant facts are reported. Allocated files which have 0 link count and no entries are also listed. If there are more entries than links and if entries are removed dropping the link count to 0, the remaining entries point to thin air and should be removed. If there are more links than entries or there is an allocated file with neither links nor entries, some disk space may be lost.

BUGS

Since **dcheck** is inherently two-pass in nature, extraneous diagnostics may be produced if it is applied to active filesystems.

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.

option	*values*
if=*file*	Input filename; standard input is default
of=*file*	Output filename; standard output is default
ibs=*n*	Input block size *n* bytes (default 512)
obs=*n*	Output block size (default 512)
bs=*n*	Set both input and output block size, superseding *ibs* and *obs*; also, if no conversion is specified, it is particularly efficient since no in-core copy need be done
cbs=*n*	Conversion buffer size
skip=*n*	Skip *n* input blocks before starting copy
seek=*n*	Seek *n* blocks from beginning of output file before copying
count=*n*	Copy only *n* input blocks
conv=ascii	Convert EBCDIC to ASCII
ebcdic	Convert ASCII to EBCDIC
ibm	Slightly different map of ASCII to EBCDIC
lcase	Map alphabetics to lowercase
ucase	Map alphabetics to uppercase
swab	Swap every pair of bytes
noerror	Do not stop processing on an error
sync	Pad every input block to *ibs*
...,...	Several comma-separated conversions

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** or **conv=ebcdic** is specified. In the former case, **cbs** characters are placed into the conversion buffer (converted to ASCII). Trailing blanks are trimmed and a newline added before sending the line to the output. In the latter case, 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)

NAME

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 pathname 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.

–r*SID* 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 **get**s 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 it is 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 newline 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.)

-y*[comment]* Arbitrary text used to describe the reason for making the delta. A null string is considered a valid *comment*.

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 newline character terminates the comment text.

-p Causes **delta** to print (on the standard output) the SCCS file differences before and after the delta is applied in a **diff**(1) format.

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), bdiff(1), cdc(1), get(1), help(1) prs(1), rmdel(1), sccsfile(4)

DIAGNOSTICS

Use **help**(1) for explanations.

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 **sccsfile**(4)] and will cause an error.

A **get** of many SCCS files, followed by a **delta** of those files, should be avoided when the **get** generates a large amount of data. Instead, multiple **get/delta** sequences should be used.

If the standard input (-) is specified on the **delta** command line, the **-m** (if necessary) and **-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.

NAME

deroff – remove nroff, tbl, and eqn constructs

SYNOPSIS

deroff [**–mx**] [**–w**] [files]

DESCRIPTION

deroff reads each of the *files* in sequence and removes all **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** commands). If a file has already been included, an **.so** naming that file is ignored. 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 (that is, 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) in the *DOCUMENTER'S WORKBENCH Technical Discussion and Reference Manual*

BUGS

The **–ml** option does not handle nested lists correctly.

NAME

devbuild, devdestroy – build and destroy virtual devices

SYNOPSIS

/etc/devbuild *devtype special options*
/etc/devbuild [–a]
/etc/devdestroy *special*
/etc/devdestroy [–a]

DESCRIPTION

With the exception of the disk containing the **root** (/) filesystem, partitioned disks and other virtual devices are considered destroyable in the sense that they can be either available to users or dismantled. **devbuild** announces to the system that *special*, a character special device, is available to users as a virtual device of type *devtype*. The *devtype* must be one supported by the system. The device *special* becomes the parent device of a set of other devices. If *devtype* is a partitioning driver, then unpartitioned disk *special* will become partitioned.

devbuild adds entries to the table of constructed devices, */etc/constab*. **devdestroy** removes entries. If invoked with no arguments, **devbuild** prints the entire table of "built" devices.

The following option is available:

–a Indicates that **devbuild** should search through */etc/devtab* for virtual device relationships to build, and attempt to build them.

devdestroy announces to the system that the previously created *special* is to be made unavailable. If *special* is a partitioned disk, then the partitioning is dismantled and *special* becomes available for use again.

devbuild can be run by any user to list previously built devices. Only a superuser can build and destroy devices.

EXAMPLES

In order to put compatibility partitions on **sd0**, which is a CDC Wren 3, you would type this command:

devbuild cmpt /dev/rdsk/sd0 wren3

This makes the special file */dev/rdsk/sd0* unusable (it cannot be opened), and makes the partitions */dev/rdsk/sd0s0* through */dev/rdsk/sd0s7* available to be mounted, swapped on, or otherwise used. To remove compatibility partitioning, you would type this command:

devdestroy /dev/rdsk/sd0

This makes the various partitions unavailable (subsequent attempts to open these devices will fail) and makes */dev/rdsk/sd0* available again. If any of the partitions are mounted filesystems or are being used as swap devices by the system, the **devdestroy** command will fail.

If you want to establish VTOC partitioning on this same disk, you could type this command:

devbuild vtoc /dev/rdsk/sd0

Note that because partitioning information is read from the VTOC (from the disk itself), a disk type is unnecessary.

FILES

/etc/constab	Table of build devices
/etc/devtab	Virtual device table

SEE ALSO

mkvtoc(1M), newfs(1M), prtvtoc(1M), devtab(4), constab(4), vtoc(7), cmpt(7)

DIAGNOSTICS

If the **V_BUILD** ioctl fails, **devbuild** prints an appropriate diagnostic, and the line it was trying to interpret when it encountered the failure. If the **V_DESTROY** ioctl fails, **devdestroy** prints an appropriate diagnostic.

devdestroy fails if *special* has not been built or if it is busy. *special* is busy if it is mounted, being swapped on, or is open by some process. In such a case, you can use **mount**(1M) to list filesystems that are using *special* and use **swap**(1M) to list swap devices which may be on *special*.

NAME

devnm – device name

SYNOPSIS

/etc/devnm [names]

DESCRIPTION

devnm identifies the special file associated with the mounted filesystem where the argument *name* resides.

This command is most commonly used by **/etc/brc** (see **brc**(1M)) to construct a mount table entry for the **root** device.

EXAMPLES

If */usr* is mounted on */dev/dsk/zd0s2*, the command:

/etc/devnm /usr

produces this output:

```
/dev/dsk/zd0s2 /usr
```

FILES

/dev/dsk/*
/etc/mnttab

SEE ALSO

brc(1M)

NAME

df – report number of free disk blocks and inodes

SYNOPSIS

df [–ltS] [–f] [*filesystem* | *directory*]

DESCRIPTION

The **df** command prints out the number of free blocks and free inodes in mounted filesystems or directories by examining the counts kept in the superblocks.

filesystem may be specified either by device name (such as */dev/dsk/zd0s2*) or by mountpoint directory name (such as */usr*).

directory can be a directory name. The report presents information for the device that contains the directory.

If no arguments are used, the free space on all locally mounted filesystems is printed.

If a -1 appears in a given field then that information is not available for that filesystem.

The **df** command uses the following options:

–l Reports only on local filesystems.

–t Causes the figures for total allocated blocks and inodes to be reported as well as the free blocks and inodes.

–f An actual count of the blocks in the free list is made, rather than taking the figure from the superblock (free inodes are not reported).

–S Reports the amount of disk space available for superusers.

FILES

/dev/dsk/*
/etc/mnttab

SEE ALSO

mount(1M), fs(4), mnttab(4)

NAME

diff – differential file comparison

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 in 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.

Missing newline at end of file X
Indicates that the last line of file X did not have a newline. If the lines are different, they will be flagged and output; although the output will seem to indicate they are the same.

BUGS

Editing scripts produced under the **–e** or **–f** option are naive about creating lines consisting of a single period (.).

NAME

diff3 – 3-way differential file comparison

SYNOPSIS

diff3 [**–ex3**] 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* = *n*2, 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 are 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**] [**–w***n*] 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 filenames 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.

–w*n* 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. Because **diskusg** reads the superblock of a filesystem, it is limited to only filesystems of type "4.2". **diskusg** outputs 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

#blocks The total number of disk blocks allocated to this user

diskusg normally reads only the inodes of filesystems 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.

–i *fnmlist* Ignore the data on those filesystems whose filesystem name is in *fnmlist*, which is a list of filesystem names separated by commas or enclosed within quotes. **diskusg** compares each name in this list with the filesystem 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 filename, 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/zd0s2*:

```
diskusg /dev/dsk/zd0s2 | acctdisk > disktacct
```

FILES

/etc/passwd Used for user ID to login name conversions

SEE ALSO

acct(1M), acctsh(1M), acct(4)

NAME

dltsectbl – refresh security table

SYNOPSIS

dltsectbl

DESCRIPTION

The Refresh Security Table command (**dltsectbl**) removes the in-memory version of the security (task) table. The next time a program in the menu system attempts to use the table, the system will generate a new table from the task file.

To maintain the */etc/passwd*, */etc/group*, and *${TFADMFILE}/etc/task* files, use the forms on the User and Group Management and the Menu Task Management menus. If you edit any of these files manually, always run the **dltsectbl** command to ensure that the in-memory table reflects the new relationships between users, groups, and tasks.

FILES

${TFADMFILE}/etc/task	System global security table
/etc/passwd	System password file
/etc/group	System group file

NAME

dmesg – collect system diagnostic messages to form error log

SYNOPSIS

/etc/dmesg [–]

DESCRIPTION

dmesg looks in a system buffer for recently printed diagnostic messages and prints them on the standard output. The messages are those printed by the system when device (hardware) errors occur and (occasionally) when system tables overflow non-fatally. If the – flag is given, then **dmesg** computes (incrementally) the new messages since the last time it was run and places these on the standard output. This is typically used with **cron**(1M) to produce the error log */usr/adm/messages* by running the following command every 10 minutes.

/etc/dmesg – >> /usr/adm/messages

FILES

/usr/adm/messages	Error log (conventional location)
/usr/adm/msgbuf	Scratch file for memory of – option

BUGS

The system error message buffer is of small finite size. As **dmesg** is run only every few minutes, not all error messages are guaranteed to be logged. This can be construed as a blessing rather than a curse.

Error diagnostics generated immediately before a system crash will never get logged.

NAME

du – summarize disk usage

SYNOPSIS

du [**–sar**] [names]

DESCRIPTION

du reports the number of 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 Causes **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 filesystem hierarchy, **du** will count the excess files more than once.

Files with holes in them will get an incorrect block count.

NAME

dump – incremental filesystem dump

SYNOPSIS

/etc/dump [key [*argument* ...] filesystem]

DESCRIPTION

dump copies to magnetic tape all files changed after a certain date in the *filesystem*. The *key* specifies the date and other options about the dump. *key* consists of characters from the set **0123456789fusdWn.**

0–9 This number is the *dump level*. The command dumps all files modified at lesser levels since the last date stored in the file */etc/dumpdates* for the same filesystem. If no date is determined by the level, the beginning of time is assumed; thus the option **0** Causes the entire filesystem to be dumped.

f Places the dump on the next *file* named as an argument in the command line instead of on the tape. If *file* is "–", *dump* writes to standard output.

u If the dump completes successfully, writes the date of the beginning of the dump on the */etc/dumpdates* file. This file records a separate date for each filesystem and each dump level. The format of */etc/dumpdates* is readable by people and consists of one free format record per line: filesystem name, increment level and **ctime**(3) format dump date. */etc/dumpdates* can be edited to change any of the fields, if necessary.

s Specifies size of the dump tape in feet. The number of feet is specified by *tapelength*. When the specified size is reached, **dump** waits for reels to be changed. The default tape size is 2300 feet.

d Specifies the tape density in BPI, as specified in *density*. This is used in calculating the amount of tape used per reel. The default is 1600.

W **dump** tells the operator what filesystems need to be dumped. This information is gleaned from the files */etc/dumpdates* and */etc/fstab*. The **W** option causes **dump** to print out, for each filesystem in */etc/dumpdates*, the most recent dump date and level, and to highlight those filesystems that should be dumped. If the **W** option is set, all other options are ignored and **dump** exits immediately.

w Like **W**, but prints only those filesystems that need to be dumped.

n Whenever **dump** requires operator attention, notify, by means similar to a **wall**(1), all operators in the group *adm*.

If no arguments are given, the *key* is assumed to be **9u** and a default filesystem is dumped to the default tape.

dump requires operator intervention on these conditions: end of tape, end of dump, tape write error, tape open error, or disk read error (if there are more than a threshold of 32). In addition to alerting all operators (implied by the **n** key), **dump** interacts with the operator on **dump**'s control terminal at times when **dump** can longer proceed or if something is grossly wrong. All questions **dump** poses **must** be answered by typing **yes** or **no** as appropriate.

Since making a dump involves a lot of time and effort for full dumps, **dump** checkpoints itself at the start of each tape volume. If writing that volume fails for some reason, **dump** , with operator permission, restarts itself from the

checkpoint after the old tape has been rewound and removed and a new tape has been mounted.

dump tells the operator what is going on at periodic intervals, including usually low estimates of the number of blocks to write, the number of tapes it will take, the time to completion, and the time until the tape change. The output is verbose, so that others know that the terminal controlling **dump** is busy and will be so for some time.

The following discussion suggests how to perform dumps. Start with a full level 0 dump:

dump 0un

Next, dumps of active filesystems are taken on a daily basis, using a modified Tower of Hanoi algorithm, with this sequence of dump levels:

3 2 5 4 7 6 9 8 9 ...

For the daily dumps, a set of 10 tapes per dumped filesystem is used on a cyclical basis. Each week, a level 1 dump is taken, and the daily Hanoi sequence repeats with 3. For weekly dumps, a set of 5 tapes per dumped filesystem is used, also on a cyclical basis. Each month, a level 0 dump is taken on a set of fresh tapes that is saved forever.

EXAMPLES

In the following example, a complete image of the */usr* filesystem is copied to a tape:

```
dump 0f /dev/rmt/xt0 /usr
```

FILES

/dev/rdsk/zd1s3	Default filesystem to dump from
/dev/rmt/xt0h	Default tape unit to dump to
/etc/dumpdates	New format dump date record
/etc/fstab	Dump table: filesystems
/etc/group	To find group *operator*

SEE ALSO

tar(1), cpio(1), restore(1M), volcopy(1M), dump(4), fstab(4)

DIAGNOSTICS

Many, and verbose.

dump exits with zero status on success. Startup errors are indicated with an exit code of 1; abnormal termination is indicated with an exit code of 3.

WARNINGS

This command is different from **/bin/dump**.

BUGS

Fewer than 32 read errors on the filesystem are ignored. Each reel requires a new process, so parent processes for reels already written just hang around until the entire tape is written.

dump with the **W** or **w** options does not report filesystems that have never been recorded in */etc/dumpdates*, even if listed in */etc/fstab*.

The dump frequency parameter is not available in */etc/fstab* so the dump frequency is always assumed to be one day when the **W** or **w** options are used.

It would be nice if **dump** knew about the dump sequence, kept track of the tapes scribbled on, told the operator which tape to mount when, and provided more assistance for the operator running **restore**.

Dump tapes contain the time of the last dump taken from the file */etc/dumpdates*. This date is stored in "local time" format and is converted to GMT for storage on the dump tape. The offset from GMT as recorded in the file */etc/TIMEZONE* must remain constant over a series of dump levels. Should it change, the message "Incremental tape too high" or "Incremental tape to low" may be seen during a restore operation, and the operation will fail.

NAME

dumpconf – dump kernel auto-configuration device table

SYNOPSIS

/etc/dumpconf [**–adkps**] [**–c** *core*] [**–u** *unix*]

DESCRIPTION

The **dumpconf** command can be used to examine the configuration of the present system or the configuration of a system that caused a panic. This command also provides information that can be used if you want to reduce the size of the kernel or to bind kernel logical names to specific hardware devices.

The **dumpconf** command has the following options:

–a Display all devices (default)

–c Specify an alternate core file to be used instead of */dev/kmem*

–d Display only physical devices

–k Display output in a format suitable for a kernel configuration file

–p Display only pseudo-devices

–s Display device-specific space definitions

–u Specify an alternate kernel file to be used instead of */unix*

At system boot time, the kernel auto-configuration process will search for available hardware based on the presence of configurable device drivers in the kernel. If hardware is present on the system but the device driver for that hardware does not exist in the kernel, the device will not be probed (and will not be found).

The auto-configuration process obtains the following information from the system configuration file (the default file is *system.std*):

- Which device drivers should be configured
- The number of devices of each type that should be searched for (there is a separate line for each device)
- Which bus a board is connected to
- Unit numbers (these numbers are an index into a table of possible addresses)

You can modify the information in your system configuration file if necessary. For example, you can remove entries for hardware that does not exist on your system, and you can specify the unit numbers associated with each device (this is called *binding* devices). After editing the file, you must rebuild the kernel.

Device unit numbers and bus numbers can be *wildcarded* in the system configuration file by using a '?' instead of an actual number. When the system is booted, logical device numbers are assigned to each device as it is found. (For example, zd0 is the first zd disk found; zd1 is the second.) Although wildcards allow hardware to be found at any address, they can cause confusion. For example, your system might include four zd disks that will be assigned a wildcard address. At powerup, all hardware will be probed and the logical numbers zd0, zd1, zd2 and zd3 will be assigned to the disks.

You might store user accounts on zd2 and your database on zd3. If zd2 developed a problem and could not be found during a subsequent boot process, the disk containing your database would become logical disk zd2. This situation can be avoided by binding the hardware to specific unit numbers and buses. To do this, replace the wildcard (?) with the appropriate unit number (and bus number, where applicable). You can use the output from the **dumpconf** command to determine the unit numbers associated with each device.

The output from **dumpconf** consists of comment lines (indicated by a # character) that describe the devices configured and the devices actually found on the system. All remaining lines describe the bindings for the available hardware or pseudo-devices. Lines beginning with 'pseudo-device' indicate that a pseudo-device driver is present. Some of these lines end with a number that is used to size various kernel resources. If the size field is empty for a particular pseudo-device, only one instance of that pseudo driver will be configured in the kernel.

EXAMPLES

The following examples were generated on an S27 system containing the following hardware:

- One MULTIBUS with one Systech terminal mux (st)
- One SCED (SCSI-Ether Controller) (sec)
- One DCC (zdc) disk controller
- One SMDE disk multiplexer (used with DCC)
- One SCSI disk (sd)
- One SCSI cartridge tape drive (ts)
- Two SMDE disks (zd)

The machine was booted using the standard kernel, which is configured to find more hardware than actually exists on this system. The following example shows output from the **/etc/dumpconf** command with no options.

```
$ /etc/dumpconf
# mbad devices 1 found 2 configured
mbad 0  on sb8000        unit 24 # slic 24
multibus 0       on mbad 0 unit ?
# sec devices 1 found 2 configured
sec 0   on sb8000        unit 22 # slic 22
sec_scsi 0       on sec 0 unit ?
# ssm devices 0 found 2 configured
# zdc devices 1 found 4 configured
zdc 0   on sb8000        unit 16 # slic 16
smde 0  on zdc 0 unit ?
# lp devices 0 found 1 configured
# st devices 1 found 4 configured
st 0    on multibus 0    unit 0  # configured on multibus 0 unit 0 using 32 maps
# xt devices 0 found 1 configured
# sd devices 1 found 4 configured
sd 0    on sec_scsi 0    unit 24 # configured on sec_scsi 0 unit 24
# wd devices 0 found 4 configured
```

```
# ts devices 1 found 1 configured
ts 0    on sec_scsi 0   unit 12 # configured on sec_scsi 0 unit 4
# tm devices 0 found 1 configured
# tg devices 0 found 1 configured
# xs devices 0 found 5 configured
# zd devices 2 found 32 configured
zd 0    on smde 0       unit 1  # configured on smde 0 unit 1
zd 1    on smde 0       unit 2  # configured on smde 0 unit 2
pseudo-device usclk     1
pseudo-device pmap      32
pseudo-device log       5
pseudo-device mem
pseudo-device kmmem
pseudo-device gentty
pseudo-device cmpt
pseudo-device vtoc
pseudo-device pty       100
pseudo-device strpipe   100
pseudo-device cn
pseudo-device clone
pseudo-device consmem
pseudo-device echo
pseudo-device ether
pseudo-device udp
pseudo-device ip
pseudo-device loop
pseudo-device tcpmux
```

The next example shows output from the **-k** option, which generates information that can be used for binding devices in the system configuration file. Notice that this listing omits the logical number (the number following st, sd, ts, and zd in the previous example). The system configuration file does not specify these logical numbers. Instead, the **config** program assigns logical numbers to devices in the order in which the devices are listed in the system configuration file.

```
$ /etc/dumpconf -dk
# mbad devices 1 found 2 configured
mbad 0  on sb8000       unit 24 # slic 24
multibus 0      on mbad 0 unit ?
# sec devices 1 found 2 configured
sec 0   on sb8000       unit 22 # slic 22
sec_scsi 0      on sec 0 unit ?
# ssm devices 0 found 2 configured
# zdc devices 1 found 4 configured
zdc 0   on sb8000       unit 16 # slic 16
smde 0  on zdc 0 unit ?
# lp devices 0 found 1 configured
# st devices 1 found 4 configured
st      on multibus 0   unit 0  # configured on multibus 0 unit 0 using 32 maps
# xt devices 0 found 1 configured
```

```
# sd devices 1 found 4 configured
sd      on sec_scsi 0   unit 24 # configured on sec_scsi 0 unit 24
# wd devices 0 found 4 configured
# ts devices 1 found 1 configured
ts      on sec_scsi 0   unit 12 # configured on sec_scsi 0 unit 4
# tm devices 0 found 1 configured
# tg devices 0 found 1 configured
# xs devices 0 found 5 configured
# zd devices 2 found 32 configured
zd      on smde 0       unit 1  # configured on smde 0 unit 1
zd      on smde 0       unit 2  # configured on smde 0 unit 2
```

FILES

/dev/kmem	default system image
/unix	default system namelist

SEE ALSO

config(1M), crash(1M), dump(8)

BUGS

There is no easy way to install the device information from **dumpconf -k** into a system configuration file. The information must be entered by hand using a text editor.

The **dumpconf** output does not indicate which device addresses were configured using a 'wildcard' identifier in the system configuration file.

NAME

dumpfs – dump filesystem information

SYNOPSIS

/etc/dumpfs *filesys* | *device*

DESCRIPTION

dumpfs prints out the superblock and cylinder group information for the filesystem or special device specified. The listing is very long and detailed. This command is useful mostly for finding out certain filesystem information such as the filesystem block size and minimum free space percentage.

SEE ALSO

tunefs(1M), fs(4)

NAME

ec – forms compiler

SYNOPSIS

ec [**–d**] [**–s**] [**–p** *path*] *srcfile(s)*

DESCRIPTION

The developer uses the **ec** form or command compiler to interface with the ptx/ADMIN menu system. EDC source files (also referred to as command description files) contain the specifications for a form. The command compiler checks the EDC source file for syntax errors. If the compiler finds any errors, the compiler displays an error message and the line number for the error. If there are no errors, the command writes the compiled command description file.

The compiled file is placed in the directory *${TFADMFILE}/edc*. If the **–p** option is used, the file is placed in the directory named by the user. The name of the compiled file matches the PROGNAME keyword in the EDC source file.

All EDC source files must have an **.e** or **.sb** suffix, similar to the **.c** suffix required by the C compiler.

–d Run the compiler in debug mode. This mode gives you additional information to help you locate syntax errors. Selection of debug mode is optional.

–s (Silent option.) No message appears when the compile is successful.

–p *destdir* Selects an alternate destination directory for the output of the compiler. If a directory is not identified, the compiled file is placed in the *${TFADMFILE}/edc* directory. Selection of an alternate destination directory is optional.

srcfile Name of the source file containing the command description used by the menu system. You can name one or more source files.

Environment Variables

TFADMFILE Identifies the directory where the menu system administration files are stored. This information is used to locate the *edc* subdirectory, the default directory for compiled files. This environment variable is defined in the Environment File *(/etc/tfenv)*.

FILES

${TFADMFILE}/edc/*command* Compiled command description file

SEE ALSO

prompt(1M)

NAME

echo – echo arguments

SYNOPSIS

echo [arg] ...

DESCRIPTION

echo writes its arguments, separated by blanks and terminated by a newline, 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 newline
\f	form-feed
\n	newline
\r	carriage return
\t	tab
\v	vertical tab
****	backslash
\0*n*	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)

NOTES

When representing an 8-bit character by using the escape convention **\0*n***, 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).

NAME

echo – put string on virtual output

SYNOPSIS

echo [string] . . .

DESCRIPTION

If no argument is given, **echo** looks to *stdin* for input. The **echo** command directs each string it is passed to *stdout*. It is often used in conditional execution or for passing a string to another command.

EXAMPLES

The following example validates field 1 as integer:

```
valid=`echo "$F1" | regex '^[0-9]+$'`
```

This example writes information to *stdout* when a user has completed using a form:

```
done=`echo "$LOGNAME is on-line" | message`
```

SEE ALSO

echo(1)

NAME

ed, red – text editor

SYNOPSIS

ed [**–s**] [**–p** string] [**–x**] [**–C**] [file]

red [**–s**] [**–p** string] [**–x**] [**–C**] [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, 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 **!***shell command*.

–p Allows the user to specify a prompt string.

–x Encryption option; when used, **ed** simulates an **X** command and prompts the user for a key. This key is used to encrypt and decrypt text using the algorithm of **crypt**(1). The **X** command makes an educated guess to determine whether text read in is encrypted. The temporary buffer file is encrypted also, using a transformed version of the key typed in for the **–x** option. See **crypt**(1). Also, see the **WARNINGS** section at the end of this manual page.

–C Encryption option; the same as the **–x** option, except that **ed** simulates a **C** command. The **C** command is like the **X** command, except that all text read in is assumed to have been encrypted.

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 allows editing of files in the current directory only and prohibits executing shell commands via **!***shell 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 either **stty –tabs** or **stty tab3** mode (see **stty**(1)), the specified tab stops will automatically be used when scanning *file*. For example, if the following line were the first line in a file, tab stops would be set at columns 5, 10, and 15, and a maximum line length of 72 would be imposed.

```
<:t5,10,15 s72:>
```

NOTE: when you are entering text into the file, this format is not in effect. Instead, because of being in **stty –tabs** or **stty tab3** mode, tabs are expanded to every eighth column.

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. Because every command requiring addresses has default addresses, the addresses can often be omitted.

In general, only one command can 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. Leave input mode by typing a period (.) 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 (such as 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 *RE*s 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 as follows:

a. The period (.), asterisk (*), left square bracket ([), and backslash (\) are always special, except when they appear within square brackets ([ ]); see 1.4 below.

b. The caret or circumflex (^) 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. The dollar sign ($) 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 is special for that RE (for example, see how slash (/) is used in the **g** command shown below.)

1.3 A period (.) is a one-character RE that matches any character except newline.

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 newline and the remaining characters in the string. The circumflex has this special meaning only if it occurs first in the string. The hyphen (–) 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). For example, **[ ]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 *RE*s 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*,\}, 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*,\} 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 (^) 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 ^*entire RE*$ constrains the entire RE to match the entire line.

The null RE (e.g., //) is equivalent to the last RE encountered. (See the last paragraph before **FILES**.)

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 character *x*, which must be an ASCII lowercase letter (a-z). 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 the last paragraph before **FILES**.)

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 the last paragraph before **FILES**.)
7. An address followed by a plus (+) or minus (–) sign followed by a decimal number specifies that address plus or 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. For example, **–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, the address **–** refers to the line preceding the current line. (To maintain compatibility with earlier versions of the editor, the character ^ in addresses is 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 are 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 can 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**) can be suffixed by **l**, **n**, or **p**. In this case, the current line is either listed, numbered or printed, respectively, as discussed below under the **l**, **n**, and **p** commands.

(.)a
<text>
.

The **a**ppend command reads the given text and appends it after the addressed line; **.** is left at the last inserted line, or, if no lines were inserted, 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 newline character).

(.)c
<text>
.

The **c**hange command deletes the addressed lines, then accepts input text that replaces these lines; **.** is left at the last line input, or, if no lines were input, at the first line that was not deleted.

C Same as the **X** command, except that **ed** assumes all text read in for the **e** and **r** commands is encrypted unless a null key is typed in.

(.,.)d The **d**elete 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 **e**dit command causes the entire contents of the buffer to be deleted and the named file to be read in; **.** is set to the last line of the buffer. If no filename is given, the currently remembered filename, 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 **!**, the rest of the line is taken to be a shell command whose output is to be read. Such a shell command is not remembered as the current filename. (See **DIAGNOSTICS**.)

E *file* The **E**dit 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**ilename command changes the currently remembered filename to *file*; otherwise, it prints the currently remembered filename.

(1,$)g/*RE***/***command list*
In the **g**lobal 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 ****; **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 **p** command. The **g**, **G**, **v**, and **V** commands are not permitted in the *command list*. (See **BUGS** and the last paragraph before **FILES**.)

(1,$)G/*RE***/** In the interactive **G**lobal command, the first step is to mark every line that matches the given RE. Then, for every such line, that line is printed, any command (other than **a**, **c**, **i**, **g**, **G**, **v**, and **V**) can be input and executed. After the execution of that command, the next marked line is printed, and so on. A newline acts as a null command; an **&** causes the re-execution of the most

recent command executed within the current invocation of G. Note that the commands input as part of the execution of the G command may address and affect any lines in the buffer. The G command can be terminated by an interrupt signal (ASCII DEL or BREAK).

h The **h**elp command gives a short error message that explains the reason for the most recent **?** diagnostic.

H The **H**elp command causes **ed** to enter a mode in which error messages are printed for all subsequent **?** diagnostics. It will also explain the previous **?** if there was one. The **H** command alternately turns this mode on and off. It is initially off.

(.)i
<text>
. The **i**nsert command inserts the given text before the addressed line; **.** is left at the last inserted line, or, if no lines were inserted, at the addressed line. This command differs from the **a** command only in the placement of the input text. Address 0 is not legal for this command. The maximum number of characters that can be entered from a terminal is 256 per line (including the new-line character).

(.,.+1)j The **j**oin command joins contiguous lines by removing the appropriate new-line characters. If exactly one address is given, this command does nothing.

(.)k*x* The **k** command marks the addressed line with name *x*, which must be an ASCII lowercase letter (**a**-**z**). The address **'***x* then addresses this line; **.** is unchanged.

(.,.)l The **l**ist command prints the addressed lines in an unambiguous way: a few non-printing characters (such as *tab*, *backspace*) are represented by visually mnemonic overstrikes. All other non-printing characters are printed in octal and long lines are folded. An **l** command may be appended to any command other than **e**, **f**, **r**, or **w**.

(.,.)m*a* The **m**ove command repositions the addressed line(s) after the line addressed by **a**. Address 0 is legal for **a** and causes the addressed line(s) to be moved to the beginning of the file. It is an error if address **a** falls within the range of moved lines; **.** is left at the last line moved.

(.,.)n The **n**umber command prints the addressed lines, preceding each line by its line number and a tab character; **.** is left at the last line printed. The **n** command can be appended to any command other than **e**, **f**, **r**, or **w**.

(.,.)p The **p**rint command prints the addressed lines; **.** is left at the last line printed. The **p** command can be appended to any command other than **e**, **f**, **r**, or **w**. For example, **dp** deletes the current line and prints the new current line.

P The editor will prompt with a * for all subsequent commands. The **P** command alternately turns this mode on and off. It is

initially off.

q The quit command causes **ed** to exit. No automatic write of a file is done. (See **DIAGNOSTICS.**)

Q 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 filename is given, the currently remembered filename, if any, is used (see the **e** and **f** commands). The currently remembered filename is not changed unless *file* is the first filename 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 command whose output is to be read. For example, **\$r !ls** appends the current directory listing to the end of the file being edited. Such a shell command is not remembered as the current filename.

(.,.)s/*RE*/*replacement*
(.,.)s/*RE*/*replacement*/**g**
(.,.)s/*RE*/*replacement*/**n** (n = 1-512)

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 newline may be used instead of / to delimit the RE and the *replacement*; **.** is left at the last line on which a substitution occurred. (See the last paragraph before **FILES.**)

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 can be split by substituting a newline character into it. The newline in the *replacement* must be escaped by preceding it with ****. Such substitution cannot be done as part of a **g** or **v**

command list.

(.,.)t*a* 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.

u The **u**ndo 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.

(1,$)v/*RE/command list*
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.

(1,$)V/*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.

(1,$)w *file* The **w**rite 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 filename is not changed unless *file* is the first filename mentioned since **ed** was invoked. If no filename is given, the currently remembered filename, 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 command whose standard input is the addressed lines. Such a shell command is not remembered as the current filename.

X A key is prompted for and it is used in subsequent **e**, **r**, and **w** commands to decrypt and encrypt text using the **crypt**(1) algorithm. An educated guess is made to determine whether text read in for the **e** and **r** commands is encrypted. A null key turns off encryption. 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.

!*shell command*
The remainder of the line after the **!** is sent to the shell to be interpreted as a command. Within the text of that command, the unescaped character **%** is replaced with the remembered filename. 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 and **.** is unchanged.

(.+1)<newline>
An address alone on a line causes the addressed line to print. A newline 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 in a line, 256 characters in a global command list, and 64 characters in the pathname of a file (counting slashes). 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.

If a file is not terminated by a newline character, **ed** adds one and puts out a message explaining what it did.

If the closing delimiter of a RE or a replacement string (such as /) would be the last character before a newline, that delimiter can 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

$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.
/usr/tmp	If */usr/tmp* exists, it is used as the directory name for the temporary work file.
/tmp	If the environmetal variable **TMPDIR** does not exist or is null and if */usr/tmp* does not exist, then */tmp* is used as the directory name for the temporary work file.
ed.hup	Work is saved here if the terminal is hung up.

SEE ALSO

csh(1), edit(1), ex(1), grep(1), ksh(1), sed(1), sh(1), stty(1), umask(1), vi(1), fspec(4), regexp(5)

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 the buffer via the **e** or **q** commands. It prints **?** and allows you to continue editing. A second **e** or **q** command at this point will take effect. The **–s** command line option inhibits this feature.

NOTES

The – option, although it continues to be supported, has been replaced in the documentation by the **–s** option that follows the Command Syntax Standard (see **intro**(1)).

The encryption options and commands are available only in the United States.

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.

The sequence **\n** in a RE does not match a newline character.

If the editor input is coming from a command file (such as **ed file < ed-cmd-file**), the editor will exit at the first failure.

NAME

edc – run an externally described command

SYNOPSIS

edc [**–i**] [**–T**] **–d** *descfile* **—** [*options*] [*operands*]

DESCRIPTION

An externally described command is run by calling **edc**. The ptx/ADMIN menu system Command Description Language and the **ec** compiler are required to create a description for an **edc** form. The compiler creates a description file in the *${TFADMFILE}/edc* directory, and **edc** uses the description file to run the form.

The **edc** and **prompt** programs are links to the same executable program. The **edc** program can run the menu system commands from the shell command line without displaying the form. The **prompt** program always displays the form and places the command line parameters in the appropriate input field.

–i The *interactive* option makes the menu system present the command's options and operands as an interactive input form. The **–i** option makes **edc** act the same as the **prompt** program.

–T The command is run in trace mode. This option instructs the menu system to display rather than execute the statements generated according to the command's description file. The **–T** option is especially useful for debugging command description files.

–d *descfile* Name of the command description file that the menu system will use. This is a compiled file created by the **ec** command compiler. Entry of a description file is required.

— The double dashes separate the options and operands for **edc** from the options and operands for the command.

options The options known to the command. Options must be separated from the command name by a double hyphen. The menu system reads the command description file and then applies the options to the command. When the **–i** option is used, the options are inserted in the input form presented by the menu system.

operands The operands used by the described command. Notice that operands always follow the double hyphen and any command options. When the **–i** option is used, the operands are inserted in the input form presented by the menu system.

Environment Variables

TFTERM Identifies the terminal type.

TERM If **TFTERM** is not set, this variable identifies the terminal type.

TERMINFO Identifies the location of the terminfo terminal descriptions. If this environment variable is not defined, the standard directory *${TFADMFILE}/terminfo* is used.

TFADMFILE Identifies the directory where the menu system administrative data files are located. This is used to locate the subdirectory *edc*, the default directory for the compiled files created by the **ec** command.

TFEDCPATH Identifies a list of directories as alternate locations for compiled files created by the **ec** command.

FILES

${TFADMFILE}/terminfo/[a–z]/*termtype*	Terminal definition file
${TFADMFILE}/edc/*command*	Compiled command description file

SEE ALSO

ec(1M), menu(1M), prompt(1M)

NAME

edit – text editor (variant of ex for casual users)

SYNOPSIS

edit [**–r**] [**–x**] [**–C**] *name...*

DESCRIPTION

edit, a variant of the text editor **ex**, is recommended for new or casual users wanting to use a command-oriented editor. It operates precisely as **ex**(1) with the following options set automatically:

novice	ON
report	ON
showmode	ON
magic	OFF

These options can be turned on or off via the **set** command in **ex**(1).

When invoking **edit**, you can use the following options:

–r Recover file after an editor or system crash.

–x Encryption option; when used the file will be encrypted as it is being written and will require an encryption key to be read. **edit** makes an educated guess to determine if a file is encrypted. See **crypt**(1).

–C Encryption option; the same as **–x** except that **edit** assumes files are encrypted.

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, type the command **edit** *name*. **edit** makes a copy of the file that you can edit and tells you how many lines and characters are in the file. To create a new file, type the command **edit** followed by a filename (**edit** *name*); the editor will tell you it is a `[New File]`.

The **edit** command prompt is the colon (:), which you should see after starting the editor. If you are editing an existing file, the lines from the file will be in **edit's** buffer (its name for the copy of the file you are editing). When you start editing, **edit** makes the last line of the file the current line. Most commands to **edit** use the current line if you do not tell them which line to use. Thus, if you say **print** (which can be abbreviated **p**) and type Return (as you should after all **edit** commands), the current line will print. If you **delete** (**d**) the current line, **edit** will print the new current line, which is usually the next line in the file. If you **delete** the last line, then the new last line becomes the current line.

If you start with an empty file or wish to add some new lines, the **append** (**a**) command can be used. After you execute this command (typing Return after the word **append**), **edit** will read lines from your terminal until you type a line consisting of just a dot (.). **edit** places 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 type before, rather than after, the current line.

edit numbers the lines in the buffer, with the first line having number 1. If you execute the command **1**, **edit** will type the first line of the buffer. If you then execute the command **d**, **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 text within the current line by using the **substitute** (**s**) command **s/*old*/*new*/** where *old* is the string of characters you want to replace and *new* is the string of characters that will replace *old*.

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 the buffer. After modifying a file, you can save the contents of the file by executing a **write** (**w**) command. You can 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 receive the message `No write since last change (:quit! overrides)`, and **edit** will wait for another command. If you do not want to write the buffer out, issue the **quit** command followed by an exclamation point (**q!**). The buffer is then irretrievably discarded and you return to the shell.

By using the **d** and **a** commands and giving line numbers to see lines in the file, you can make any changes you want. You should learn at least a few more things, however, if you will use **edit** more than a few times.

The **change** (**c**) command changes the current line to a sequence of lines you supply (as in **append**, you type lines up to a line consisting of only a dot (.). You can tell **change** to change more than one line by giving the line numbers of the lines you want to change, such as **3,5c**. You can also print lines this way. **1,23p** prints the first 23 lines of the file.

The **undo** (**u**) command reverses the effect of the last command you executed that changed the buffer. Thus, if you execute a **substitute** command that does not do what you want, type **u** and the old contents of the line will be restored. You can also **undo** an **undo** command. **edit** will give you a warning message when a command affects more than one line of the buffer. Note that commands such as **write** and **quit** cannot be undone.

To look at the next line in the buffer, type Return. To look at a number of lines, type Ctrl-D (while holding down the control key, press d) rather than Return. This will show you a half-screen of lines on a CRT or 12 lines on a hardcopy terminal.

You can look at nearby text by executing the **z** command. The current line will appear in the middle of the text displayed, and the last line displayed will become the current line. You can get back to the line where you were before you executed the **z** command by typing ''.

The **z** command has other options; **z–** 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 **z.*n***. When *n* is an odd number, **edit** displays a total of *n* lines, centered about the current line. For example, type **z.11** to display five lines before and five lines after the current line. When *n* is an even number, **edit** displays *n*–1 lines, so that the lines displayed are centered around the

current line. You can give counts after other commands. For example, the command **d5** deletes 5 lines starting with the current line.

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 backward and forward 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 *text*, it wraps 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 omit the trailing / or **?** in these commands.

The current line has the symbolic name dot (.). You can use this name to specify a range of lines. For example, **.,$p** prints the current line plus the rest of the lines in the file. To move to the last line in the file, you can refer to it by its symbolic name **$**. Thus the command **$d** deletes the last line in the file, no matter what the current line is. Arithmetic with line references is also possible. Thus the command **$–5** refers to the fifth line before the last line and the command **.+20** refers to 20 lines after the current line.

You can find out the current line by typing **.=**. This is useful if you wish to move or copy a section of text within a file or between files. Find the first and last line numbers you wish to copy or move. To move lines 10 through 20, type **10,20d a** to delete these lines from the file and place them in a buffer named **a**. **edit** has 26 such buffers named **a** through **z**. To put the contents of buffer **a** after the current line, type **put a**. If you want to move or copy these lines to another file, execute an **edit** (**e**) command after copying the lines. Then follow the **e** command with the name of the other file you want to edit, such as **edit chapter2**. To copy lines without deleting them, use **yank** (**y**) in place of **d**. If the text you wish to move or copy is all within one file, it is not necessary to use named buffers. For example, to move lines 10 through 20 to the end of the file, type **10,20m$**.

SEE ALSO

ed(1), ex(1), vi(1)

NOTES

The encryption options are available only in the United States.

NAME

edquota – edit user quotas

SYNOPSIS

/usr/etc/edquota [**–p** *proto-user*] *users* ...

/usr/etc/edquota –t

DESCRIPTION

edquota is a quota editor. One or more user login names or IDs can be specified on the command line. A temporary file with an ASCII representation of current disk quotas is created for each user and an editor is invoked on the file. New quotas can be added and existing quotas can be changed. Upon leaving the editor, **edquota** reads the temporary file and modifies the binary quota files to reflect the changes.

Unless the environment variable **EDITOR** specifies otherwise, the editor invoked is **vi**(1).

Only the superuser can edit quotas.

The following options are available:

–p Duplicate the quotas of the specified prototypical user for each user specified. This is the normal mechanism used to initialize quotas for groups of users.

–t Edit the soft time limits for each filesystem. If the time limits are zero, the default time limits are used. The default time limits are specified in the file *kernel/i386_space/param_space.c*. These default limits can be changed by reconfiguring the kernel. A time unit of either sec(onds), min(utes), hour(s), day(s), week(s), or month(s) must be supplied for each of the time limits. Time limits are printed in the greatest possible time unit such that the value is greater than or equal to one.

FILES

quotas	At the root of each filesystem with quotas
/etc/fstab	Contains filesystem names and locations

SEE ALSO

quota(1), quotacheck(1M), quotaon(1M), repquota(1M), quotactl(2seq)

DIAGNOSTICS

Various self-explanatory messages about inaccessible files

BUGS

The format of the temporary file is inscrutable.

NAME

efl – Extended Fortran Language

SYNOPSIS

efl [options] [files]

DESCRIPTION

efl compiles a program written in the EFL language into clean FORTRAN on the standard output. **efl** provides the C-like control constructs of *ratfor*(1) as follows:

statement grouping with braces

decision-making:

if, **if-else**, and **select-case** (also known as **switch-case**); **while**, **for**, FORTRAN **do**, **repeat**, and **repeat ... until** loops; multi-level **break** and **next**.

EFL has C-like data structures, for example:

```
struct
        {
        integer flags(3)
        character(8) name
        long real coords(2)
        } table(100)
```

The language offers generic functions, assignment operators (+=, &=, etc.), and sequentially evaluated logical operators (&& and | |). There is a uniform input/output syntax:

```
write(6,x,y:f(7,2), do i=1,10 { a(i,j),z.b(i) })
```

EFL also provides some syntactic "sugar":

free-form input:
multiple statements per line; automatic continuation; statement label names (not just numbers).

comments:
this is a comment.

translation of relational and logical operators:
>, >=, &, etc., become **.GT.**, **.GE.**, **.AND.**, etc.

return expression to caller from function:
return (*expression*)

defines:
define *name replacement*

includes:
include *file*

efl understands several option arguments: **–w** suppresses warning messages, **–#** suppresses comments in the generated program, and the default option **–C** causes comments to be included in the generated program.

An argument with an embedded = (equal sign) sets an EFL option as if it had appeared in an **option** statement at the start of the program. Many options are described in the reference manual. A set of defaults for a particular target

machine may be selected by one of the choices: **system=unix**, **system=gcos**, or **system=cray**. The default setting of the **system** option is the same as the machine the compiler is running on.

Other specific options determine the style of input/output, error handling, continuation conventions, the number of characters packed per word, and default formats.

efl is best used with **f77**(1).

SEE ALSO

cc(1), f77(1), ratfor(1)

NAME

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 alphanumeric 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 \(and \) and with the addition of the following:

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 newline 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 expressions, 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 newline.

If no files are specified, **egrep** assumes standard input. Normally, each line found is copied to the standard output. If there is more than one input file, the filename is printed before each line found.

Command line options are as follows:

–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 containing the pattern.

–i Ignore upper/lowercase distinction during comparisons.

–l Print the names of files with matching lines once, separated by newlines. 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 containing 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

csh(1), ed(1), fgrep(1), grep(1), ksh(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

enable, disable – enable/disable LP printers

SYNOPSIS

enable printers
disable [*options*] *printers*

DESCRIPTION

The **enable** command activates the named *printers*, enabling them to print requests taken by **lp**(1). Use **lpstat**(1) to find the status of printers.

The **disable** command 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. You can use the following options with the **disable** command:

–c Cancel any requests that are currently printing on any of the designated printers. This option cannot be used with the **–W** option.

–r *reason* Assign a *reason* for disabling the printers. This *reason* applies to all printers mentioned up to the next **–r** option. This *reason* is reported by **lpstat**(1). If the **–r** option is not present, a default reason will be used.

–W Disable the specified printers when the print requests currently printing have finished. This option cannot be used with the **–c** option.

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

csh(1), ksh(1), sh(1), exec(2), profile(4), environ(5)

NAME

ex – text editor

SYNOPSIS

ex [**–s**] [**–v**] [**–t** tag] [**–r** file] [**–L**] [**–R**] [**–x**] [**–C**] [**–c** command] file ...

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 focuses on the display-editing portion of **ex.**

For ed Users

If you have used **ed**(1) you will find that, in addition to having all of the **ed**(1) commands available, **ex** has a number of additional features useful on CRT terminals. Intelligent terminals and high speed terminals are very pleasant to use with **vi.** Generally, the **ex** editor uses far more of the capabilities of terminals than **ed**(1) does. **ex** uses the terminal capability database (see **terminfo**(4)) and the type of the terminal you are using (taken from the environmental variable **TERM**) to determine how to drive your terminal efficiently. The editor makes use of features such as inserting and deleting characters and lines 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 features for easily viewing the text of a file. The **z** command gives easy access to windows of text. Typing Ctrl-D causes the editor to scroll a half-window of text and is more useful for quickly stepping through a file than just typing Return. Of course, the screen-oriented **visual** mode gives constant access to editing context.

ex gives you 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 overwrite 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 (or **–r** *file* option) 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 filenames or a pattern as used by the shell to specify a new set of files to be dealt with. In general, filenames in the editor may be formed with full shell metasyntax. The metacharacter **%** is also available in forming filenames and is replaced by the name of the current file.

The editor has a group of buffers whose names are the ASCII lowercase letters (**a-z**). You can place text in these named buffers and then insert the text elsewhere in the file. The contents of the buffers remain available when you begin editing a new file using the **edit** (**e**) command.

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 the case of letters in searches and substitutions. **ex** also allows the construction of regular expressions to match words. This is convenient, for example, 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 your liking. For example, the **autoindent** option allows the editor to supply leading white space to align text automatically. You can then use Ctrl-D to backtab, and space or tab to move forward to align new code easily.

Miscellaneous useful features include an intelligent **join** (j) command that automatically supplies white space between joined lines, the commands < and >, which shift groups of lines, and the ability to filter portions of the buffer through commands such as **sort**(1).

Invocation Options

The following invocation options are interpreted by **ex** (previously documented options are discussed in the **NOTES** section at the end of this manual page):

–s	Suppress all interactive-user feedback. This is useful in processing editor scripts.
–v	Invoke **vi**.
–t *tag*	Edit the file containing the *tag* and position the editor at its definition.
–r *file*	Edit *file* after an editor or system crash. (Recovers the version of *file* that was in the buffer when the crash occurred.)
–L	List the names of all files saved as the result of an editor or system crash.
–R	**Readonly** mode; the **readonly** flag is set, preventing accidental overwriting of the file.
–x	Encryption option; when used, **ex** simulates an **X** command and prompts the user for a key. This key is used to encrypt and decrypt text using the algorithm of **crypt**(1). The **X** command makes an educated guess to determine whether text read in is encrypted. The temporary buffer file is encrypted also, using a transformed version of the key typed for the **–x** option. See **crypt**(1).
–C	Encryption option; the same as the **–x** option, except that **ex** simulates a **C** command. The **C** command is like the **X** command, except that all text read in is assumed to have been encrypted.
–c *command*	Begin editing by executing the specified editor *command* (usually a search or positioning command).

The *file* argument indicates one or more files to be edited.

ex States

Command	Normal and initial state. Input prompted for by :. Your line kill character cancels a partial command.
Insert	Entered by **a**, **i**, or **c**. Arbitrary text may be entered. Insert state normally is terminated by a line having only "." on it, or, abnormally, with an interrupt.
Visual	Entered by typing **vi**; terminated by typing **Q** or ^\ (Ctrl-\).

ex Command Names and Abbreviations

abbrev	**ab**	**map**		**set**	**se**
append	**a**	**mark**	**ma**	**shell**	**sh**
args	**ar**	**move**	**m**	**source**	**so**
change	**c**	**next**	**n**	**substitute**	**s**
copy	**co**	**number**	**nu**	**unabbrev**	**unab**
delete	**d**	**preserve**	**pre**	**undo**	**u**
edit	**e**	**print**	**p**	**unmap**	**unm**
file	**f**	**put**	**pu**	**version**	**ve**
global	**g**	**quit**	**q**	**visual**	**vi**
insert	**i**	**read**	**r**	**write**	**w**
join	**j**	**recover**	**rec**	**xit**	**x**
list	**l**	**rewind**	**rew**	**yank**	**ya**

ex Commands

shell escape	!
forced encryption	C
heuristic encryption	X
lshift	<
print next	Return
resubst	&
rshift	>
scroll	^D
window	z

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 variable
$HOME/.exrc	Editor initialization file
./.exrc	Editor initialization file
set *x*	Enable option *x*

set no*x*	Disable option *x*
set *x*=*val*	Give value *val* to option *x*
set	Show changed options
set all	Show all options
set *x*?	Show value of option *x*

Useful options and their abbreviations

autoindent	**ai**	Supply indent
autowrite	**aw**	Write before changing files
directory		Pathname of directory for temporary work files
exrc	**ex**	Allow **vi/ex** to read the *.exrc* in the current directory. This option is set in the **EXINIT** shell variable or in the *.exrc* file in the *$HOME* directory.
ignorecase	**ic**	Ignore case of letters in scanning
list		Print ^I for tab, $ at end
magic		Treat . [* special in patterns
modelines		First five lines and last five lines executed as **vi/ex** commands if they are of the form **ex:***command*: or **vi:***command*:
number	**nu**	Number lines
paragraphs	**para**	Macro names that start paragraphs
redraw		Simulate smart terminal
report		Informs you if the number of lines modified by the last command is greater than the value of the **report** variable
scroll		Command mode lines
sections	**sect**	Macro names that start sections
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
term		Specifies to **vi** the type of terminal being used (the default is the value of the environmental variable TERM)
window		Visual mode lines
wrapmargin	**wm**	Automatic line splitting
wrapscan	**ws**	Search around end (or beginning) of buffer

Scanning pattern formation

^	Beginning of line
$	End of line
.	Any character
\<	Beginning of word
\>	End of word
[*str*]	Any character in *str*
[^*str*]	Any character not in *str*
[*x*–*y*]	Any character between *x* and *y*

*	Any number of preceding characters

AUTHOR

vi and **ex** are based on software developed by The University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

FILES

/usr/lib/exstrings	Error messages
/usr/lib/exrecover	Recover command
/usr/lib/expreserve	Preserve command
/usr/lib/terminfo/*	Describes capabilities of terminals
$HOME/.exrc	Editor startup file
./.exrc	Editor startup file
/tmp/Ex*nnnnn*	Editor temporary
/tmp/Rx*nnnnn*	Named buffer temporary
/usr/preserve/*login*	Preservation directory (where *login* is the user's login)

SEE ALSO

crypt(1), ed(1), edit(1), grep(1), sed(1), sort(1), vi(1), curses(3X), term(4), terminfo(4)

NOTES

Several options, although they continue to be supported, have been replaced in the documentation by options that follow the Command Syntax Standard (see **intro**(1)). The – option has been replaced by **–s**, an **–r** option that is not followed with an option-argument has been replaced by **–L**, and **+***command* has been replaced by **–c** *command*.

The encryption options and commands are available only in the United States.

BUGS

The **z** command prints the 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 **–s** 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

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. Characters 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 characters 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 argument, which must be a regular expression. Regular expression syntax is the same as that of **ed**(1), except that all patterns are "anchored" (that is, begin with ^). 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

```
a=`expr $a + 1`
```
Adds 1 to the shell variable **a**.

```
# `For $a equal to either "/usr/abc/file" or just "file"'
expr $a : '.*/\(.*\)' \| $a
```
Returns the last segment of a pathname (i.e., *file*). Watch out for / alone as an argument; **expr** will take it as the division operator (see BUGS below).

```
# A better representation of example 2.
expr //$a : '.*/\(.*\)'
```

The addition of the // characters eliminates any ambiguity about the division operator and simplifies the whole expression.

```
expr $VAR : '.*'
```

Returns the number of characters in **$VAR**.

SEE ALSO

csh(1), ksh(1), 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 to the 14th power, 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 the square root of 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

ff – list filenames and statistics for a filesystem

SYNOPSIS

/etc/ff [options] special

DESCRIPTION

ff reads the i-list and directories of the *special* file, assuming it is a filesystem. Inode data is saved for files which match the selection criteria. Output consists of the pathname for each saved inode, plus other file information requested by using the print options below. Output fields are positional. The output is produced in inode order; fields are separated by tabs. The default line produced by **ff** is as follows:

pathname i-number

With all options enabled, output fields would be:

pathname i-number size uid

The argument *n* in the option descriptions that follow is used as a decimal integer (optionally signed), where *+n* means more than *n*, *–n* means less than *n*, and *n* means exactly *n*. A day is defined as a 24 hour period.

–I	Do not print the inode number after each pathname.
–l	Generate a supplementary list of all pathnames for multiply-linked files.
–p *prefix*	The specified *prefix* will be added to each generated pathname. The default is . (dot).
–s	Print the file size, in bytes, after each pathname.
–u	Print the owner's login name after each pathname.
–a *n*	Select if the inode has been accessed in *n* days.
–m *n*	Select if the inode has been modified in *n* days.
–c *n*	Select if the inode has been changed in *n* days.
–n *file*	Select if the inode has been modified more recently than the argument *file*.
–i *inode-list*	Generate names only for those inodes specified in *inode-list*.

SEE ALSO

find(1), ncheck(1M)

BUGS

If the **–l** option is not specified, only a single pathname out of all possible ones is generated for a multiply-linked inode. If **–l** is specified, all possible names for every linked file on the filesystem are included in the output. However, no selection criteria apply to the names generated.

NAME

fgrep – search a file for a character string

SYNOPSIS

fgrep [options] string [file ...]

DESCRIPTION

fgrep (fast **grep**) seaches 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 compact 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. If there is more than one input file, the filename is printed before each line found.

Command line options are as follows:

–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/lowercase 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

csh(1), ed(1), egrep(1), grep(1), ksh(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 having a *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 pathname-list expression

DESCRIPTION

find recursively descends the directory hierarchy for each pathname in the *pathname-list* (that is, one or more pathnames) 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 as follows:

–name *file* — True if *file* matches the current filename. Normal shell argument syntax may be used if escaped (watch out for **[**, **?** and ***).

[**–perm**] *–onum* — True if the file permission flags match the octal number *onum* exactly (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**, **p**, **f**, or **l** for block special file, character special file, directory, fifo (a.k.a named pipe), plain file or symbolic link 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 *pathname-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 pathname.

–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 pathname to print.

–cpio *device* — Always true; writes 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 filesystem containing the directory specified, or, if no directory was specified, to the filesystem containing the current directory.

-local — True if the file physically resides on the local system.

-fstype *type* — True if the filesystem to which the file belongs is of type *type*.

-prune — Always yields true. Has the side effect of pruning the search tree at the file.

(*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).

EXAMPLES

This example removes 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".

NAME

format – format disks or perform other defect management functions

SYNOPSIS

/etc/format –f|u|a xxx [–wov] scsi_diskname
/etc/format –f|r|m|y|l|u|a xxx **[–wovnc] –b** file **–s** n **–e** n **–p** n **–d** n **–t** type zdc_diskname

DESCRIPTION

The **format** program is used to format disks and also to perform various other defect management functions. Because formatting and some of the other functions of the **format** program are potentially dangerous in that they destroy the data on the disk, this program should be used with care. To be safe, **format** has some built-in protections. First, **format** opens the device exclusively. If filesystems are mounted or for any other reason the device is already open, **format** will fail. In addition, if the device has a valid VTOC (see **vtoc**(7)) and the specified function is one that destroys data on the disk, **format** will fail unless the **–o** overwrite option is specified.

The functions and options available vary according to the disk subsystem being operated on. Currently, **format** supports the *zdc* disk subsystem and the embedded SCSI disks on the *sd* and *wd* subsystems. **format** determines the subsystem based on the diskname. Therefore, the diskname must use the standard Sequent prefixes (e.g., *zd*, *sd*, *wd*). One of the arguments listed in the first section of the usage line above is required and indicates the function to be performed. The function, along with the subsystem, then determines which of the remaining arguments are necessary (or legal).

All supported functions and options are listed below. Note that each subsystem does not necessarily support each function. Since some functions and options have different requirements or expect arguments in different formats depending on the subsystem, they are described in greater detail under the subsystem description sections. Following is a simple description of each argument.

Functions

–f FORMAT the disk.

–r REFORMAT the disk using the disk's current bad block list.

–m REFORMAT the disk using the disk's copy of the manufacturer's defect list.

–a *xxx* ADD a bad block to the bad block list. Data on the bad block is recovered when possible.

–y VERIFY - run verify passes on the disk. During verify, data patterns are written on the disk and read back. If an error occurs, the block with the error is added to the bad block list.

–l LIST bad blocks. This displays the disk's bad block list.

–u USAGE will give a usage line based on the diskname provided.

Options

-v Enable verbose output.

-w Disable writing the diagnostic data at the end of the disk at the completion of the formatter function. By default, diagnostics are written after any operation which may destroy them, such as format or verify.

-n Disable doing the verify passes at the completion of the formatter function. By default, those subsystems which support the VERIFY function will do a default number of passes after a format or reformat operation.

-b *filename*
Specify the file containing the defect list.

-s *n* Specify the starting address for the verify passes. By default, verify passes start at the beginning of the disk. Note that when **-n** is set, this is ignored.

-e *n* Specify the ending address for the verify passes. By default, verify passes write the full disk. Note that when **-n** is set, this is ignored.

-p Specify the number of full verify passes to perform. A full verify pass causes the data patterns to be written from the specified starting address to the specified ending address. The default number of full passes is 3. Note that when **-n** is set, this is ignored.

-d Specify the number of defect verify passes to perform. A defect verify pass causes the data patterns to be written only on those cylinders with known defects. The default number of defect passes is 45. Note that when **-n** is set, this is ignored.

-c Check the data on the verify passes. Normally, during verify the data read back isn't actually compared to what was written. With this option, it is. Note that when **-n** is set, this is ignored.

-t *disktype*
Specify the type of the disk being operated on. This should correspond to one of the types specified by the */etc/diskinfo/*.zd* files.

-o Overwrite the VTOC on the disk. This is a protective feature. If the specified device has a valid VTOC and any function is specified that destroys data on the disk, **format** will fail unless this option is used.

ZDC Support

This section describes items specific to the *zdc* subsystem. This includes descriptions of the arguments which differ in format or meaning from the generic description.

format expects to find channel configuration for each supported ZDC disk type in */etc/diskinfo/*.zd,* where the base filename corresponds to a Sequent alias for the disk type. This is the name that would be entered for the **-t** type option. For example, the file *m2344k.zd* contains the channel configuration for the Fujitsu M2344K disk drive. To specify this disk type, you would use **-t** *m2344k.*

The *zdc* subsystem of the formatter supports the FORMAT function, as well as two types of REFORMAT (**-r** and **-m**). Be cautious in using the **-m** function, as any defects added to the bad block list after the original manufacturer's list

was put there will be lost. With any of the formatting functions (**-f|r|m**), verify passes will be done by default. This can be disabled with the **-n** option. When formatting (**-f**) a *zdc* disk, the following arguments are required:

format -f -b *defect_file* **-t** *disk_type zd_diskname*

Following are the arguments requiring description.

-a *xxx* The ADDBAD function requires an argument, which can be in one of two forms. The argument can be a filename, or it can be the actual defect data. **format** takes the first word following the **-a** and attempts to open a file corresponding to that word. If the open fails, it assumes the actual defect is specified. Defects for the ADDBAD function on the *zdc* disks contain the following four fields separated by white space:

cylinder head sector type

where *type* is 0 for a bad header, or 1 for bad data. Multiple defects may be specified in a file, with one defect per line.

-b *defect_file*
The defect file contains the manufacturer's list of defects. It should contain one defect per line, with each defect consisting of four fields separated by white space. The four fields are:
cylinder
head
number of bytes from index
length of defect in bits

-t *disktype*
The disk type must be one of the supported ZDC disk types. There must be a file in */etc/diskinfo* that corresponds to the specified type.

-s *n* The argument indicates the starting cylinder for verify passes.

-e*n* The argument indicates the last cylinder for verify passes.

SCSI Support

The SCSI subsystem of the **format** program supports CCS (Common Command Set) SCSI disks. This section describes items specific to the *scsi* subsystem. **format** expects a file */etc/diskinfo/*.scsi* corresponding to each supported SCSI disk type. These files contain information used by the **format** program. There are no specific differences or requirements for SCSI disks other than those described generically above.

Following are those arguments which require description.

-a *xxx* The ADDBAD function requires an argument, which can be in one of two forms. The argument can be a filename, or it can be the actual defect data. **format** takes the first word following the **-a** and attempts to open a file corresponding to that word. If the open fails, it assumes the actual defect is specified. Defects for the ADDBAD function on the SCSI disks consist of the logical block address of the defect. More than one may be entered on the command line, separated by spaces. In a file, they must be separated by newlines.

FILES

/etc/diskinfo/*.zd
/etc/diskinfo/*.scsi

SEE ALSO

CCSformat(8), zdformat(8)

DIAGNOSTICS

The diagnostics produced by **format** are intended to be self-explanatory.

NAME

fsck – filesystem consistency check and interactive repair

SYNOPSIS

/etc/fsck **–p** [filesystem ...]
/etc/fsck [**–A** block#] [**–c**] [**–y**] [**–n**] [filesystem] ...

DESCRIPTION

fsck with the **–p** option preens (checks and repairs) a standard set of filesystems or the specified filesystems. It is normally used in the script **/etc/rc** during automatic reboot. In this case **fsck** reads the table */etc/checklist* to determine which filesystems to check. It uses the information there to inspect groups of disks in parallel taking maximum advantage of I/O overlap to check the filesystems as quickly as possible. To maximize I/O overlap, **fsck** creates a list of partitions for each physical device to be checked. **fsck** tries to keep one instance of **fsck** active one partition of each physical device to be checked. When one partition on a physical device has completed **fsck**, another partition on that device is started immediately.

The system allows a restricted class of inconsistencies unless hardware or software failures intervene. These inconsistencies are limited to the following:

Unreferenced inodes

Link counts in inodes too large

Missing blocks in the free list

Blocks in the free list also in files

Counts in the superblock wrong

These are the only inconsistencies which **fsck** with the **–p** option will correct. If it encounters other inconsistencies, it exits with an abnormal return status and an automatic reboot then fails. For each corrected inconsistency, one or more lines are printed identifying the filesystem on which the correction will take place and the nature of the correction. After successfully correcting a filesystem, **fsck** prints the number of files on that filesystem, the number of used and free blocks, and the percentage of fragmentation.

Without the **–p** option, **fsck** audits and interactively repairs inconsistent conditions for filesystems. If the filesystem is inconsistent, the operator is prompted for agreement before each correction is attempted. Note that a number of the corrective actions which cannot be fixed under the **–p** option result in some loss of data. The amount and severity of data lost can be determined from the diagnostic output. The default action for each consistency correction is to wait for the operator to respond `yes` or `no`. If the operator does not have write permission, **fsck** defaults to the **–n** option.

fsck has more consistency checks than its predecessors **check, dcheck, fcheck,** and **icheck** combined.

The following options can be specified with **fsck**:

–A Use the block specified immediately after the flag as the superblock for the filesystem. Block 32 is always an alternate superblock.

–y Assume a `yes` response to all questions asked by **fsck**. This should be used with great caution as this allows **fsck** to continue after essentially

unlimited trouble has been encountered.

–n Assume a no response to all questions asked by **fsck**; do not open the filesystem for writing.

–c Convert the files system to the new (dynamic table) format, if it is in the old (static table) format. If the file system is in the new format, convert it to the old format provided the old format can support the filesystem configuration. In interactive mode, **fsck** lists the direction the conversion is to be made and asks whether the conversion should be done. If a negative answer is given, no further operations are done on the filesystem. With the **–p** option, the direction of the conversion is listed and done if possible without user interaction. Conversion with **–p** is best used when all the file systems are being converted at once. The format of a file system can be determined from the first line of output from **dumpfs**(1M).

If no filesystems are given to **fsck**, a default list of filesystems is read from the file */etc/checklist*.

Inconsistencies checked are as follows:

1. Blocks claimed by more than one inode or the free list.
2. Blocks claimed by an inode or the free list outside the range of the filesystem.
3. Incorrect link counts.
4. Size checks:
   Directory size not of proper format.
5. Bad inode format.
6. Blocks not accounted for anywhere.
7. Directory checks:
   File pointing to unallocated inode.
   Inode number out of range.
8. Superblock checks:
   More blocks for inodes than there are in the filesystem.
9. Bad free block list format.
10. Total free block and/or free inode count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with the operator's agreement, reconnected by placing them in the *lost+found* directory. The name assigned is the inode number. If the *lost+found* directory does not exist, it is created. If there is insufficient space in *lost+found*, its size is increased.

Checking the raw device is almost always faster.

FILES

/etc/checklist Contains default list of filesystems to check

DIAGNOSTICS

The diagnostics produced by **fsck** are intended to be self-explanatory.

SEE ALSO

crash(1M), dumpfs(1M), mkfs(1M), fs(4), checklist(4)

NAME

fsirand – install random inode generation numbers

SYNOPSIS

/etc/fsirand [**-p**] special

DESCRIPTION

fsirand installs random inode generation numbers on all the inodes on device *special*, and also installs a filesystem ID in the superblock. This helps increase the security of filesystems exported by NFS.

fsirand must be used only on an unmounted filesystem that has been checked with **fsck**(1M). The only exception is that it can be used on the root filesystem in single-user mode if the system is immediately re-booted afterwards.

fsirand is automatically invoked by newfs either mkfs when it they are used to create new a new file system.

The following option is available:

-p Print out the generation numbers for all the inodes, but do not change the generation numbers.

SEE ALSO

newfs(1M), mkfs(1M)

NAME

fsplit – split f77, ratfor, or efl files

SYNOPSIS

fsplit –fresd files

DESCRIPTION

fsplit splits the named *file(s)* into separate files, with one procedure per file. A procedure includes *blockdata*, *function*, *main*, *program*, and *subroutine* program segments. Depending on the language option chosen, procedure *X* is put in file *X*.**f**, *X*.**r**, or *X*.**e** with the following exceptions: *main* is put in the file *MAINn*.[**efr**] and unnamed *blockdata* segments are put in the files *blockdataN*.[**efr**] where *n* and *N* are unique integers values for each file.

The **fsplit** options are as follows:

–f Input files are *f77*. This is the default.

–r Input files are *ratfor*.

–e Input files are *efl*.

–s Strip *f77* input lines to 72 or fewer characters with trailing blanks removed.

–d Collect compiler directives (lines beginning with **$**) from the beginning of the file and place them at the beginning of each file that is created.

SEE ALSO

csplit(1), split(1)

NAME

fsstat – report filesystem status

SYNOPSIS

/etc/fsstat special_file

DESCRIPTION

fsstat reports on the status of the filesystem on *special_file*. During startup, this command is used to determine if the filesystem needs checking before it is mounted. **fsstat** succeeds if the filesystem is unmounted and appears okay. For the root filesystem, it succeeds if the filesystem is active and not marked bad.

SEE ALSO

fs(4)

DIAGNOSTICS

The command has the following exit codes:

0 – The filesystem is not mounted and appears okay
(except for root, where 0 means mounted and okay).
1 – The filesystem is not mounted and needs to be checked.
2 – The filesystem is mounted.
3 – The command failed.

NAME

fstyp – determine filesystem identifier

SYNOPSIS

/etc/fstyp *special*

DESCRIPTION

fstyp allows the user to determine the filesystem identifier of mounted or unmounted filesystems using heuristic programs. The filesystem type is required by **mount**(2) and sometimes by **mount**(1M) to mount filesystems of different types.

The directory */etc/fstyp.d* contains a program for each filesystem 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 filesystem 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 filesystem type identifier is printed and **fstyp** exits immediately. If no program succeeds, **fstyp** prints `Unknown_fstyp` to indicate failure.

SEE ALSO

mount(1M), mount(2), sysfs(2)

WARNINGS

The use of heuristics implies that the result of **fstyp** is not guaranteed to be accurate.

NAME

fuser – identify processes using a file or file structure

SYNOPSIS

/etc/fuser [**–ku**] *files* [**–**] [**–**[**ku**] *files*]

/etc/fuser [**–cfku**] *files* [**–**[**cfku**] *files*]

DESCRIPTION

fuser outputs the process IDs of the processes that are using the *files* specified as arguments.

In the **fuser** output, 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, all processes using any file on that device are listed. The **–c** flag may be used with files which are mount points for file systems. It specifies that the report is for the use of any files within the mounted file system. When the **–f** flag is used, the report is only for use of the named file, not files within a mounted device or file system. If the **–u** option is specified, the login name, in parentheses, also follows the process ID. In addition, if the **–k** option is specified, the SIGKILL signal is sent to each process. Options may be respecified between groups of files. The new set of options replaces the old set, with a lone dash canceling any options currently in force.

The process IDs are printed as a single line on the standard output, separated by spaces and terminated with a single newline. All other output is written on standard error.

Any user with permission to read */dev/kmem* and */dev/mem* can use **fuser**. Only the superuser can terminate another user's process.

FILES

/unix	For system namelist
/dev/kmem	For system image
/dev/mem	Also for system image

SEE ALSO

ps(1), mount(1M) kill(2), signal(2)

NAME

fwtmp, wtmpfix – manipulate connect accounting records

SYNOPSIS

/usr/lib/acct/fwtmp [**–ic**]
/usr/lib/acct/wtmpfix [files]

DESCRIPTION

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 fault 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

acctcom(1), ed(1), acct(1M), acctcms(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), runacct(1M), acct(2), acct(4), utmp(4)

NAME

gdev: hpd, erase, hardcopy, tekset, td – graphical device routines and filters

SYNOPSIS

hpd [–options] [GPS file ...]
erase
hardcopy
tekset
td [–ern*n*] [GPS file ...]

DESCRIPTION

All of the commands described below reside in */usr/bin/graf* (see **graphics**(1G)).

hpd **hpd** translates a GPS (graphical primitive string; see **gps**(4)) to instructions for the Hewlett-Packard 7221A Graphics Plotter. A viewing window is computed from the maximum and minimum points in *file* unless the **–u** or **–r** option is provided. If no *file* is given, the standard input is assumed. The options for the **hpd** command are as follows:

- **c***n* Select character set *n*, *n* between 0 and 5 (see the *HP7221A Plotter Operating and Programming Manual, Appendix A*).
- **p***n* Select pen numbered *n*, *n* between 1 and 4 inclusive.
- **r***n* Window on GPS region *n*, *n* between 1 and 25 inclusive.
- **s***n* Slant characters *n* degrees clockwise from the vertical.
- **u** Window on the entire GPS universe.
- **xd***n* Set x displacement of the viewport's lower left corner to *n* inches.
- **xv***n* Set width of viewport to *n* inches.
- **yd***n* Set y displacement of the viewport's lower left corner to *n* inches.
- **yv***n* Set height of viewport to *n* inches.

erase **erase** sends characters to a Tektronix 4010 series storage terminal to erase the screen.

hardcopy When issued at a Tektronix display terminal with a hard copy unit, **hardcopy** generates a screen copy on the unit.

tekset **tekset** sends characters to a Tektronix terminal to clear the display screen, set the display mode to alpha, and set characters to the smallest font.

td **td** translates a GPS to scope code for a Tektronix 4010 series storage terminal. A viewing window is computed from the maximum and minimum points in *file* unless the **–u** or **–r** *option* is provided. If no *file* is given, the standard input is assumed. Options are as follows:

- **e** Do not erase screen before initiating display.
- **r***n* Display GPS region *n*, *n* between 1 and 25 inclusive.

u Display the entire GPS universe.

SEE ALSO

ged(1G), graphics(1G), gps(4)

NAME

ged – graphical editor

SYNOPSIS

ged [**–eruR**]*n*] [GPS file ...]

DESCRIPTION

ged is an interactive graphical editor used to display, construct, and edit GPS files on Tektronix 4010 series display terminals. If GPS *file*(s) are given, **ged** reads them into an internal display buffer and displays the buffer. The GPS in the buffer can then be edited. If – is given as a filename, **ged** reads a GPS from the standard input.

ged accepts the following command line options:

e Do not erase the screen before the initial display.

r*n* Display region number *n*.

u Display the entire GPS *universe*.

R Restricted shell invoked on use of !.

A GPS file is composed of instances of three graphical objects: *lines*, *arc*, and *text*. *arc* and *lines* objects have a start point, or *object-handle*, followed by zero or more points, or *point-handles*. *Text* has only an object-handle. The objects are positioned within a Cartesian plane, or universe, having 64K (–32K to +32K) points, or universe-units, on each axis. The universe is divided into 25 equal sized areas called regions. Regions are arranged in five rows of five squares each, numbered 1 to 25 from the lower left of the universe to the upper right.

ged maps rectangular areas, called windows, from the universe onto the display screen. Windows allow the user to view pictures from different locations and at different magnifications. The universe-window is the window with minimum magnification (that is, the window that views the entire universe. The home-window is the window that completely displays the contents of the display buffer).

Commands

ged commands are entered in stages. Typically, each stage ends with a Return. Prior to the final Return, the command may be aborted by typing the Rubout key. The input of a stage may be edited during the stage by using the Erase and Kill characters of the calling shell. The prompt * indicates that **ged** is waiting at stage 1.

Each command consists of a subset of the following stages:

1. *Command line*
   A command line consists of a command *name* followed by *argument*(s) followed by a Return. A command *name* is a single character. Command *arguments* are either *option*(s) or a *filename*. *Options* are indicated by a leading –.
2. *Text* *Text* is a sequence of characters terminated by an unescaped (120 lines of text maximum).
3. *Points* *Points* is a sequence of one or more screen locations (maximum of 30) indicated either by the terminal crosshairs or by name. The

prompt for entering *points* is the appearance of the crosshairs. When the crosshairs are visible, you can type these commands:

sp (space)
Enters the current location as a *point*. The *point* is identified with a number.

$*n* Enters the previous *point* numbered *n*.

>*x* Labels the last *point* entered with the uppercase letter *x*.

$*x* Enters the *point* labeled *x*.

. Establishes the previous *points* as the current *points*. At the start of a command, the previous *points* are those locations given with the previous command.

= Echoes the current *points*.

$*n* Enters the *point* numbered *n* from the previous *points*.

Erases the last *point* entered.

@ Erases all of the *points* entered.

4. *Pivot* The *pivot* is a single location, entered by typing Return or by using the **$** operator, and indicated with a *****.

5. *Destination*
The *destination* is a single location entered by typing Return or by using **$**.

Command Summary

In the summary, characters typed by the user are printed in **bold**. Command stages are printed in italics. Arguments surrounded by brackets "[ ]" are optional. Parentheses "( )" surrounding arguments separated by "or" means that exactly one of the arguments must be given.

Construct commands:

Arc	[**–echo,style,weight**] *points*
Box	[**–echo,style,weight**] *points*
Circle	[**–echo,style,weight**] *points*
Hardware	[**–echo**] *text points*
Lines	[**–echo,style,weight**] *points*
Text	[**–angle,echo,height,mid-point,right-point,text,weight**] *text points*

Edit commands:

Delete	(**–** (universe or view) or *points*)
Edit	[**–angle,echo,height,style,weight**] (**–** (universe or view) or *points*)
Kopy	[**–echo,points,x**] *points pivot destination*
Move	[**–echo,points,x**] *points pivot destination*

Rotate	[**–angle,echo,kopy,x**] *points pivot destination*
Scale	[**–echo,factor,kopy,x**] *points pivot destination*

View commands:

coordinates	*points*
erase	
new-display	
object-handles	(– (**universe** or **view**) or *points*)
point-handles	(– (**labelled-points** or **universe** or **view**) or *points*)
view	(– (**home** or **universe** or **region**) or [**–x**] *pivot destination*)
x	[**–view**] *points*
zoom	[**–out**] *points*

Other commands:

quit or **Quit**	
read	[**–angle,echo,height,mid-point,right-point,text,weight** *filename* [*destination*]
set	[**–angle,echo,factor,height,kopy,mid-point,points, right-point,style,text,weight,x**]
write	*filename*
!*command*	
?	

Options:

Options specify parameters used to construct, edit, and view graphical objects. If a parameter used by a command is not specifed as an option, the default value for the parameter will be used (see **set** below). The format of command options is as follows where option is *keyletter*[*value*].

–*option*[,*option*]

Flags take on the values of true or false, indicated by + and – respectively. If no value is given with a flag, true is assumed.

Object options:

angle*n*	Angle of *n* degrees.
echo	When true, echo additions to the display buffer.
factor*n*	Scale factor is *n* percent.
height*n*	Height of *text* is *n* universe-units ($0 \le n < 1280$).
kopy	When true, copy rather than move.
mid-point	When true, mid-point is used to locate text string.

points When true, operate on points; otherwise operate on objects.

right-point When true, right-point is used to locate *text* string.

style*type* Line style set to one of following *types*:

so	solid
da	dashed
dd	dot-dashed
do	dotted
ld	long-dashed

text When false, *text* strings are outlined rather than drawn.

weight*type* Sets line weight to one of following *types*:

n	narrow
m	medium
b	bold

Area options:

home Reference the home-window.

out Reduce magnification.

region*n* Reference region *n*.

universe Reference the universe-window.

view Reference those objects currently in view.

x Indicate the center of the referenced area.

Command Descriptions

Construct commands:

Arc and Lines

These commands behave similarly. Each consists of a *command line* followed by *points*. The first *point* entered is the object-handle. Successive *points* are point-handles. Lines connect the handles in numerical order. Arc fits a curve to the handles (currently a maximum of 3 points will be fit with a circular arc; splines will be added in a later version).

Box and Circle

These commands are special cases of Lines and Arc, respectively. Box generates a rectangle with sides parallel to the universe axes. A diagonal of the rectangle would connect the first *point* entered with the last *point*. The first *point* is the object-handle. Point-handles are created at each of the vertices. Circle generates a circular arc centered about the *point* numbered zero and passing through the last *point*. The circle's object-handle coincides with the last *point*. A point-handle is generated 180 degrees around the circle from the object-handle.

Text and Hardware

These commands generate *text* objects. Each consists of a *command line*, *text*, and *points*. *Text* is a sequence of characters delimited by Return. Multiple lines of text may be entered by preceding a Return with a backslash (i.e., \Return). The Text command creates software-generated characters. Each line of software text is treated as a separate *text* object. The first *point* entered is the object-handle for the first line of text. The Hardware command sends the characters in *text* uninterpreted to the

terminal.

Edit commands:

Edit commands operate on portions of the display buffer called defined areas. A defined area is referenced either with an area option or interactively. If an area option is not given, the perimeter of the defined area is indicated by *points*. If no *point* is entered, a small defined area is built around the location of the Return. This is useful to reference a single *point*. If only one *point* is entered, the location of the Return is taken in conjunction with the *point* to indicate a diagonal of a rectangle. A defined area referenced by *points* will be outlined with dotted lines.

Delete
Removes all objects whose object-handle lies within a defined area. The **universe** option removes all objects and erases the screen.

Edit Modifies the parameters of the objects within a defined area. Parameters that can be edited are as follows:

angle	Angle of *text*
height	Height of *text*
style	Style of *lines* and *arc*
weight	Weight of *lines*, *arc*, and *text*

Kopy (or Move)
Copies (or moves) object- and/or point-handles within a defined area by the displacement from the *pivot* to the *destination*.

Rotate
Rotates objects within a defined area around the *pivot*. If the **kopy** flag is true, the objects are copied rather than moved.

Scale
For objects whose object handles are within a defined area, point displacements from the *pivot* are scaled by **factor** percent. If the **kopy** flag is true, the objects are copied rather than moved.

View commands:

coordinates
Prints the location of *point*(s) in universe- and screen-units.

erase
Clears the screen (but not the display buffer).

new-display
Erases the screen, then displays the display buffer.

object-handles (or point-handles)
Labels object-handles (and/or point-handles) that lie within the defined area with **O** (or **P**). **Point-handles** identifies labeled points when the **labeled-points** flag is true.

view
Moves the window so that the universe point corresponding to the *pivot* coincides with the screen point corresponding to the *destination*. Options for **home**, **universe**, and **region** display particular windows in the universe.

x Indicates the center of a defined area. Option view indicates the center of the screen.

zoom
Decreases (zoom out) or increases the magnification of the viewing window based on the defined area. For increased magnification, the window is set to circumscribe the defined area. For a decrease in magnification, the current window is inscribed within the defined area.

Other commands:

quit or Quit
Exit from *ged*. Quit responds with ? if the display buffer has not been written since the last modification.

read Inputs the contents of a file. If the file contains a GPS, it is read directly. If the file contains text it is converted into *text* object(s). The first line of a text file begins at *destination*.

set When given *option*(s), resets default parameters. Otherwise ,it prints current default values. .

write
Outputs the contents of the display buffer to a file.

! Escapes **ged** to execute an operating system command.

? Lists **ged** commands.

SEE ALSO

gdev(1G), graphics(1G), sh(1), gps(4)

WARNINGS

See Appendix A of the *Tektronix 4014 Computer Display Terminal User's Manual* for a discussion of the appropriate terminal strap options.

NAME

gencc – create a front-end to the cc command

SYNOPSIS

gencc

DESCRIPTION

The **gencc** command is an interactive command designed to aid in the creation of a front-end to the **cc** command. Since hard-coded pathnames have been eliminated from the C Compilation System (CCS), it is possible to move pieces of the CCS to new locations without recompiling the CCS. The new locations of moved pieces can be specified through the **–Y** option to the **cc** command. However, it is inconvenient to supply the proper **–Y** options with every invocation of the **cc** command. Further, if a system administrator moves pieces of the CCS, such movement should be invisible to users.

The front-end to the **cc** command which **gencc** generates is a one-line shell script which calls the **cc** command with the proper **–Y** options specified. The front-end to the **cc** command will also pass all user supplied options to the **cc** command.

gencc prompts for the location of each tool and directory which can be respecified by a **–Y** option to the **cc** command. If no location is specified, it assumes that that piece of the CCS has not been relocated. After all the locations have been prompted for, **gencc** will create the front-end to the **cc** command.

gencc creates the front-end to the **cc** command in the current working directory and gives the file the same name as the **cc** command. Thus, **gencc** can not be run in the same directory containing the actual **cc** command. Further, if a system administrator has redistributed the CCS, the actual **cc** command should be placed somewhere which is not typically in a user's **PATH** (for example, */lib*). This will prevent users from accidentally invoking the **cc** command without using the front-end.

FILES

./cc	Front-end to cc

SEE ALSO

cc(1)

WARNINGS

gencc does not produce any warnings if a tool or directory does not exist at the specified location. Also, **gencc** does not actually move any files to new locations.

NAME

get – get a version of an SCCS file

SYNOPSIS

get [**–r**SID] [**–c**cutoff] [**–i**list] [**–x**list] [**–w**string] [**–a**seq-no.] [**–k**] [**–e**] [**–l**[p]] [**–p**] [**–m**] [**–n**] [**–s**] [**–b**] [**–g**] [**–t**] file ...

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 pathname 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 filename by simply removing the leading *s.*; (see **FILES**).

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 apply independently to each named file.

–r*SID* The SCCS identification string (SID) of the version (delta) of an SCCS file to be retrieved. Table 1 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.

–c*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 **get**s within, say, the input to a **send**(1C) command:

```
~!get "–c%E% %U%" s.file
```

–i*list* 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.

-x*list* A *list* of deltas to be excluded in the creation of the generated file. See the **-i** 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 **get**s 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.) 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 that 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 (for example, **–r1**), or release and level (for example, **–r1.2**).

–w *string* Substitute *string* for all occurrences of %W% when getting the file.

–a*seq-no.* The delta sequence number of the SCCS file delta (version) to be retrieved [see **sccsfile**(4)]. 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 filename 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

SID* Specified	**–b** Keyletter Used†	Other Conditions	SID Retrieved	SID of Delta to be Created
none‡	no	R defaults to mR	mR.mL	mR.(mL+1)
none‡	yes	R defaults to mR	mR.mL	mR.mL.(mB+1).1
R	no	R > mR	mR.mL	R.1***
R	no	R = mR	mR.mL	mR.(mL+1)
R	yes	R > mR	mR.mL	mR.mL.(mB+1).1
R	yes	R = mR	mR.mL	mR.mL.(mB+1).1
R	–	R < mR and R does *not* exist	hR.mL**	hR.mL.(mB+1).1
R	–	Trunk succ.#	R.mL	R.mL.(mB+1).1

		in release > R and R exists		
R.L	no	No trunk succ.	R.L	R.(L+1)
R.L	yes	No trunk succ.	R.L	R.L.(mB+1).1
R.L	–	Trunk succ. in release ≥ R	R.L	R.L.(mB+1).1
R.L.B	no	No branch succ.	R.L.B.mS	R.L.B.(mS+1)
R.L.B	yes	No branch succ.	R.L.B.mS	R.L.(mB+1).1
R.L.B.S	no	No branch succ.	R.L.B.S	R.L.B.(S+1)
R.L.B.S	yes	No branch succ.	R.L.B.S	R.L.(mB+1).1
R.L.B.S	–	Branch succ.	R.L.B.S	R.L.(mB+1).1

* R, L, B, and S are the release, level, branch, and sequence components of the SID, respectively; m means maximum. 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 filename
%P%	Fully qualified SCCS filename
%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 **@(#)** 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 filename is formed from the SCCS filename: the last component of all SCCS filenames 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 *s.xyz.c*, the auxiliary filenames 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 of creation (in the form YY/MM/DD HH:MM:SS)
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 newline. 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

getfrm – returns the current frame number

SYNOPSIS

getfrm

DESCRIPTION

The **getfrm** command takes no arguments. It returns the current frame number.

EXAMPLES

```
action=OPEN TEXT Text.std `getfrm`
```

The action here is to open the text object *Text.std,* passing the current frame number to that text object as its ARG1.

NAME

getitems – returns a list of the currently marked menu items

SYNOPSIS

getitems [delimiter_string]

DESCRIPTION

The **getitems** command takes a delimiter string as its only argument. It returns a list of the names (or `$LININFO`, if it is defined) of the currently marked menu items, delimited by the argument string. If no argument is given, the default delimiter is NEWLINE.

EXAMPLES

This code defines a menu:

```
Menu="Example"
multiselect=TRUE
done='getitems ":" | message'

name="Item 1"
action='message "You selected item 1"'

name="Item 2"
lininfo="This is item 2"
action='message "You selected item 2"'

name="Item 3"
action='message "You selected item 3"'
```

If all three items in this menu were marked when the Return key is pressed, the following string appears on the message line:

```
Item 1:This is item 2:Item 3
```

Note that since `$LININFO` is defined for item 2, its value is substituted for the name.

NAME

getopt – parse command options

SYNOPSIS

set -- `getopt** optstring **$*`

DESCRIPTION

WARNING: Start using the new command **getopts**(1) in place of **getopt**(1). **getopt**(1) will not be supported in the next major release. For more information, see the **WARNINGS** section.

getopt is used to break up options in command lines for easy parsing by shell procedures and to check for legal options. *optstring* 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.

EXAMPLES

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 -- `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 accepts 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), ksh(1), sh(1), getopt(3C)

DIAGNOSTICS

getopt prints an error message on the standard error when it encounters an option letter not included in *optstring*.

WARNINGS

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 conversion tool, **getoptcvt**, has been provided. For more information about **getopts** and **getoptcvt**, see **getopts**(1).

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 **getopts** 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, **getopts** 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, **getopts** 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, **getopts** exits with a non-zero exit status. The special option "**--**" may be used to delimit the end of the options.

By default, **getopts** parses the positional parameters. If extra arguments (*arg ...*) are given on the **getopts** command line, **getopts** will parse them instead.

/usr/lib/getoptcvt reads the shell script in *file*, converts it to use **getopts**(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 operating 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 **getopts**(1) or **getopt**(1).

To ensure that all new commands adhere to the command syntax standard described in **intro**(1), they should use **getopts**(1) or **getopt**(3C) to parse positional parameters and check for options that are legal for that command (see **WARNINGS**, below).

EXAMPLES

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 getopts abo: c
do
        case $c in
        a | b)      FLAG=$c;;
        o)          OARG=$OPTARG;;
        \?)         echo $USAGE
```

```
                exit 2;;
        esac
done
shift `expr $OPTIND - 1`
```

This code accepts 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), ksh(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 *optstring*.

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 operating system. It can only be executed by the superuser; 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** at 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. In any type except **none**, a **TERM** environment variable is created and exported. **getty** recognizes the following types:

none	default
ds40-1	Dataspeed40/1
tektronix,tek	Tektronix
vt61	DEC vt61
vt100	DEC vt100
hp45	Hewlett-Packard 45
c100	Concept 100

The default terminal is **none** (that is, 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, newline 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, it is terminated by a newline 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 lowercase alphabetic characters; if not, and if the name is non-empty, the system is told to map any future uppercase characters into the corresponding lowercase 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

login (1), ct(1C), init(1M), ioctl(2), gettydefs(4), inittab(4), 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, Ctrl-U, Ctrl-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

glossary – definitions of common operating system terms and symbols

SYNOPSIS

[**help**] **glossary** [term]

DESCRIPTION

The Help Facility **glossary** command provides definitions of common technical terms and symbols.

Without an argument, **glossary** displays a menu screen listing the terms and symbols that are currently included in **glossary.** A user can either choose one of the terms or exit to the shell by typing q (for "quit"). When a term is selected, its definition is retrieved and displayed. By selecting the appropriate menu choice, the list of terms and symbols can be redisplayed.

A term's definition may also be requested directly from shell level (as shown above), causing a definition to be retrieved and the list of terms and symbols not to display. Some of the symbols must be escaped if requested at shell level in order for the facility to understand the symbol. The following table lists the symbols and their escape sequences:

Symbol	Escape Sequence
""	\"\"
”	\'\'
[]	\\[\\]
“	\‘\‘
#	\#
&	\&
*	*
\	\\\\
|	\\|

From any screen in the Help Facility, a user can execute a command via the shell by typing an exclamation mark (!) and the command to be executed. The screen will be redrawn if the command was entered at a first-level prompt. If entered at any other prompt level, only the prompt will be redrawn.

By default, the Help Facility scrolls the data that is presented to the user. If you prefer to have the screen clear before printing the data (non-scrolling), the shell variable **SCROLL** must be set to **no** and exported so it will become part of your environment. This is done by adding the following line to your *.profile* file (see **profile**(4)): "`export SCROLL ; SCROLL=no`". If you later decide that scrolling is desired, SCROLL must be set to **yes.**

Information on each of the Help Facility commands (**starter, locate, usage, glossary,** and **help**) is located on their respective manual pages.

SEE ALSO

csh(1), help(1), helpadm(1M), ksh(1), locate(1), sh(1), starter(1), usage(1), term(5)

WARNINGS

If the shell variable **TERM** is not set in the user's *.profile* file, then **TERM** will default to the terminal value type 450 (a hard-copy terminal). For a list of valid terminal types, refer to **term**(5).

NAME

graph – draw a graph

SYNOPSIS

graph [options]

DESCRIPTION

graph with no options takes pairs of numbers from the standard input as abscissas and ordinates of a graph. Successive points are connected by straight lines. The graph is encoded on the standard output for display by the **tplot**(1G) filters.

If the coordinates of a point are followed by a non-numeric string, that string is printed as a label beginning on the point. Labels may be surrounded with quotes ("), in which case they may be empty or contain blanks and numbers; labels never contain newlines.

The following options are recognized, each as a separate argument:

–a Supply abscissas automatically (they are missing from the input); spacing is given by the next argument (default 1). A second optional argument is the starting point for automatic abscissas (default 0 or lower limit given by **–x**).

–b Break (disconnect) the graph after each label in the input.

–c Character string given by next argument is default label for each point.

–g Next argument is grid style, 0 no grid, 1 frame with ticks, 2 full grid (default).

–l Next argument is label for graph.

–m Next argument is mode (style) of connecting lines: 0 disconnected, 1 connected (default). Some devices give distinguishable line styles for other small integers (e.g., the Tektronix 4014: 2=dotted, 3=dash-dot, 4=short-dash, 5=long-dash).

–s Save screen, do not erase before plotting.

–x [l] If l is present, x axis is logarithmic. Next 1 (or 2) arguments are lower (and upper) *x* limits. Third argument, if present, is grid spacing on *x* axis. Normally, these quantities are determined automatically.

–y [l] Similarly for *y*.

–h Next argument is fraction of space for height.

–w Similarly for width.

–r Next argument is fraction of space to move right before plotting.

–u Similarly to move up before plotting.

–t Transpose horizontal and vertical axes. (Option **–x** now applies to the vertical axis.)

A legend indicating grid range is produced with a grid unless the **–s** option is present. If a specified lower limit exceeds the upper limit, the axis is reversed.

SEE ALSO

graphics(1G), spline(1G), tplot(1G)

BUGS

graph stores all points internally and drops those for which there is no room.

Segments that run out of bounds are dropped, not windowed.

Logarithmic axes may not be reversed.

NAME

graphics – access graphical and numerical commands

SYNOPSIS

graphics [–r]

DESCRIPTION

graphics prefixes the pathname */usr/bin/graf* to the current **$PATH** value, changes the primary shell prompt to a caret (^), and executes a new shell. The directory */usr/bin/graf* contains all of the Graphics subsystem commands. If the **–r** option is given, access to the graphical commands is created in a restricted environment; that is, **$PATH** is set to the following:

```
: /usr/bin/graf : /rbin : /usr/rbin : /bin : /usr/bin
```

The restricted shell (**rsh**) is also invoked. To restore the environment that existed prior to issuing the **graphics** command, type EOT (Ctrl-D on most terminals). To logoff from the graphics environment, type **quit**.

The command line format for a command in **graphics** is *command name* followed by *argument*(s). An *argument* may be a *filename* or an *option string*. A *filename* is the name of any operating system file except those beginning with –. The *filename* **2-** is the name for the standard input. An *option string* consists of – followed by one or more *option*(s). An option consists of a keyletter possibly followed by a value. Options may be separated by commas.

The graphical commands have been partitioned into four groups.

- Commands that manipulate and plot numerical data; see **stat**(1G)
- Commands that generate tables of contents; see **toc**(1G)
- Commands that interact with graphical devices; see **gdev**(1G) and **ged**(1G)
- A collection of graphical utility commands; see **gutil**(1G)

A list of the **graphics** commands can be generated by typing **whatis** in the **graphics** environment.

SEE ALSO

gdev(1G), ged(1G), gutil(1G), stat(1G), toc(1G), gps(4)

NAME

greek – select terminal filter

SYNOPSIS

greek [**–T**terminal]

DESCRIPTION

greek is a filter that reinterprets the extended character set, as well as the reverse and half-line motions, of a 128-character Teletype Model 37 terminal for certain other terminals. Special characters are simulated by overstriking, if necessary and possible. If the argument is omitted, **greek** attempts to use the environment variable **$TERM** (see **environ**(5)). Currently, the following *terminal*s are recognized:

300	DASI 300.
300-12	DASI 300 in 12-pitch.
300s	DASI 300s.
300s-12	DASI 300s in 12-pitch.
450	DASI 450.
450-12	DASI 450 in 12-pitch.
1620	Diablo 1620 (alias DASI 450).
1620-12	Diablo 1620 (alias DASI 450) in 12-pitch.
2621	Hewlett-Packard 2621, 2640, and 2645.
2640	Hewlett-Packard 2621, 2640, and 2645.
2645	Hewlett-Packard 2621, 2640, and 2645.
4014	Tektronix 4014.
hp	Hewlett-Packard 2621, 2640, and 2645.
tek	Tektronix 4014.

FILES

/usr/bin/300
/usr/bin/300s
/usr/bin/4014
/usr/bin/450
/usr/bin/hp

SEE ALSO

300(1), 4014(1), 450(1), hp(1), tplot(1G), environ(5), term(5)

eqn(1), mm(1), nroff(1) in the *DOCUMENTER'S WORKBENCH Technical Discussion and Reference Manual*

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 having 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 filename is printed before each line found if there is more than one input file.

Command line options are as follows:

–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/lowercase distinction during comparisons.

–l Print the names of files with matching lines once, separated by newlines. 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

csh(1), ed(1), egrep(1), fgrep(1), ksh(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 BUFSIZ characters; longer lines are truncated. BUFSIZ is defined in */usr/include/stdio.h*.

If there is a line with embedded nulls, **grep** will match up to the first null only; if it matches, it will print the entire line.

NAME

groups – show group memberships

SYNOPSIS

groups

DESCRIPTION

The **groups** command shows the groups to which you belong. Each user belongs to a group specified in the password file */etc/passwd* and possibly to other groups as specified in the file */etc/group*. If you do not own a file but you belong to the group that owns it, you are granted group access to the file.

When a new file is created, it is assigned to the group that owns the directory containing the file.

SEE ALSO

setgroups(2)

FILES

/etc/passwd, /etc/group

NAME

gutil – graphical utilities

SYNOPSIS

command-name [options] [files]

DESCRIPTION

Below is a list of miscellaneous device-independent utility commands found in */usr/bin/graf*. If no *files* are given, input is from the standard input. All output is to the standard output. Graphical data is stored in GPS format; see **gps**(4).

bel – Send bel character to terminal

cvrtopt [=**s***string* **f***string* **i***string* **t***string*] [*args*] – Options converter
cvrtopt reformats *args* (usually the command line arguments of a calling shell procedure) to facilitate processing by shell procedures. An *arg* is either a filename (a string not beginning with a –, or a – by itself) or an option string (a string of options beginning with a –). Output is of the form:

–option –option . . . filename(s)

All options appear singularly and preceding any filenames. Options that take values (e.g., –r1.1) or are two letters long must be described through options to **cvrtopt**.

cvrtopt is usually used with **set** in the following manner as the first line of a shell procedure:

```
"set - `cvrtopt =[" options "] $@`"
```

Options to **cvrtopt** are as follows:

s*string* *String* accepts string values.

f*string* *String* accepts floating-point numbers as values.

i*string* *String* accepts integers as values.

t*string* *String* is a two-letter option name that takes no value.

String is a one- or two-letter option name.

gd [GPS *files*] – GPS dump
gd prints a human readable listing of GPS.

gtop [**–r***n* **u**] [GPS *files*] – GPS to **plot**(4) filter
gtop transforms a GPS into **plot**(4) commands displayable by **plot** filters. GPS objects are translated if they fall within the window that circumscribes the first *file* unless an *option* is given. The options are as follows:

r*n* Translate objects in GPS region *n*.

u Translate all objects in the GPS universe.

pd [**plot**(5) *files*] – **plot**(4) dump
pd prints a human readable listing of **plot**(4) format graphical commands.

ptog [**plot**(5) *files*] – **plot**(4) to GPS filter
ptog transforms **plot**(4) commands into a GPS.

quit – Terminate session

remcom [*files*] – Remove comments
remcom copies its input to its output with comments removed. Comments are as defined in C (i.e., /* comment */).

whatis [**–o**] [*names*] – Brief on-line documentation
whatis prints a brief description of each *name* given. If no *name* is given, then the current list of description *names* is printed. The command **whatis *** prints out every description. The following option is available:

- **o** Just print command options

yoo *file* – Pipe fitting
yoo is a piping primitive that deposits the output of a pipeline into a *file* used in the pipeline. Note that without **yoo**, this is not usually successful as it causes a read and write on the same file simultaneously.

SEE ALSO
graphics(1G), gps(4), plot(4)

NAME

help – operating system Help Facility

SYNOPSIS

help
[**help**] **starter**
[**help**] **usage** [**–d**] [**–e**] [**–o**] [command_name]
[**help**] **locate** [keyword1 [keyword2] ...]
[**help**] **glossary** [term]
help arg ...

DESCRIPTION

The Help Facility provides on-line assistance for operating system users, whether they desire general information or specific assistance for use of the Source Code Control System (SCCS) commands.

Without arguments, **help** prints a menu of available on-line assistance commands with a short description of their functions. The commands and their descriptions are as follows:

Command	Description
starter	Operating system information for the beginning user
locate	Locate operating system commands using function-related keywords
usage	Operating system command usage information
glossary	Definitions of operating system technical terms

The user can choose one of the above commands by entering its corresponding letter (given in the menu) or can exit to the shell by typing q (for "quit").

With arguments, **help** directly invokes the named on-line assistance command, bypassing the initial **help** menu. The commands **starter**, **locate**, **usage**, and **glossary**, optionally preceded by the word **help**, may also be specified at shell level. When executing **glossary** from shell level, some of the symbols listed in the glossary must be escaped (preceded by one or more backslashes (\)) to be understood by the Help Facility. For a list of symbols and the required number of backslashes, refer to the **glossary**(1) manual page.

From any screen in the Help Facility, a user can execute a command via the shell by typing an exclamation mark (!) and the command to be executed. The screen will be redrawn if the command was entered at a first-level prompt. If entered at any other prompt level, only the prompt will be redrawn.

By default, the Help Facility scrolls the data that is presented to the user. If you prefer to have the screen clear before printing the data (non-scrolling), the shell variable **SCROLL** must be set to **no** and exported so it will become part of your environment. This is done by adding the following line to your *.profile* file (see **profile**(4)): "`export SCROLL ; SCROLL=no`". If you later decide that scrolling is desired, **SCROLL** must be set to **yes**.

Information on each of the Help Facility commands (**starter**, **locate**, **usage**, **glossary**, and **help**) is located on their respective manual pages.

The Help Facility can be tailored to a customer's needs by use of the **helpadm**(1M) command.

If the first argument to **help** is different from **starter, usage, locate,** or **glossary, help** assumes information is being requested about the SCCS Facility. The arguments can be either message numbers (which normally appear in parentheses following messages) or command names of the following types:

type1 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 (such as ge3 for message 3 from the **get** command).

type2 Does not contain numerics (a command, such as **get**).

type3 Is all numeric (such as 212).

SEE ALSO

admin(1), cdc(1), comb(1), csh(1), delta(1), get(1), glossary(1), helpadm(1M), ksh(1), locate(1), prs(1), rmdel(1), sact(1), sccsdiff(1), sh(1), starter(1), usage(1) unget(1), val(1), vc(1), what(1), profile(4), sccsfile(4), term(5)

WARNINGS

If the shell variable **TERM** is not set in the user's *.profile* file, then **TERM** will default to the terminal value type 450 (a hard-copy terminal). For a list of valid terminal types, refer to **term**(5).

NAME

helpadm – change the Help Facility database

SYNOPSIS

/etc/helpadm

DESCRIPTION

The Help Facility command, **helpadm**, allows system administrators and command developers to define the content of the Help Facility database for specific commands and to monitor use of the Help Facility. The **helpadm** command can only be executed by login **root**, login **bin**, or a login that is a member of group bin.

The **helpadm** command prints a menu of three types of Help Facility data which can be modified, and two choices relating to monitoring use of the Help Facility. The five choices are:

- Modify *startup* data
- Add, modify, or delete a *glossary* term
- Add, modify, or delete command data (description, options, examples, and keywords)
- Prevent monitoring use of the Help Facility (login **root** and login **bin** only)
- Permit monitoring use of the Help Facility (login **root** and login **bin** only)

The user may make one of the above choices by entering its corresponding letter (given in the menu), or may exit to the shell by typing q (for "quit").

If one of the first three choices is chosen, then the user is prompted for additional information; specifically, which *startup* screen, *glossary* term definition, or command description is to be modified. The user may also be prompted for information to identify whether the changes to the database are additions, modifications, or deletions. If the user is modifying existing data or adding new data, they are prompted to make the appropriate modifications or additions. If the user is deleting a *glossary* term or a command from the database, then they must respond affirmatively to the next query in order for the deletion to be done. In any case, before the user's changes are final, they must respond affirmatively when asked whether they are sure they want their requested database changes to be done.

By default, **helpadm** will put the user into **ed**(1) to make additions or modifications to database information. If the user wishes to be put into a different editor, then they should set the environment variable **EDITOR** in their environment to the desired editor and then export **EDITOR**.

If the user chooses to monitor/prevent monitoring use of the Help Facility, the choice made is acted on with no further interaction by the user.

FILES

HELPLOG	/usr/lib/help/HELPLOG
helpclean	/usr/lib/help/helpclean

SEE ALSO

ed(1), glossary(1), help(1), locate(1), starter(1), usage(1)

WARNINGS

When the operating system is delivered to a customer, */etc/profile* exports the

environment variable **LOGNAME**. If */etc/profile* has been changed so that **LOGNAME** is not exported, the options to monitor/prevent monitoring use of the Help Facility may not work properly.

NAME

hp – handle special functions of Hewlett-Packard terminals

SYNOPSIS

hp [-e] [-m]

DESCRIPTION

hp supports special functions of the Hewlett-Packard 2640 series of terminals, with the primary purpose of producing accurate representations of most **nroff** output. A typical usage is in conjunction with DOCUMENTER'S WORKBENCH Software:

nroff -h *files* ... **| hp**

Regardless of the hardware options on your terminal, **hp** tries to do sensible things with underlining and reverse line-feeds. If the terminal has the "display enhancements" feature, subscripts and superscripts can be indicated in distinct ways. If it has the "mathematical-symbol" feature, Greek and other special characters can be displayed.

The flags are as follows:

-e It is assumed that your terminal has the display enhancements feature and maximum use is made of the added display modes. Overstruck characters are presented in the Underline mode. Superscripts are shown in Half-bright mode, and subscripts in Half-bright, Underlined mode. If this flag is omitted, **hp** assumes that your terminal lacks the display enhancements feature. In this case, all overstruck characters, subscripts, and superscripts are displayed in Inverse Video mode (that is, dark-on-light, rather than the usual light-on-dark).

-m Requests minimization of output by removal of newlines. Any contiguous sequence of 3 or more newlines is converted into a sequence of only 2 newlines; that is, any number of successive blank lines produces only a single blank output line. This allows you to retain more actual text on the screen.

With regard to Greek and other special characters, **hp** provides the same set as does 300(1), except that "not" is approximated by a right arrow, and only the top half of the integral sign is shown.

SEE ALSO

300(1), greek(1)

col(1), eqn(1), nroff(1), tbl(1) in the *DOCUMENTER'S WORKBENCH Technical Discussion and Reference Manual*

DIAGNOSTICS

line too long The representation of a line exceeds 1024 characters.

The exit codes are 0 for normal termination, 2 for all errors.

BUGS

An overstriking sequence is defined as a printing character followed by a backspace followed by another printing character. In such sequences, if either printing character is an underscore, the other printing character is shown underlined or in Inverse Video. Otherwise, only the first printing character is shown (again, underlined or in Inverse Video). Nothing special is done if a backspace is adjacent to an ASCII control character. Sequences of control

characters (for example, reverse line-feeds, backspaces) can make text "disappear." In particular, tables generated by **tbl**(1) that contain vertical lines will often be missing the lines of text that contain the "foot" of a vertical line, unless the input to **hp** is piped through **col**(1).

Although some terminals do provide numerical superscript characters, no attempt is made to display them.

NAME

hpio – Hewlett-Packard 2645A terminal tape file archiver

SYNOPSIS

hpio –o[rc] file ...

hpio –i[rta] [**–n** count]

DESCRIPTION

hpio is designed to take advantage of the tape drives on Hewlett-Packard 2645A terminals. Up to 255 operating system files can be archived onto a tape cartridge for off-line storage or for transfer to another system. The actual number of files depends on the sizes of the files. One file of about 115,000 bytes will almost fill a tape cartridge. Almost 300 1-byte files will fit on a tape, but the terminal will not be able to retrieve files after the first 255. This manual page is not intended to be a guide for using tapes on Hewlett-Packard 2645A terminals, but tries to give enough information to enable a user to create and read tape archives and to position a tape for access to a desired file in an archive.

hpio –o (copy out) copies the specified *file*(s), together with pathname and status information, to a tape drive on your terminal (which is assumed to be positioned at the beginning of a tape or immediately after a tape mark). The left tape drive is used by default. Each *file* is written to a separate tape file and terminated with a tape mark. When **hpio** finishes, the tape is positioned following the last tape mark written.

hpio –i (copy in) extracts a file(s) from a tape drive (which is assumed to be positioned at the beginning of a file that was previously written by a **hpio –o**). The default action extracts the next file from the left tape drive.

hpio always leaves the tape positioned after the last file read from or written to the tape. Tapes should always be rewound before the terminal is turned off. To rewind a tape, depress the green function button, then function key 5, and then select the appropriate tape drive by depressing either function key 5 for the left tape drive or function key 6 for the right. If several files have been archived onto a tape, the tape can be positioned at the beginning of a specific file by depressing the green function button, then function key 8, followed by typing the desired file number (1–255) (don't press Return). Finally, press either function key 5 for the left tape or function key 6 for the right tape. The desired file number can also be specified by a signed number relative to the current file number.

The available options are as follows:

r Use the right tape drive.

c Include a checksum at the end of each *file*. **hpio –i** always checks the checksum for each file **hpio –o** writes with this option.

n *count* The number of input files to be extracted is set to *count*. If this option is not given, *count* defaults to 1. An arbitrarily large *count* may be specified to extract all files from the tape. **hpio** will stop at the end of data mark on the tape.

t Print a table of contents only. No files are created. Printed information gives the file size in bytes, the filename, the file access

modes, and whether a checksum is included for the file.

a Ask before creating a file. **hpio –i** normally prints the file size and name, creates and reads in the file, and prints a status message when the file has been read in. If a checksum is included with the file, it reports whether the checksum matched its computed value. With this option, the file size and name are printed followed by a **?**. Any response beginning with **y** or **Y** will cause the file to be copied in as above. Any other response will cause the file to be skipped.

FILES

/dev/tty??/???? To block messages while accessing a tape

SEE ALSO

cu(1C)

DIAGNOSTICS

BREAK
An interrupt signal terminated processing.

Can't create '*file*'.
Filesystem access permissions did not allow *file* to be created.

Can't get tty options on stdout.
hpio was unable to get the input-output control settings associated with the terminal.

Can't open '*file*'.
File could not be accessed to copy it to tape.

End of Tape.
No tape record was available when a read from a tape was requested. An end of data mark is the usual reason for this, but it may also occur if the wrong tape drive is being accessed and no tape is present.

'*file*' not a regular file.
File is a directory or other special file. Only regular files will be copied to tape.

Readcnt = *rc*, termcnt = *tc*.
hpio expected to read *rc* bytes from the next block on the tape, but the block contained *tc* bytes. This is caused by having the tape improperly positioned or by a tape block being mangled by interference from other terminal I/O.

Skip to next file failed.
An attempt to skip over a tape mark failed.

Tape mark write failed.
An attempt to write a tape mark at the end of a file failed.

Write failed.
A tape write failed. This is most frequently caused by specifying the wrong tape drive, running off the end of the tape, or trying to write on a tape that is write protected.

WARNINGS

Tape I/O operations may copy bad data if any other I/O involving the terminal occurs. Do not attempt any type ahead while **hpio** is running. **hpio** turns off

write permissions for other users while it is running, but processes started asynchronously from your terminal can still interfere. While a tape is being written, the most common indication of this problem is the appearance of characters on the display screen that should have been copied to tape.

The keyboard, including the terminal Break key, is locked during tape write operations. The Break key is functional only between writes.

hpio must have complete control of the attributes of the terminal to communicate with the tape drives. Interaction with commands such as **cu**(1C) may interfere and prevent successful operation.

BUGS

Some binary files contain sequences that will confuse the terminal.

An **hpio –i** that encounters the end-of-data mark on the tape (e.g., scanning the entire tape with **hpio –itn 300**), leaves the tape positioned after the end-of-data mark. If a subsequent **hpio –o** is done at this point, the data will not be retrievable. Before the **hpio –o** is started, the tape must be repositioned manually by using the terminal FIND FILE –1 operation (depress the green function button, then function key 8, and finally, function key 5 for the left tape or function key 6 for the right tape).

If an interrupt is received by **hpio** while a tape is being written, the terminal may be left with the keyboard locked. If this happens, the terminal's Reset Terminal key will unlock the keyboard.

NAME

icheck – filesystem storage consistency check

SYNOPSIS

/etc/icheck [**–b** numbers] [filesystem]

DESCRIPTION

icheck is obsoleted for normal consistency checking by **fsck**(1M).

icheck examines a filesystem, builds a bit map of used blocks, and compares this bit map against the free list maintained on the filesystem. If the filesystem is not specified, a set of default filesystems

Following the **–b** option is a list of block numbers. Whenever any of the named blocks turns up in a file, a diagnostic is produced.

icheck is faster if the raw version of the special file is used, since it reads the i-list many blocks at a time.

FILES

Default filesystems vary with installation.

SEE ALSO

clri(1M), dcheck(1M), fsck(1M), ncheck(1M), fs(4)

DIAGNOSTICS

For duplicate blocks and bad blocks (which lie outside the filesystem), **icheck** announces the difficulty, the i-number, and the kind of block involved. If a read error is encountered, the block number of the bad block is printed and **icheck** considers it to contain 0.

BUGS

Since **icheck** is inherently two-pass in nature, extraneous diagnostics may be produced if applied to active filesystems.

It believes even preposterous superblocks and consequently can get core images.

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

logname(1), getuid(2)

NAME

ident – identify files

SYNOPSIS

ident file ...

DESCRIPTION

ident searches the named files for all occurrences of the pattern *$keyword:...$*, where *keyword* is one of the following:

Author
Date
Header
Locker
Log
Revision
Source
State

These patterns are normally inserted automatically by the RCS command **co**(1), but can also be inserted manually.

ident works on text files as well as object files. For example, if the C program in file *f.c* contains the following:

```
#ident      "$Header:  Header information $"
```

and *f.c* is compiled into *f.o*, the command

ident f.c f.o

will print

```
f.c:
        $Header:  Header information $
f.o:
        $Header:  Header information $
```

SEE ALSO

ci(1), co(1), rcs(1), rcsdiff(1), rcsintro(1), rcsmerge(1), rlog(1), rcsfile(4)

NOTES

Copyright 1982 by Walter F. Tichy.

NAME

indicator – display application specific alarms and/or the working indicator

SYNOPSIS

indicator [**–c** column] [**–l** length] [**–o**] [**–w**] [**–b** [n]] "[string]" . . .

DESCRIPTION

The **–c** option dictates what column of the banner line to start the indicator string on. *column* is an integer from 0 to 79. If the **–c** option is not used, the default is 0.

The **–l** option limits the length of the indicator. If the string is longer than *length*, it will be truncated. *length* is an integer from 1 to 80. If **–l** is not used, the default is the entire string.

The **–o** option causes indicator to "tee" its output to *stdout*. The **–w** option turns on the working indicator. The **–b** option rings the terminal bell *n* times, where *n* is an integer from 1 to 10. The default is 1. If the terminal has no bell, the screen is flashed instead, if possible.

If the **indicator** command is being used solely for the bell or working indicator control, remember to give it a null string argument unless input is being piped to it. The string should always be the last argument given. The indicator is not automatically cleared.

EXAMPLES

When the value entered in the field is wrong, ring the bell 3 times and put up an indicator saying WRONG in column 1.

```
invalidmsg=`indicator -b 3 -c 1 "WRONG"`
```

This example clears the indicator after telling the user that the entry is wrong:

```
invalidmsg=`indicator -b 9 -c 1 "WRONG";
          indicator -c 1 "     "`
```

NAME

infocmp – compare or print out terminfo descriptions

SYNOPSIS

infocmp [**–d**] [**–c**] [**–n**] [**–I**] [**–L**] [**–C**] [**–r**] [**–u**] [**–s d | i | l | c**] [**–v**] [**–V**] [**–1**] [**–w** width] [**–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, followed by the string fields.

Default Options

If no options are specified and zero or one *termnames* are specified, the **–I** option is assumed. If more than one *termname* is specified, the **–d** option is 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 capability is defined for only one of the terminals, the value returned will depend on the type of the capability: **F** for boolean variables, **–1** 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 necessary 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 *termnames*. This can be used as a quick check to see if anything was left out of the description.

Source Listing Options [–I] [–L] [–C]

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 terminal 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 information 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 as follows:

Terminfo	Termcap	Representative Terminals
%p1%c	%.	adm
%p1%d	%d	hp, ANSI standard, vt100
%p1%'x'%+%c	%+x	concept
%i	%i	ANSI standard, vt100
%p1%?%'x'%>%t%p1%'y'%+%;	%>xy	concept
%p2 is printed before %p1	%r	hp

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 be 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 **tic**(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 in which the entries are given. **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.

Other Options [–s d|i|l|c] [–v] [–V]

–s Sort the fields within each type according to the argument below:

- **d** Leave fields in the order that they are stored in the *terminfo* database.
- **i** Sort by *terminfo* name.
- **l** 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.

–1 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 *termnames*. 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

SEE ALSO

captoinfo(1M), tic(1M), curses(3X), term(4), terminfo(4)

DIAGNOSTICS

malloc is out of space!

There was not enough memory available to process all terminal descriptions requested. Run **infocmp** several times, each time including a subset of the desired *termnames*.

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.

NOTES

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 [**0123456SsQqabc**]

/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)).

At any given time, the system is in one of eight possible run levels. A run level is a software configuration of the system under which only a selected group of processes exist. The processes spawned by **init** for each of these run levels is defined in */etc/inittab*. **init** can be in one of eight run levels, **0–6** and **S** or **s** (run levels **S** and **s** are identical). The run level changes when a privileged user runs **/etc/init**. This user-spawned **init** sends appropriate signals to the original **init** spawned by the operating system when the system was booted, telling it which run level to change to.

The following are the arguments to **init**.

0 Shut the machine down so it is safe to remove the power. Have the machine remove power if it can.

1 Put the system in single-user mode. Unmount all filesystems except root. All user processes are killed except those connected to the console.

2 Put the system in multiuser mode. All multiuser environment terminal processes and daemons are spawned. This state is commonly referred to as the multiuser state.

3 Is available to be defined as an alternative multiuser environment configuration. It is not necessary for system operation and is usually not used.

4 Is available to be defined as an alternative multiuser environment configuration. It is not necessary for system operation and is usually not used.

5 Stop the operating system and go to the firmware monitor.

6 Stop the operating system and reboot to the state defined by the *initdefault* entry in */etc/inittab*.

a,b,c Process only those */etc/inittab* entries having the **a**, **b** or **c** run level set. These are pseudostates that may be defined to run certain commands but that do not cause the current run level to change.

Q,q Re-examine */etc/inittab*.

S,s Enter single-user mode. When this occurs, a shell prompt will only appear on the system console. This is the only run level that doesn't require the existence of a properly formatted */etc/inittab* file. If this file does not exist, then by default the only legal run level that **init** can enter is the single-user mode. When the system enters **S** or **s**, all mounted filesystems remain mounted and only processes spawned by

init are killed.

When the system is booted, **init** is invoked and the following occurs. 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 there is no *initdefault* entry in */etc/inittab*, **init** requests that the user enter a run level from the system console. If an **S** or **s** is entered, **init** goes to the single-user state. In the single-user state the system console terminal is opened for reading and writing. The command **/bin/su** is invoked which in turn executes the shell for the user root as described in the */etc/passwd* file. 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 the */etc/inittab* file does not exist.

If a **0** through **6** is entered, **init** enters the corresponding run level. Note that 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 since power up that **init** has entered a run level other than single-user state, **init** first scans */etc/inittab* for *boot* and *bootwait* entries (see **inittab**(4)). These entries are performed before any other processing of */etc/inittab* takes place, providing that the run level entered matches that of the entry. In this way, any special initialization of the operating system such as mounting filesystems can take place before users are allowed onto the system. **init** then scans */etc/inittab* and executes all other entries that are to be processed for that run level.

In a multiuser environment, */etc/inittab* is set up so that **init** will create a **getty** process for each terminal that the administrator sets up to respawn.

To spawn each process in */etc/inittab*, **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 */etc/inittab*, **init** waits for one of its descendant processes to die, for a powerfail signal, or for a signal from another **init** or **telinit** process to change the system's run level. When one of these conditions occurs, **init** re-examines */etc/inittab*. New entries can be added to */etc/inittab* at any time; however, **init** still waits for one of the above three conditions to occur before re-examining */etc/inittab*. To get around this, **init Q** or **init q** command wakes **init** to re-examine */etc/inittab* immediately.

When **init** comes up at boot time and whenever the system changes from the single-user state to another run state, **init** sets the **ioctl**(2) states of the system console to the default modes.

When a run level change request is made, **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).

The shell running on each terminal will terminate when the user types an end-of-file or hangs 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/utmp* and */etc/wtmp* if it exists (see **who**(1)). A history of the processes spawned is kept in */etc/wtmp*.

If **init** receives a *powerfail* signal (SIGPWR) it scans */etc/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 states (**S** and **s**), only *powerfail* and *powerwait* entries are executed.

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** to take the appropriate action.

FILES

/etc/inittab
/etc/utmp
/etc/wtmp
/dev/syscon

SEE ALSO

login(1), sh(1), stty(1), su(1), who(1), getty(1M), shutdown(1M), kill(2), gettydefs(4), passwd(4), inittab(4), utmp(4), console(7), termio(7)

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 */etc/inittab*.

WARNINGS

init and **telinit** can be run only by someone who is superuser.

The **S** or **s** state must not be used indiscriminately in the */etc/inittab* file. A good rule to follow when modifying this file is to avoid adding this state to any line other than the *initdefault*.

The change to */etc/gettydefs* described in the **WARNINGS** section of the **gettydefs**(4) manual page will permit terminals to pass 8 bits to the system as long as the system is in multiuser state (run level greater than 1). When the system changes to single-user state, the **getty** is killed and the terminal attributes are lost. To permit a terminal to pass 8 bits to the system in single-user state, after you are in single-user state, type this command:

stty –istrip cs8

NAME

install – install commands

SYNOPSIS

/etc/install [**–c** dira] [**–f** dirb] [**–i**] [**–n** dirc] [**–m** mode] [**–u** user] [**–g** group] [**–o**] [**–s**] file [dirx ...]

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 filesystem. 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 as follows:

–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*. Only available to the superuser.

–u *user* — The owner of the new file is set to *user*. Only available to the superuser.

–g *group* — The group ID of the new file is set to *group*. Only available to the superuser.

–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

installboot – installs a boot program on a disk

SYNOPSIS

/etc/installboot disk

DESCRIPTION

installboot either determines the type of bootstrap that should be installed on *disk* and then installs the proper bootstrap in the bootstrap area of the disk or installs the system controller firmware in the firmware area of the disk.

The *disk* argument does not need to be a special file in */dev*, but can simply be the controller type concatenated with the disk number (e.g., **sd0**). **installboot** uses the *disk* argument to determine the disk type.

For disks booting from the SCED, **installboot** reads the VTOC from the disk and searches for a boot partition. (This is a partition of type **V_BOOT**, as defined in */usr/include/sys/vtoc.h*.) **installboot** then opens the first boot partition and writes the proper bootstrap to it. The bootstrap is taken from */stand/boot***XX**, where **XX** corresponds to the disk controller name.

For disks booting from the SSM, **installboot** reads the VTOC from the disk and searches for a firmware partition. (This is a partition of type **V_FW**, as defined in */usr/include/sys/vtoc.h*.) **installboot** then opens the first firmware partition and writes the firmware to it. The firmware is taken from */ssm/fw/lvl2*.

Because of the way **installboot** works, the disk on which the bootstrap or firmware is installed must have a VTOC present. If it does not, add one by using **mkvtoc**(1M). There should be only one partition of type **V_BOOT** and it should cover the bootstrap area for this particular disk. Also, there should be at most one partition of type **V_FW** and it should be large enough to hold the system controller firmware.

EXAMPLES

The following line will install a SCED bootstrap on **zd0.**

```
installboot zd0
```

FILES

/stand/boot??	SCED boot programs
/ssm/fw/lvl2	SSM Level 2 Firmware

SEE ALSO

mkvtoc(1M), prtvtoc(1M), vtoc(7)

NAME

ipcrm – remove a message queue, semaphore set, or shared memory ID

SYNOPSIS

ipcrm [*options*]

DESCRIPTION

ipcrm will remove one or more specified messages, semaphores, or shared memory identifiers. The identifiers are specified by the following *options*:

–q *msqid* Removes the message queue identifier *msqid* from the system and destroys the message queue and data structure associated with it.

–m *shmid* Removes the shared memory identifier *shmid* from the system. The shared memory segment and data structure associated with it are destroyed after the last detach.

–s *semid* Removes the semaphore identifier *semid* from the system and destroys the set of semaphores and data structure associated with it.

–Q *msgkey* Removes the message queue identifier (created with key *msgkey*) from the system and destroys the message queue and data structure associated with it.

–M *shmkey* Removes the shared memory identifier (created with key *shmkey*) from the system. The shared memory segment and data structure associated with it are destroyed after the last detach.

–S *semkey* Removes the semaphore identifier (created with key *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). Use **ipcs**(1) to find the identifiers and keys.

SEE ALSO

ipcs(1) msgctl(2), msgget(2), msgop(2), semctl(2), semget(2), semop(2), shmctl(2), shmget(2), shmop(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 displays 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 options will print. If none of these options are specified, information about all three options will print subject to the following 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, process ID of last process to receive a message on message queues, 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*
Use the argument *namelist* in place of */unix*.

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 determine only the information 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 **shmget** 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 characters that are interpreted as described below. The first two characters are as follows:

- **R** If a process is waiting on a **msgrcv**.
- **S** If a process is waiting on a **msgsnd**.
- **D** If the associated shared memory segment has been removed. It 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 permissions of others in the user-group of the facility entry, and the last to all others. Within each set, the first character indicates permission to read, the second character indicates permission to write or alter the facility entry, and the last character is currently unused.

The permissions are indicated as follows:

- **r** Read permission is granted.
- **w** Write permission is granted.
- **a** Alter permission is granted.
- **-** The indicated permission is not granted.

OWNER (all)
The login name of the owner of the facility entry.

GROUP (all)
The name of the group for the owner of the facility entry.

CREATOR (a,c)
The login name of the creator of the facility entry.

CGROUP (a,c)
The name of the group for 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 associated 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.

NATTCH (a,o)
The number of processes attached to the associated shared memory segment.

SEGSZ (a,b)
The size of the associated shared memory segment.

CPID (a,p)
The process ID of the creator of the shared memory entry.

LPID (a,p)
The process ID of the last process to attach or detach the shared memory segment.

ATIME (a,t)
The time the last attach was completed to the associated shared memory segment.

DTIME (a,t)
The time the last detach was completed on the associated shared memory segment.

NSEMS (a,b)
The number of semaphores in the set associated with the semaphore entry.

OTIME (a,t)
The time the last semaphore operation was completed on the set associated with the semaphore entry.

FILES

/unix	System namelist
/dev/kmem	Memory
/etc/passwd	User names
/etc/group	Group names

SEE ALSO

msgop(2), semop(2), shmop(2)

WARNINGS

If the user specifies either the **–C** or the **–N** flag, the real and effective UID/GID will be set to the real UID/GID of the user invoking **ipcs**.

BUGS

Things can change while **ipcs** is running. The picture it gives is only a close approximation to reality.

NAME

isismgr – ISIS file manager, updates text files in ISIS file format

SYNOPSIS

isismgr **–m** *r* [**–O** *fldnum,fldnum...*] [**–s** *outsep*] **–k** *keyval,keyval,...*
–K *keynum,keynum,...* **–f** *fldseps* [**–r** *recsep*] [**–e** *escchar*]
[**–c**/**–C** *commchar*] [**–v**] [**–l**] *file(s)*

isismgr **–m** *a* **–i** *image* **–k** *keyval,keyval,...* **–K** *keynum,keynum,...*
–f *fldseps* [**–r** *recsep*] [**–e** *escchar*] [**–c**/**–C** *commchar*] [**–v**] [**–l**] *file(s)*

isismgr **–m** *c* [[**–i** *image*] or [**–u** *updval,updval...* **–U** *updnum,updnum...*]]
–k *keyval,keyval,...* **–K** *keynum,keynum,...* **–f** *fldseps* [**–r** *recsep*]
[**–e** *escchar*] [**–c**/**–C** *commchar*] [**–v**] [**–l**] *file(s)*

isismgr **–m** *d* **–k** *keyval,keyval,...* **–K** *keynum,keynum,...* **–f** *fldseps*
[**–r** *recsep*] [**–e** *escchar*] [**–c**/**–C** *commchar*] [**–v**] [**–l**] *file(s)*

DESCRIPTION

The command **isismgr** updates records in ASCII file(s) that contain a single type of record. Each field in a record is delimited by a known field separator and all records are delimited by a single record separator. Blank lines and comments may appear in the file.

isismgr identifies records that must be updated by matching against given key value(s). Duplicate records are ignored.

The command line options are as follows:

–m *mode* Action that is taken when updating a single record in the file. *mode* is **r** (read), **a** (add), **c** (change), or **d** (delete). Entry of a mode is required.

–i *image* The complete image of the record (including its field separator characters) that is added to or changed in the file. *image* may contain escape characters. Entry of a record image is required for add mode only and is optional for change mode.

–k *keyval,keyval,...*
A list of key values that are used to search for the record that must be updated. Each value is matched against the fields given by the **–K** option. Entry of a key value list is required.

–K *keynum,keynum,...*
A list of field numbers (starting with 0) that designates the field positions that will be matched against the parallel key value list given by the **–k** option. Entry of a key field position list is required.

–u *updval,updval,...*
A list of values used to update specific fields in the record. Each

value is matched against, and then replaces, the fields given by the **–U** option. When you use the change mode, you can enter an update value list instead of a record image (**–i** option).

–U *updnum,updnum,...*
A list of field numbers (starting with 0) that tells the system the field positions that will be matched against the parallel update value list given by the **–u** option. Entry of an update field position list is required when using the **–u** option.

–O *fldnum,fldnum,...*
A list of field numbers (starting with 0) that tells the system the fields that are written on the standard output. Each of these fields is followed by its delimiter in the record. Entry of an output field position list is optional with read mode.

–s *outsep* Alternate character used to delimit fields displayed when using read mode. An escape character may be given. Entry of an alternate separator character is optional with read mode.

–f *fldseps* A string of characters used to describe the separators that delimit fields within a record. The entries in the string are order dependent and are of the form *xcxcxc...* where *x* is an optional list of codes that describe the characteristics of the field separator *c*. These codes are listed below:

- W*n* The minimum width of the field in *n* characters
- L Leading whitespace for the field is ignored
- R The field separator **c** is repeated, and will delimit all remaining fields in the record

Example: "::L-" describes a record in the following format:

1st field	Delimited by a colon
2nd field	Delimited by a colon
3rd field	Delimited by a dash and leading whitespace is ignored
4th field	Delimited by the record sep

Example: ":LW5:L R," describes a record in the following format:

1st field	Delimited by a colon
2nd field	Has a minimum width of 5, leading whitespace is ignored, and is delimited by a colon
3rd field	Delimited by whitespace and leading whitespace is ignored
4th field	Delimited by a comma, and, since it is repeated, fields 5 through N are also delimited by a comma

Field separators can contain escape sequences to represent all characters in the ASCII set. Escape sequences are \b (backspace), \f (formfeed), \n (newline), \r (return), \t (tab), \v (vertical-tab),

and \; (backslash). The sequence \nnn (where nnn is an octal number) can also be used to represent characters in the ASCII set. Entry of a field separator description is required.

–r *recsep* Alternate character used to delimit records. The default record separator is a newline. An escape character can be given. Entry of a record separator character is optional.

–e *escchar* When records are read from a file, this character is recognized as an escape. Any character preceded by it is treated as a literal character and does not delimit a field. The escape character is most often embedded in a record to facilitate the insertion of field separator and comment characters. It may also appear as the last character in a record, indicating the record is continued on the next line in the file. An escape sequence can be given. Entry of a escape character is optional.

–c *commchar*

Character recognized as the beginning of a comment. All text between the comment character and the end of the record is ignored. An escape sequence may be given. Entry of a comment character is optional.

–C *commchar*

The –C option can be used in place of the –c option above. The comment character is recognized as such only if it appears as the first character in a record. An escape sequence can be given. Entry of a comment character is optional.

–l The lock option tells the system to ensure that another update does not occur at the same time. The system checks the file for the presence of a lock by another **isismgr** process. If the system discovers that the file is locked, the update is abandoned and the system displays a message on the screen to advise the user. The selection of file locking is optional.

–v The verbose option tells the system to notify the user regarding the success or failure of the update. The selection of verbose mode is optional.

EXAMPLES

Begin with a simple file named *tfile* containing the following data:

```
one:1::uno
two:2::dos
four:4: some text:quatro
```

A record is added to the end of the file using the following command:

```
isismgr -m a -i "three:3::tres" -k three -K 0 -f "::W:" tfile
```

The record containing the key "dos" is changed with the following command:

```
isismgr -m c -i "two:4::dos" -k 2 -K 1 -f "R:" -v tfile
```

The file now appears as follows:

```
one:1::uno
two:4::dos
```

```
four:4: some text:quatro
three:3::tres
```

The following command deletes the record containing the key components "4" and "quatro":

isismgr -m d -k 4,quatro -K 1,3 -f "R:" -v -l tfile

The following is the final form of the file:

```
one:1::uno
two:4::dos
three:3::tres
```

NOTES

The following escape characters are supported in several of the options:

Backspace	\b
Form Feed	\f
New Line	\n
Horizontal Tab	\t
Vertical Tab	\v
Backslash	\

You must have write permission on the file you are updating as well as write permission in the directory where the file will reside.

Do NOT make this a *setuid* program.

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 newline. 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:

–a*n* In addition to the normal output, produce a line for each unpairable line in file *n*, where *n* is 1 or 2.

–e *s* Replace empty output fields by string *s*.

–j*n m* Join on the *m*th field of file *n*. If *n* is missing, use the *m*th 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.

–t*c* 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.

EXAMPLES

The following command line joins 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 are 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** 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 superuser.

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 filesystems 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

kill(1), ps(1), fuser(1M), shutdown(1M) signal(2)

NOTES

The **killall** command can be run only by the superuser.

NAME

ksh, rksh – KornShell, a standard/restricted command and programming language

SYNOPSIS

ksh [**±aefhikmnoprstuvx**] [**±o** option] ... [**–c** string] [arg ...]
rksh [**±aefhikmnoprstuvx**] [**±o** option] ... [**–c** string] [arg ...]

DESCRIPTION

ksh is a command and programming language that executes commands read from a terminal or a file. **rksh** is a restricted version of the command interpreter **ksh**; 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 *metacharacter* is one of the following characters:

; & ( ) | < > new-line space tab

A *blank* is a **tab** or a **space**. An *identifier* is a sequence of letters, digits, or underscores starting with a letter or underscore. Identifiers are used as names for *functions* and *named parameters*. A *word* is a sequence of *characters* separated by one or more non-quoted *metacharacters*.

A *command* is a sequence of characters in the syntax of the shell language. The shell reads each command and carries out the desired action either directly or by invoking separate utilities. A special command is a command that is carried out by the shell without creating a separate process. Except for documented side effects, most special commands can be implemented as separate utilities.

Commands

A *simple-command* is a sequence of *blank* separated words which may be preceded by a parameter assignment list. (See *Environment* below). 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 five 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 asynchronous execution of the preceding command or pipeline with a two-way pipe established to the parent shell. The standard input and output of the spawned command can be written to and

read from by the parent shell using the **-p** option of the special commands **read** and **print** described later. The symbol **&&** (**| |**) causes the *list* following it to be executed only if the preceding pipeline returns a zero (non-zero) value. An arbitrary number of new-lines may appear in a *list,* instead of a semicolon, to delimit a command.

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 *identifier* [**in** *word* ...] **;do** *list* **;done**
Each time a **for** command is executed, *identifier* 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.

select *identifier* [**in** *word* ...] **;do** *list* **;done**
A **select** command prints on standard error (file descriptor 2), the set of *word*s, each preceded by a number. If **in** *word* ... is omitted, then the positional parameters are used instead (see *Parameter Substitution* below). The **PS3** prompt is printed and a line is read from the standard input. If this line consists of the number of one of the listed *word*s, then the value of the parameter *identifier* is set to the *word* corresponding to this number. If this line is empty the selection list is printed again. Otherwise the value of the parameter *identifier* is set to *null*. The contents of the line read from standard input is saved in the parameter **REPLY.** The *list* is executed for each selection until a **break** or *end-of-file* is encountered.

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 *Filename Generation* below).

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**
until *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 separate environment. Note, that if two adjacent open parentheses are needed for nesting, a space must be inserted to avoid arithmetic evaluation as described below.

{ *list*;}
list is simply executed. Note that unlike the metacharacters (and), { and } are *reserved words* and must at the beginning of a line or after a ; in order to be recognized.

[[*expression*]]
Evaluates *expression* and returns a zero exit status when *expression* is true. See *Conditional Expressions* below, for a description of *expression*.

function *identifier* { *list* ;}
identifier () { *list* ;}
Define a function which is referenced by *identifier*. The body of the function is the *list* of commands between { and }. (See *Functions* below).

time *pipeline*
The *pipeline* is executed and the elapsed time as well as the user and system time are printed on standard error.

The following reserved 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 { } function select time [[ ]]

Comments
A word beginning with # causes that word and all the following characters up to a new-line to be ignored.

Aliasing
The first word of each command is replaced by the text of an **alias** if an **alias** for this word has been defined. The first character of an **alias** name can be any non-special printable character, but the rest of the characters must be the same as for a valid *identifier*. The replacement string can contain any valid shell script including the metacharacters listed above. The first word of each command in the replaced text, other than any that are in the process of being replaced, will be tested for aliases. If the last character of the alias value is a *blank* then the word following the alias will also be checked for alias substitution. Aliases can be used to redefine special builtin commands but cannot be used to redefine the reserved words listed above. Aliases can be created, listed, and exported with the **alias** command and can be removed with the **unalias** command. Exported aliases remain in effect for scripts invoked by name, but must be reinitialized for separate invocations of the shell (See *Invocation* below).

Aliasing is performed when scripts are read, not while they are executed. Therefore, for an alias to take effect the **alias** definition command has to be executed before the command which references the alias is read.

Aliases are frequently used as a short hand for full path names. An option to the aliasing facility allows the value of the alias to be automatically set to the full pathname of the corresponding command. These aliases are called *tracked* aliases. The value of a *tracked* alias is defined the first time the corresponding command is looked up and becomes undefined each time the **PATH** variable is reset. These aliases remain *tracked* so that the next subsequent

reference will redefine the value. Several tracked aliases are compiled into the shell. The **-h** option of the **set** command makes each referenced command name into a tracked alias.

The following *exported aliases* are compiled into the shell but can be unset or redefined:

```
autoload='typeset -fu'
false='let 0'
functions='typeset -f'
hash='alias -t'
history='fc -l'
integer='typeset -i'
nohup='nohup '
r='fc -e -'
true=':'
type='whence -v'
```

Tilde Substitution

After alias substitution is performed, each word is checked to see if it begins with an unquoted ~. If it does, then the word up to a / is checked to see if it matches a user name in the */etc/passwd* file. If a match is found, the ~ and the matched login name is replaced by the login directory of the matched user. This is called a *tilde* substitution. If no match is found, the original text is left unchanged. A ~ by itself, or in front of a /, is replaced by the value of the **HOME** parameter. A ~ followed by a + or - is replaced by **$PWD** and **$OLDPWD** respectively.

In addition, *tilde* substitution is attempted when the value of a *variable assignment parameter* begins with a ~.

Command Substitution

The standard output from a command enclosed in parenthesis preceded by a dollar sign (**$()**) or a pair of grave accents (``) may be used as part or all of a word; trailing new-lines are removed. In the second (archaic) form, the string between the quotes is processed for special quoting characters before the command is executed. (See *Quoting* below). The command substitution **$(cat file)** can be replaced by the equivalent but faster **$(<file)**. Command substitution of most special commands that do not perform input/output redirection are carried out without creating a separate process.

An arithmetic expression enclosed in double parenthesis preceded by a dollar sign (**$(())**) is replaced by the value of the arithmetic expression within the double parenthesis.

Parameter Substitution

A *parameter* is an *identifier*, one or more digits, or any of the characters *, @, #, ?, -, $, and !. A *named parameter* (a parameter denoted by an identifier) has a *value* and zero or more *attributes*. *Named parameters* can be assigned *values* and *attributes* by using the **typeset** special command. The attributes supported by the shell are described later with the **typeset** special command. Exported parameters pass values and attributes to the environment.

The shell supports a one-dimensional array facility. An element of an array parameter is referenced by a *subscript*. A *subscript* is denoted by a [, followed by an *arithmetic expression* (see *Arithmetic Evaluation* below) followed by a].

To assign values to an array, use **set –A** *name value* .... The value of all subscripts must be in the range of 0 through 1023. Arrays need not be declared. Any reference to a named parameter with a valid subscript is legal and an array will be created if necessary. Referencing an array without a subscript is equivalent to referencing the element zero.

The *value* of a *named parameter* may also be assigned by writing:

name=value [*name=value*] ...

If the integer attribute, **–i**, is set for *name* the *value* is subject to arithmetic evaluation as described below.

Positional parameters, parameters denoted by a number, may be assigned values with the **set** special command. Parameter **$0** is set from argument zero when the shell is invoked.

The character **$** is used to introduce substitutable *parameters*.

${*parameter***}**

The shell reads all the characters from **${** to the matching **}** as part of the same word even if it contains braces or metacharacters. The value, if any, of the parameter is substituted. The braces are required when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name or when a named parameter is subscripted. If *parameter* is one or more digits then it is a positional parameter. A positional parameter of more than one digit must be enclosed in braces. If *parameter* is * or @, then all the positional parameters, starting with **$1**, are substituted (separated by a field separator character). If an array *identifier* with subscript * or @ is used, then the value for each of the elements is substituted (separated by a field separator character).

${#*parameter***}**

If *parameter* is * or @, the number of positional parameters is substituted. Otherwise, the length of the value of the *parameter* is substituted.

${#*identifier***[*]}**

The number of elements in the array *identifier* is substituted.

${*parameter***:–***word***}**

If *parameter* is set and is non-null then substitute its value; otherwise, substitute *word*.

${*parameter***:=***word***}**

If *parameter* is not set or is null then set it to *word*; the value of the parameter is then substituted. Positional parameters may not be assigned to in this way.

${*parameter***:?***word***}**

If *parameter* is set and is non-null then substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted then a standard message is printed.

${*parameter***:+***word***}**

If *parameter* is set and is non-null then substitute *word*; otherwise, substitute nothing.

${*parameter***#***pattern***}**
${*parameter***##***pattern***}**
If the shell *pattern* matches the beginning of the value of *parameter*, then the value of this substitution is the value of the *parameter* with the matched portion deleted; otherwise, the value of this *parameter* is substituted. In the first form the smallest matching pattern is deleted and in the second form the largest matching pattern is deleted.

${*parameter***%***pattern***}**
${*parameter***%%***pattern***}**
If the shell *pattern* matches the end of the value of *parameter*, then the value of this substitution is the value of the *parameter* with the matched part deleted; otherwise, substitute the value of *parameter*. In the first form the smallest matching pattern is deleted and in the second form the largest matching pattern is deleted.

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:

```
echo ${d:-$(pwd)}
```

If the colon (:) is omitted from the above expressions, then 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 executed command.
$	The process number of this shell.
_	Initially, the value _ is an absolute pathname of the shell or script being executed as passed in the *environment*. Subsequently it is assigned the last argument of the previous command. This parameter is not set for commands which are asynchronous. This parameter is also used to hold the name of the matching **MAIL** file when checking for mail.
!	The process number of the last background command invoked.
ERRNO	The value of *errno* as set by the most recently failed system call. This value is system dependent and is intended for debugging purposes.
LINENO	The line number of the current line within the script or function being executed.
OLDPWD	The previous working directory set by the **cd** command.
OPTARG	The value of the last option argument processed by the **getopts** special command.
OPTIND	The index of the last option argument processed by the **getopts** special command.
PPID	The process number of the parent of the shell.
PWD	The present working directory set by the **cd** command.
RANDOM	Each time this parameter is referenced, a random integer, uniformly distributed between 0 and 32767, is generated. The sequence of random numbers can be initialized by assigning a

numeric value to **RANDOM**.

REPLY This parameter is set by the **select** statement and by the **read** special command when no arguments are supplied.

SECONDS Each time this parameter is referenced, the number of seconds since shell invocation is returned. If this parameter is assigned a value, then the value returned upon reference will be the value that was assigned plus the number of seconds since the assignment.

The following parameters are used by the shell:

CDPATH The search path for the *cd* command.

COLUMNS If this variable is set, the value is used to define the width of the edit window for the shell edit modes and for printing **select** lists.

EDITOR If the value of this variable ends in **emacs**, **gmacs**, or **vi** and the **VISUAL** variable is not set, then the corresponding option (see Special Command **set** below) will be turned on.

ENV If this parameter is set, then parameter substitution is performed on the value to generate the pathname of the script that will be executed when the *shell* is invoked. (See *Invocation* below.) This file is typically used for *alias* and *function* definitions.

FCEDIT The default editor name for the **fc** command.

FPATH The search path for function definitions. This path is searched when a function with the **–u** attribute is referenced and when a command is not found. If an executable file is found, then it is read and executed in the current environment.

IFS Internal field separators, normally **space**, **tab**, and **new-line** that are used to separate command words which result from command or parameter substitution and for separating words with the special command **read**. The first character of the **IFS** parameter is used to separate arguments for the "**$***" substitution (See *Quoting* below).

HISTFILE If this parameter is set when the shell is invoked, then the value is the pathname of the file that will be used to store the command history. (See *Command re-entry* below.)

HISTSIZE If this parameter is set when the shell is invoked, then the number of previously entered commands that are accessible by this shell will be greater than or equal to this number. The default is 128.

HOME The default argument (home directory) for the **cd** command.

LINES If this variable is set, the value is used to determine the column length for printing **select** lists. Select lists will print vertically until about two-thirds of **LINES** lines are filled.

MAIL If this parameter is set to the name of a mail file *and* the **MAILPATH** parameter is not set, then the shell informs the user of arrival of mail in the specified file.

MAILCHECK This variable specifies how often (in seconds) the shell will check for changes in the modification time of any of the files specified by the **MAILPATH** or **MAIL** parameters. The default value is 600 seconds. When the time has elapsed the shell will

check before issuing the next prompt.

MAILPATH A colon (:) separated list of file names. If this parameter is set then the shell informs the user of any modifications to the specified files that have occurred within the last **MAILCHECK** seconds. Each file name can be followed by a **?** and a message that will be printed. The message will undergo parameter substitution with the parameter, **$_** defined as the name of the file that has changed. The default message is *you have mail in $_*.

PATH The search path for commands (see *Execution* below). The user may not change **PATH** if executing under **rksh** (except in *.profile*).

PS1 The value of this parameter is expanded for parameter substitution to define the primary prompt string which by default is "**$** ". The character **!** in the primary prompt string is replaced by the *command* number (see *Command Re-entry* below).

PS2 Secondary prompt string, by default "**>** ".

PS3 Selection prompt string used within a **select** loop, by default "**#?** ".

PS4 The value of this parameter is expanded for parameter substitution and precedes each line of an execution trace. If omitted, the execution trace prompt is "**+** ".

SHELL The pathname of the *shell* is kept in the environment. At invocation, if the basename of this variable matches the pattern ***r*sh**, then the shell becomes restricted.

TMOUT If set to a value greater than zero, the shell will terminate if a command is not entered within the prescribed number of seconds after issuing the **PS1** prompt. (Note that the shell can be compiled with a maximum bound for this value which cannot be exceeded.)

VISUAL If the value of this variable ends in **emacs**, **gmacs**, or *vi* then the corresponding option (see Special Command **set** below) is turned on.

The shell gives default values to **PATH**, **PS1**, **PS2**, **MAILCHECK**, **TMOUT** and **IFS**, while **HOME**, **SHELL ENV** and **MAIL** are not set at all by the shell (although **HOME** *is* set by **login**(1)). On some systems **MAIL** and **SHELL** are also set by **login**(1)).

Blank Interpretation

After parameter and command substitution, the results of substitutions are scanned for the 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.

Filename Generation

Following substitution, each command *word* is scanned for the characters *****, **?**, and **[** unless the **–f** option has been **set**. If one of these characters appears then the word is regarded as a *pattern*. The word is replaced with lexicographically sorted file names that match the pattern. If no file name is found that matches the pattern, then the word is left unchanged. When a *pattern* is used for file name generation, the character **.** at the start of a file name or immediately following a **/**, as well as the character **/** itself, must be matched

explicitly. In other instances of pattern matching the / and . are not treated specially.

*	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 "! " then any character not enclosed is matched. A – can be included in the character set by putting it as the first or last character.

A *pattern-list* is a list of one or more patterns separated by each other with a |. Composite patterns can be formed with one or more of the following:

?(*pattern-list*)	Optionally matches any one of the given patterns.
*(*pattern-list*)	Matches zero or more occurrences of the given patterns.
+(*pattern-list*)	Matches one or more occurrences of the given patterns.
@(*pattern-list*)	Matches exactly one of the given patterns.
!(*pattern-list*)	Matches anything, except one of the given patterns.

Quoting

Each of the *metacharacters* listed above (See *Definitions* above) has a special meaning to the shell and causes termination of a word unless quoted. A character may be *quoted* (i.e., made to stand for itself) by preceding it with a \. The pair **\new-line** is ignored. All characters enclosed between a pair of single quote marks (''), are quoted. A single quote cannot appear within single quotes. Inside double quote marks (" "), parameter and command substitution occurs and \ quotes the characters \, `, ", and $. The meaning of **$*** and **$@** is identical when not quoted or when used as a parameter assignment value or as a file name. However, when used as a command argument, **"$*"** is equivalent to **"$1***d***$2***d*...", where *d* is the first character of the **IFS** parameter, whereas **"$@"** is equivalent to **"$1" "$2"** .... Inside grave quote marks (``) \ quotes the characters \, `, and $. If the grave quotes occur within double quotes then \ also quotes the character ".

The special meaning of reserved words or aliases can be removed by quoting any character of the reserved word. The recognition of function names or special command names listed below cannot be altered by quoting them.

Arithmetic Evaluation

An ability to perform integer arithmetic is provided with the special command **let**. Evaluations are performed using *long* arithmetic. Constants are of the form [*base*#]*n* where *base* is a decimal number between two and thirty-six representing the arithmetic base and *n* is a number in that base. If *base* is omitted, base 10 is used.

An arithmetic expression uses the same syntax, precedence, and associativity of expression of the C language. All the integral operators, other than ++, --, ?:, and , are supported. Named parameters can be referenced by name within an arithmetic expression without using the parameter substitution syntax. When a named parameter is referenced, its value is evaluated as an arithmetic expression.

An internal integer representation of a *named parameter* can be specified with the **–i** option of the **typeset** special command. Arithmetic evaluation is performed on the value of each assignment to a named parameter with the **–i** attribute. If you do not specify an arithmetic base, the first assignment to the parameter determines the arithmetic base. This base is used when parameter substitution occurs.

Since many of the arithmetic operators require quoting, an alternative form of the **let** command is provided. For any command which begins with a ((, all the characters until a matching)) are treated as a quoted expression. More precisely, ((...)) is equivalent to **let "..."**.

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, then the secondary prompt (i.e., the value of **PS2**) is issued.

Conditional Expressions

A *conditional expression* is used with the [[compound command to test attributes of files and to compare strings. Word splitting and file name generation are not performed on the words between [[and]]. Each expression can be constructed from one or more of the following unary or binary expressions:

–a *file*	True, if *file* exists.
–b *file*	True, if *file* exists and is a block special file.
–c *file*	True, if *file* exists and is a character special file.
–d *file*	True, if *file* exists and is a directory.
–f *file*	True, if *file* exists and is an ordinary file.
–g *file*	True, if *file* exists and is has its setgid bit set.
–k *file*	True, if *file* exists and is has its sticky bit set.
–n *string*	True, if length of *string* is non-zero.
–o *option*	True, if option named *option* is on.
–p *file*	True, if *file* exists and is a fifo special file or a pipe.
–r *file*	True, if *file* exists and is readable by current process.
–s *file*	True, if *file* exists and has size greater than zero.
–t *fildes*	True, if file descriptor number *fildes* is open and associated with a terminal device.
–u *file*	True, if *file* exists and is has its setuid bit set.
–w *file*	True, if *file* exists and is writable by current process.
–x *file*	True, if *file* exists and is executable by current process. If *file* exists and is a directory, then the current process has permission to search in the directory.
–z *string*	True, if length of *string* is zero.
–L *file*	True, if *file* exists and is a symbolic link.
–O *file*	True, if *file* exists and is owned by the effective user ID of this process.
–G *file*	True, if *file* exists and its group matches the effective group ID of this process.
–S *file*	True, if *file* exists and is a socket.
file1 **–nt** *file2*	True, if *file1* exists and is newer than *file2*.
file1 **–ot** *file2*	True, if *file1* exists and is older than *file2*.

file1 **-ef** *file2*	True, if *file1* and *file2* exist and refer to the same file.
string = *pattern*	True, if *string* matches *pattern*.
string != *pattern*	True, if *string* does not match *pattern*.
string1 < *string2*	True, if *string1* comes before *string2* based on ASCII value of their characters.
string1 > *string2*	True, if *string1* comes after *string2* based on ASCII value of their characters.
exp1 **-eq** *exp2*	True, if *exp1* is equal to *exp2*.
exp1 **-ne** *exp2*	True, if *exp1* is not equal to *exp2*.
exp1 **-lt** *exp2*	True, if *exp1* is less than *exp2*.
exp1 **-gt** *exp2*	True, if *exp1* is greater than *exp2*.
exp1 **-le** *exp2*	True, if *exp1* is less than or equal to *exp2*.
exp1 **-ge** *exp2*	True, if *exp1* is greater than or equal to *exp2*.

In each of the above expressions, if *file* is of the form **/dev/fd/***n*, where *n* is an integer, then the test applied to the open file whose descriptor number is *n*.

A compound expression can be constructed from these primitives by using any of the following, listed in decreasing order of precedence.

(*expression*) True, if *expression* is true. Used to group expressions.

! *expression* True if *expression* is false.

expression1 && *expression2*
True, if *expression1* and *expression2* are both true.

expression1 || *expression2*
True, if either *expression1* or *expression2* is true.

Input/Output

Before a command is executed, its 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 to the invoked command. Command and parameter substitution occurs before *word* or *digit* is used except as noted below. File name generation occurs only if the pattern matches a single file and blank interpretation is not performed.

<*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 then it is created. If the file exists, and the **noclobber** option is on, this causes an error; otherwise, it is truncated to zero length.
>|*word*	Sames as >, except that it overrides the **noclobber** option.
>>*word*	Use file *word* as standard output. If the file exists then output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.
<>*word*	Open file *word* for reading and writing as standard input.
<<[-]*word*	The shell input is read up to a line that is the same as *word*, or to an end-of-file. No parameter substitution, command substitution or file name generation is performed on *word*. The resulting document, called a *here-document*, becomes the standard input. If any character of *word* is quoted, then no

interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, **\new-line** is ignored, and \ must be used to quote the characters \, $, `, and the first character of *word*. If – is appended to <<, then all leading tabs are stripped from *word* and from the document.

<&*digit* The standard input is duplicated from file descriptor *digit* (see **dup**(2)). Similarly for the standard output using **>&** *digit*.

<&– The standard input is closed. Similarly for the standard output using **>&–**.

<&p The input from the co-process is moved to standard input.

>&p The output to the co-process is moved to standard output.

If one of the above is preceded by a digit, then the file descriptor number referred to is that specified by the digit (instead of the default 0 or 1). For example:

... 2>&1

means file descriptor 2 is to be opened for writing as a duplicate of file descriptor 1.

The order in which redirections are specified is significant. The shell evaluates each redirection in terms of the (*file descriptor*, *file*) association at the time of evaluation. For example:

... 1>*fname* 2>&1

first associates file descriptor 1 with file *fname*. It then associates file descriptor 2 with the file associated with file descriptor 1 (i.e. *fname*). If the order of redirections were reversed, file descriptor 2 would be associated with the terminal (assuming file descriptor 1 had been) and then file descriptor 1 would be associated with file *fname*.

If a command is followed by & and job control is not active, then 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.

Environment

The *environment* (see **environ**(5)) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The names must be *identifiers* and the values are character strings. 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 and marking it *export*. Executed commands inherit the environment. If the user modifies the values of these parameters or creates new ones, using the **export** or **typeset –x** commands they become part of the environment. The environment seen by any executed command is thus composed of any name-value pairs originally inherited by the shell, whose values may be modified by the current shell, plus any additions which must be noted in **export** or **typeset –x** commands.

The environment for any *simple-command* or function may be augmented by prefixing it with one or more parameter assignments. A parameter

assignment argument is a word of the form *identifier=value*. Thus:

```
TERM=450 cmd args                    and
(export TERM; TERM=450; cmd args)
```

are equivalent (as far as the above execution of *cmd* is concerned).

If the **-k** flag is set, *all* parameter assignment arguments are placed in the environment, even if they occur after the command name. The following first prints **a=b c** and then **c:**

```
echo a=b c
set -k
echo a=b c
```

This feature is intended for use with scripts written for early versions of the shell and its use in new scripts is strongly discouraged. It is likely to disappear someday.

Functions

The **function** reserved word, described in the *Commands* section above, is used to define shell functions. Shell functions are read in and stored internally. Alias names are resolved when the function is read. Functions are executed like commands with the arguments passed as positional parameters. (See **Execution** below).

Functions execute in the same process as the caller and share all files and present working directory with the caller. Traps caught by the caller are reset to their default action inside the function. A trap condition that is not caught or ignored by the function causes the function to terminate and the condition to be passed on to the caller. A trap on **EXIT** set inside a function is executed after the function completes in the environment of the caller. Ordinarily, variables are shared between the calling program and the function. However, the **typeset** special command used within a function defines local variables whose scope includes the current function and all functions it calls.

The special command **return** is used to return from function calls. Errors within functions return control to the caller.

Function identifiers can be listed with the **-f** or **+f** option of the **typeset** special command. The text of functions will also be listed with **-f**. Function can be undefined with the **-f** option of the **unset** special command.

Ordinarily, functions are unset when the shell executes a shell script. The **-xf** option of the **typeset** command allows a function to be exported to scripts that are executed without a separate invocation of the shell. Functions that need to be defined across separate invocations of the shell should be specified in the **ENV** file with the **-xf** option of **typeset**

Jobs

If the **monitor** option of the **set** command is turned on, an interactive 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 (Ctrl-Z) which sends a STOP signal to the current job. The shell then normally indicates 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.

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. A job can be referred to by the process ID of any process of the job or by one of the following:

%number	The job with the given number.
%string	Any job whose command line begins with *string*.
%?string	Any job whose command line contains *string*.
%%	Current job.
%+	Equivalent to *%%*.
%–	Previous job.

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.

When the monitor mode is on, each background job that completes triggers any trap set for **CHLD**.

When you try to leave the shell while jobs are running or stopped, you will be warned that 'You have stopped (running) 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 stopped jobs will be terminated.

Signals

The INT and QUIT signals for an invoked command are ignored if the command is followed by & and job **monitor** option is not active. Otherwise, signals have the values inherited by the shell from its parent (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 within the current shell process. Next, the command name is checked to see if it matches one of the user defined functions. If it does, the positional parameters are saved and then reset to the arguments of the *function* call. When the *function* completes or issues a **return**, the positional parameter list is restored and any trap set on **EXIT** within the function is executed. The value of a *function* is the value of the last command executed. A

function is also executed in the current shell process. If a command name is not a *special command* or a user defined *function*, a 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 **/bin**, **/usr/bin**, and the current directory in that order). The current directory can be specified by two or more adjacent colons, or by a colon at the beginning or end of the path list. If the command name contains a / then the search path is not used. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not a directory or an **a.out** file, it is assumed to be a file containing shell commands. A sub-shell is spawned to read it. All non-exported aliases, functions, and named parameters are removed in this case. If the shell command file doesn't have read permission, or if the *setuid* and/or *setgid* bits are set on the file, then the shell executes an agent whose job it is to set up the permissions and execute the shell with the shell command file passed down as an open file. A parenthesized command is executed in a sub-shell without removing non-exported quantities.

Command Re-entry

The text of the last **HISTSIZE** (default 128) commands entered from a terminal device is saved in a *history* file. The file *$HOME/.sh_history* is used if the **HISTFILE** variable is not set or is not writable. A shell can access the commands of all *interactive* shells which use the same named **HISTFILE**. The special command **fc** is used to list or edit a portion of this file. The portion of the file to be edited or listed can be selected by number or by giving the first character or characters of the command. A single command or range of commands can be specified. If you do not specify an editor program as an argument to **fc** then the value of the parameter **FCEDIT** is used. If **FCEDIT** is not defined then **/bin/ed** is used. The edited command(s) is printed and re-executed upon leaving the editor. The editor name – is used to skip the editing phase and to re-execute the command. In this case a substitution parameter of the form *old=new* can be used to modify the command before execution. For example, if **r** is aliased to **'fc –e –'** then typing **'r bad=good c'** will re-execute the most recent command which starts with the letter **c**, replacing the first occurrence of the string **bad** with the string **good**.

In-line Editing Options

Normally, each command line entered from a terminal device is simply typed followed by a new-line ('RETURN' or 'LINE FEED'). If either the **emacs**, **gmacs**, or **vi** option is active, the user can edit the command line. To be in either of these edit modes **set** the corresponding option. An editing option is automatically selected each time the **VISUAL** or **EDITOR** variable is assigned a value ending in either of these option names.

The editing features require that the user's terminal accept 'RETURN' as carriage return without line feed and that a space (' ') must overwrite the current character on the screen. ADM terminal users should set the "space - advance" switch to 'space'. Hewlett-Packard series 2621 terminal users should set the straps to 'bcGHxZ etX'.

The editing modes implement a concept where the user is looking through a window at the current line. The window width is the value of **COLUMNS** if it

is defined, otherwise 80. If the line is longer than the window width minus two, a mark is displayed at the end of the window to notify the user. As the cursor moves and reaches the window boundaries the window will be centered about the cursor. The mark is a > (<, *) if the line extends on the right (left, both) side(s) of the window.

The search commands in each edit mode provide access to the history file. Only strings are matched, not patterns, although a leading ^ in the string restricts the match to begin at the first character in the line.

Emacs Editing Mode

This mode is entered by enabling either the *emacs* or *gmacs* option. The only difference between these two modes is the way they handle ^T. To edit, the user moves the cursor to the point needing correction and then inserts or deletes characters or words as needed. All the editing commands are control characters or escape sequences. The notation for control characters is caret (^) followed by the character. For example, ^F is the notation for control F. This is entered by depressing 'f' while holding down the 'CTRL' (control) key. The 'SHIFT' key is *not* depressed. (The notation ^? indicates the DEL (delete) key.)

The notation for escape sequences is M- followed by a character. For example, M-f (pronounced Meta f) is entered by depressing ESC (ascii 033) followed by 'f'. (M-F would be the notation for ESC followed by 'SHIFT' (capital) 'F'.)

All edit commands operate from any place on the line (not just at the beginning). Neither the "RETURN" nor the "LINE FEED" key is entered after edit commands except when noted.

^F	Move cursor forward (right) one character.
M-f	Move cursor forward one word. (The emacs editor's idea of a word is a string of characters consisting of only letters, digits and underscores.)
^B	Move cursor backward (left) one character.
M-b	Move cursor backward one word.
^A	Move cursor to start of line.
^E	Move cursor to end of line.
^]*char*	Move cursor forward to character *char* on current line.
M-^]*char*	Move cursor back to character *char* on current line.
^X^X	Interchange the cursor and mark.
erase	(User defined erase character as defined by the **stty**(1) command, usually ^H or #.) Delete previous character.
^D	Delete current character.
M-d	Delete current word.
M-^H	(Meta-backspace) Delete previous word.
M-h	Delete previous word.
M-^?	(Meta-DEL) Delete previous word (if your interrupt character is ^? (DEL, the default) then this command will not work).
^T	Transpose current character with next character in *emacs* mode. Transpose two previous characters in *gmacs* mode.
^C	Capitalize current character.
M-c	Capitalize current word.
M-l	Change the current word to lower case.
^K	Delete from the cursor to the end of the line. If preceded by a numerical parameter whose value is less than the current cursor

position, then delete from given position up to the cursor. If preceded by a numerical parameter whose value is greater than the current cursor position, then delete from cursor up to given cursor position.

^W Kill from the cursor to the mark.

M-p Push the region from the cursor to the mark on the stack.

kill (User defined kill character as defined by the stty command, usually **^G** or **@**.) Kill the entire current line. If two *kill* characters are entered in succession, all kill characters from then on cause a line feed (useful when using paper terminals).

^Y Restore last item removed from line. (Yank item back to the line.)

^L Line feed and print current line.

^@ (Null character) Set mark.

M-*space* (Meta space) Set mark.

^J (New line) Execute the current line.

^M (Return) Execute the current line.

eof End-of-file character, normally **^D**, is processed as an End-of-file only if the current line is null.

^P Fetch previous command. Each time **^P** is entered the previous command back in time is accessed. Moves back one line when not on the first line of a multi-line command.

M-< Fetch the least recent (oldest) history line.

M-> Fetch the most recent (youngest) history line.

^N Fetch next command line. Each time **^N** is entered the next command line forward in time is accessed.

^R*string* Reverse search history for a previous command line containing *string*. If a parameter of zero is given, the search is forward. *String* is terminated by a "RETURN" or "NEW LINE". If string is preceded by a **^**, the matched line must begin with *string*. If *string* is omitted, then the next command line containing the most recent *string* is accessed. In this case a parameter of zero reverses the direction of the search.

^O Operate – Execute the current line and fetch the next line relative to current line from the history file.

M-*digits* (Escape) Define numeric parameter, the digits are taken as a parameter to the next command. The commands that accept a parameter are **^F**, **^B**, *erase*, **^C**, **^D**, **^K**, **^R**, **^P**, **^N**, **^]**, **M-.**, **M-^]**, **M-_**, **M-b**, **M-c**, **M-d**, **M-f**, **M-h** **M-l** and **M-^H**.

M-*letter* Soft-key – Your alias list is searched for an alias by the name *_letter* and if an alias of this name is defined, its value will be inserted on the input queue. The *letter* must not be one of the above meta-functions. **M-]***letter* Soft-key – Your alias list is searched for an alias by the name *__letter* and if an alias of this name is defined, its value will be inserted on the input queue. The can be used to program functions keys on many terminals.

M-. The last word of the previous command is inserted on the line. If preceded by a numeric parameter, the value of this parameter determines which word to insert rather than the last word.

M-_ Same as **M-.**.

M-* Attempt file name generation on the current word. An asterisk is appended if the word doesn't match any file or contain any special

pattern characters.

M-ESC File name completion. Replaces the current word with the longest common prefix of all filenames matching the current word with an asterisk appended. If the match is unique, a / is appended if the file is a directory and a space is appended if the file is not a directory.

M-= List files matching current word pattern if an asterisk were appended.

^U Multiply parameter of next command by 4.

**** Escape next character. Editing characters, the user's erase, kill and interrupt (normally **^?**) characters may be entered in a command line or in a search string if preceded by a ****. The **** removes the next character's editing features (if any).

^V Display version of the shell.

M-# Insert a # at the beginning of the line and execute it. This causes a comment to be inserted in the history file.

Vi Editing Mode

There are two typing modes. Initially, when you enter a command you are in the *input* mode. To edit, the user enters *control* mode by typing ESC (**033**) and moves the cursor to the point needing correction and then inserts or deletes characters or words as needed. Most control commands accept an optional repeat *count* prior to the command.

When in **vi** mode on most systems, canonical processing is initially enabled and the command will be echoed again if the speed is 1200 baud or greater and it contains any control characters or less than one second has elapsed since the prompt was printed. The ESC character terminates canonical processing for the remainder of the command and the user can then modify the command line. This scheme has the advantages of canonical processing with the type-ahead echoing of raw mode.

If the option **viraw** is also set, the terminal will always have canonical processing disabled. This mode is implicit for systems that do not support two alternate end of line delimiters, and may be helpful for certain terminals.

Input Edit Commands

By default the editor is in input mode.

erase (User defined erase character as defined by the stty command, usually **^H** or **#**.) Delete previous character.

^W Delete the previous blank separated word.

^D Terminate the shell.

^V Escape next character. Editing characters, the user's erase or kill characters may be entered in a command line or in a search string if preceded by a **^V**. The **^V** removes the next character's editing features (if any).

**** Escape the next *erase* or *kill* character.

Motion Edit Commands

These commands will move the cursor.

[*count*]**l** Cursor forward (right) one character.

[*count*]**w** Cursor forward one alpha-numeric word.

[*count*]**W** Cursor to the beginning of the next word that follows a blank.

[*count*]**e** Cursor to end of word.
[*count*]**E** Cursor to end of the current blank delimited word.
[*count*]**h** Cursor backward (left) one character.
[*count*]**b** Cursor backward one word.
[*count*]**B** Cursor to preceding blank separated word.
[*count*]**|** Cursor to column *count*.
[*count*]**f***c* Find the next character *c* in the current line.
[*count*]**F***c* Find the previous character *c* in the current line.
[*count*]**t***c* Equivalent to **f** followed by **h**.
[*count*]**T***c* Equivalent to **F** followed by **l**.
[*count*]**;** Repeats *count* times, the last single character find command, **f**, **F**, **t**, or **T**.
[*count*]**,** Reverses the last single character find command *count* times.
0 Cursor to start of line.
^ Cursor to first non-blank character in line.
$ Cursor to end of line.

Search Edit Commands

These commands access your command history.

[*count*]**k** Fetch previous command. Each time **k** is entered the previous command back in time is accessed.
[*count*]**–** Equivalent to **k**.
[*count*]**j** Fetch next command. Each time **j** is entered the next command forward in time is accessed.
[*count*]**+** Equivalent to **j**.
[*count*]**G** The command number *count* is fetched. The default is the least recent history command.
/string Search backward through history for a previous command containing *string*. *String* is terminated by a "RETURN" or "NEW LINE". If string is preceded by a ^, the matched line must begin with *string*. If *string* is null the previous string will be used.
?string Same as / except that search will be in the forward direction.
n Search for next match of the last pattern to / or ? commands.
N Search for next match of the last pattern to / or ?, but in reverse direction. Search history for the *string* entered by the previous / command.

Text Modification Edit Commands

These commands will modify the line.

a Enter input mode and enter text after the current character.
A Append text to the end of the line. Equivalent to **$a**.
[*count*]**c***motion*
c[*count*]*motion*
Delete current character through the character that *motion* would move the cursor to and enter input mode. If *motion* is **c**, the entire line will be deleted and input mode entered.
C Delete the current character through the end of line and enter input mode. Equivalent to **c$**.

S Equivalent to **cc**.

D Delete the current character through the end of line. Equivalent to **d$**.

[*count*]**d***motion*

d[*count*]*motion*

Delete current character through the character that *motion* would move to. If *motion* is **d** , the entire line will be deleted.

i Enter input mode and insert text before the current character.

I Insert text before the beginning of the line. Equivalent to **0i**.

[*count*]**P** Place the previous text modification before the cursor.

[*count*]**p** Place the previous text modification after the cursor.

R Enter input mode and replace characters on the screen with characters you type overlay fashion.

[*count*]**r***c* Replace the *count* character(s) starting at the current cursor position with *c*, and advance the cursor.

[*count*]**x** Delete current character.

[*count*]**X** Delete preceding character.

[*count*]**.** Repeat the previous text modification command.

[*count*]**~** Invert the case of the *count* character(s) starting at the current cursor position and advance the cursor.

[*count*]**_** Causes the *count* word of the previous command to be appended and input mode entered. The last word is used if *count* is omitted.

***** Causes an * to be appended to the current word and file name generation attempted. If no match is found, it rings the bell. Otherwise, the word is replaced by the matching pattern and input mode is entered.

**** Filename completion. Replaces the current word with the longest common prefix of all filenames matching the current word with an asterisk appended. If the match is unique, a / is appended if the file is a directory and a space is appended if the file is not a directory.

Other Edit Commands

Miscellaneous commands.

[*count*]**y***motion*

y[*count*]*motion*

Yank current character through character that *motion* would move the cursor to and puts them into the delete buffer. The text and cursor are unchanged.

Y Yanks from current position to end of line. Equivalent to **y$**.

u Undo the last text modifying command.

U Undo all the text modifying commands performed on the line.

[*count*]**v** Returns the command **fc –e ${VISUAL:–${EDITOR:–vi}}** *count* in the input buffer. If *count* is omitted, then the current line is used.

^L	Line feed and print current line. Has effect only in control mode.
^J	(New line) Execute the current line, regardless of mode.
^M	(Return) Execute the current line, regardless of mode.
#	Sends the line after inserting a # in front of the line. Useful for causing the current line to be inserted in the history without being executed.
=	List the file names that match the current word if an asterisk were appended it.
@*letter*	Your alias list is searched for an alias by the name *_letter* and if an alias of this name is defined, its value will be inserted on the input queue for processing.

Special Commands

The following simple commands are executed in the shell process. Input/Output redirection is permitted. Unless otherwise indicated, the output is written on file descriptor 1 and the exit status, when there is no syntax error, is zero. Commands that are preceded by one or two † are treated specially in the following ways:

1. Parameter assignment lists preceding the command remain in effect when the command completes.
2. I/O redirections are processed after parameter assignments.
3. Errors cause a script that contains them to abort.
4. Words, following a command preceded by †† that are in the format of a parameter assignment, are expanded with the same rules as a parameter assignment. This means that tilde substitution is performed after the = sign and word splitting and file name generation are not performed.

† **:** [*arg* ...]
The command only expands parameters.

† **.** *file* [*arg* ...]
Read the complete *file* then execute the commands. The commands are executed in the current shell environment. The search path specified by **PATH** is used to find the directory containing *file*. If any arguments *arg* are given, they become the positional parameters. Otherwise the positional parameters are unchanged. The exit status is the exit status of the last command executed.

†† **alias** [**-tx**] [*name*[=*value*]]...
Alias with no arguments prints the list of aliases in the form *name=value* on standard output. An *alias* is defined for each name whose *value* is given. A trailing space in *value* causes the next word to be checked for alias substitution. The **-t** flag is used to set and list tracked aliases. The value of a tracked alias is the full pathname corresponding to the given *name*. The value becomes undefined when the value of **PATH** is reset but the aliases remained tracked. Without the **-t** flag, for each *name* in the argument list for which no *value* is given, the name and value of the alias is printed. The **-x** flag is used to set or print exported aliases. An exported alias is defined for scripts invoked by name. The exit status is non-zero if a *name* is given, but no value, for which no alias has been defined.

bg [*job*...]
This command is only on systems that support job control. Puts each specified *job* into the background. The current job is put in the background if *job* is not specified. See *Jobs* for a description of the format of *job*.

† **break** [*n*]
Exit from the enclosing **for while until** or **select** loop, if any. If *n* is specified then break *n* levels.

† **continue** [*n*]
Resume the next iteration of the enclosing **for while until** or **select** loop. If *n* is specified then resume at the *n*-th enclosing loop.

cd [*arg*]
cd *old new*
This command can be in either of two forms. In the first form it changes the current directory to *arg*. If *arg* is – the directory is changed to the previous directory. The shell parameter **HOME** is the default *arg*. The parameter **PWD** is set to the current directory. 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 / then the search path is not used. Otherwise, each directory in the path is searched for *arg*.

The second form of **cd** substitutes the string *new* for the string *old* in the current directory name, **PWD** and tries to change to this new directory.

The **cd** command may not be executed by **rksh**.

echo [*arg* ...]
See **echo**(1) for usage and description.

† **eval** [*arg* ...]
The arguments are read as input to the shell and the resulting command(s) executed.

† **exec** [*arg* ...]
If *arg* is given, the command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and affect the current process. If no arguments are given the effect of this command is to modify file descriptors as prescribed by the input/output redirection list. In this case, any file descriptor numbers greater than 2 that are opened with this mechanism are closed when invoking another program.

† **exit** [*n*]
Causes the shell to exit with the exit status specified by *n*. If *n* is omitted then the exit status is that of the last command executed. An end-of-file will also cause the shell to exit except for a shell which has the *ignoreeof* option (See **set** below) turned on.

†† **export** [*name*[=*value*]] ...

The given *name*s are marked for automatic export to the *environment* of subsequently-executed commands.

fc [**–e** *ename*] [**–nlr**] [*first* [*last*]]
fc –e – [*old=new*] [*command*]

In the first form, a range of commands from *first* to *last* is selected from the last **HISTSIZE** commands that were typed at the terminal. The arguments *first* and *last* may be specified as a number or as a string. A string is used to locate the most recent command starting with the given string. A negative number is used as an offset to the current command number. If the flag **–l**, is selected, the commands are listed on standard output. Otherwise, the editor program *ename* is invoked on a file containing these keyboard commands. If *ename* is not supplied, then the value of the parameter **FCEDIT** (default /bin/ed) is used as the editor. When editing is complete, the edited command(s) is executed. If *last* is not specified then it will be set to *first*. If *first* is not specified the default is the previous command for editing and –16 for listing. The flag **–r** reverses the order of the commands and the flag **–n** suppresses command numbers when listing. In the second form the *command* is re-executed after the substitution *old=new* is performed.

fg [*job*...]

This command is only on systems that support job control. Each *job* specified is brought to the foreground. Otherwise, the current job is brought into the foreground. See *Jobs* for a description of the format of *job*.

getopts *optstring name* [*arg* ...]

Checks *arg* for legal options. If *arg* is omitted, the positional parameters are used. An option argument begins with a + or a –. An option not beginning with + or – or the argument –– ends the options. *optstring* contains the letters that *getopts* recognizes. If a letter is followed by a :, that option is expected to have an argument. The options can be separated from the argument by blanks.

getopts places the next option letter it finds inside variable *name* each time it is invoked with a + prepended when *arg* begins with a +. The index of the next *arg* is stored in **OPTIND**. The option argument, if any, gets stored in **OPTARG**.

A leading **:** in *optstring* causes **getopts** to store the letter of an invalid option in **OPTARG**, and to set *name* to **?** for an unknown option and to **:** when a required option is missing. Otherwise, **getopts** prints an error message. The exit status is non-zero when there are no more options.

jobs [**–lnp**] [*job* ...]

Lists information about each given job; or all active jobs if *job* is omitted. The **–l** flag lists process ids in addition to the normal information. The **–n** flag only displays jobs that have stopped or exited since last notified. The **–p** flag causes only the process group to be listed. See *Jobs* for a description of the format of *job*.

kill [*–sig*] *job* ...

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"). If the signal being sent is TERM (terminate) or HUP (hangup), then the job or process will be sent a CONT (continue) signal if it is stopped. The argument *job* can the process ID of a process that is not a member of one of the active jobs. See *Jobs* for a description of the format of *job*. In the second form, **kill –l**, the signal numbers and names are listed.

let *arg* ...

Each *arg* is a separate *arithmetic expression* to be evaluated. See *Arithmetic Evaluation* above, for a description of arithmetic expression evaluation.

The exit status is 0 if the value of the last expression is non-zero, and 1 otherwise.

† **newgrp** [*arg* ...]

Equivalent to **exec /bin/newgrp** *arg* ....

print [**–Rnprsu**[*n*]][*arg* ...]

The shell output mechanism. With no flags or with flag – or –– the arguments are printed on standard output as described by **echo**(1). In raw mode, **–R** or **–r**, the escape conventions of *echo* are ignored. The **–R** option will print all subsequent arguments and options other than **–n**. The **–p** option causes the arguments to be written onto the pipe of the process spawned with **|&** instead of standard output. The **–s** option causes the arguments to be written onto the history file instead of standard output. The **–u** flag can be used to specify a one digit file descriptor unit number **n** on which the output will be placed. The default is 1. If the flag **–n** is used, no **new-line** is added to the output.

pwd Equivalent to **print –r – $PWD**

read [**–prsu**[*n*]][*name*?*prompt*][*name* ...]

The shell input mechanism. One line is read and is broken up into fields using the characters in **IFS** as separators. In raw mode, **–r**, a \ at the end of a line does not signify line continuation. The first field is assigned to the first *name*, the second field to the second *name*, etc., with leftover fields assigned to the last *name*. The **–p** option causes the input line to be taken from the input pipe of a process spawned by the shell using **|&**. If the **–s** flag is present, the input will be saved as a command in the history file. The flag **–u** can be used to specify a one digit file descriptor unit to read from. The file descriptor can be opened with the **exec** special command. The default value of *n* is 0. If *name* is omitted then **REPLY** is used as the default *name*. The exit status is 0 unless an end-of-file is encountered. An end-of-file with the **–p** option causes cleanup for this process so that another can be spawned. If the first argument contains a **?**, the remainder of this word is used as a *prompt* on standard error when the shell is interactive. The exit status is 0 unless an end-of-file is encountered.

†† **readonly** [*name*[=*value*]]...
The given *names* are marked readonly and these names cannot be changed by subsequent assignment.

† **return** [*n*]
Causes a shell *function* to return to the invoking script with the return status specified by *n*. If *n* is omitted then the return status is that of the last command executed. If **return** is invoked while not in a *function* or a . script, then it is the same as an **exit**.

set [±**aefhkmnopstuvx**] [±**o** *option*]... [±**A** *name*] [*arg* ...]
The flags for this command have meaning as follows:

-A Array assignment. Unset the variable *name* and assign values sequentially from the list *arg*. If **+A** is used, the variable *name* is not unset first.

-a All subsequent parameters that are defined are automatically exported.

-e If a command has a non-zero exit status, execute the **ERR** trap, if set, and exit. This mode is disabled while reading profiles.

-f Disables file name generation.

-h Each command becomes a tracked alias when first encountered.

-k All parameter assignment arguments are placed in the environment for a command, not just those that precede the command name.

-m Background jobs will run in a separate process group and a line will print upon completion. The exit status of background jobs is reported in a completion message. On systems with job control, this flag is turned on automatically for interactive shells.

-n Read commands and check them for syntax errors, but do not execute them. Ignored for interactive shells.

-o The following arguments can be option names. If no option name is supplied, the current option settings are printed.

allexport Same as **-a**.

errexit Same as **-e**.

bgnice All background jobs are run at a lower priority. This is the default mode.

emacs Puts you in an *emacs* style in-line editor for command entry.

gmacs Puts you in a *gmacs* style in-line editor for command entry.

ignoreeof
The shell will not exit on end-of-file. The command **exit** must be used.

keyword Same a **-k**.

markdirs
All directory names resulting from file name generation have a trailing / appended.

monitor Same as **-m**.

noclobber
Prevents redirection > from truncating existing files. Require >| to truncate a file when turned on. .

noexec Same as **-n**.

noglob Same as **-f**.

nolog Do not save function definitions in history file.

nounset Same as **-u**.

privileged
Same as **-p**.

verbose Same as **-v**.

trackall Same as **-h**.

vi Puts you in insert mode of a *vi* style in-line editor until you hit escape character **033**. This puts you in move mode. A return sends the line.

viraw Each character is processed as it is typed in *vi* mode.

xtrace Same as **-x**.

-s Sort the positional parameters lexicographically.

-t Exit after reading and executing one command.

-u Treat unset parameters as an error when substituting.

-v Print shell input lines as they are read.

-x Print commands and their arguments as they are executed.

- Turns off **-x** and **-v** flags and stops examining arguments for flags.

-- Do not change any of the flags; useful in setting **$1** to a value beginning with -. If no arguments follow this flag then the positional parameters are unset.

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 **$-**. Unless **-A** is specified, the remaining arguments are positional parameters and are assigned, in order, to **$1** **$2** .... If no arguments are given then the names and values of all named parameters are printed on the standard output. If the only argument is +, the names of all named parameters are printed.

† **shift** [*n*]
The positional parameters from **$***n*+**1**... are renamed **$1**... , default *n* is 1. The parameter *n* can be any arithmetic expression that evaluates to a non-negative number less than or equal to **$#**.

† **times**
Print the accumulated user and system times for the shell and for processes run from the shell.

† **trap** [*arg*][*sig*]...
arg is a command to be read and executed when the shell receives signal(s) *sig*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Each *sig* can be given as a number or as the name of the signal. **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. If *arg* is

omitted or is –, then all trap(s) *sig* 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 *sig* is **ERR**, then *arg* is executed whenever a command has a non-zero exit status. If *sig* is **DEBUG**, then *arg* is executed after each command. If *sig* is **0** or **EXIT** and the **trap** statement is executed inside the body of a function, then the command *arg* is executed after the function completes. If *sig* is **0** or **EXIT** for a **trap** set outside any function, then 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.

†† **typeset** [±**HLRZfilrtux**[*n*]] [*name*[=*value*]]...

Sets attributes and values for shell parameters. When invoked inside a function, a new instance of the parameter *name* is created. The parameter value and type are restored when the function completes. The following list of attributes may be specified:

–H This flag provides system-to-host-name file mapping on other operating systems.

–L Left justify and remove leading blanks from *value*. If *n* is non-zero, it defines the width of the field; otherwise, it is determined by the width of the value of first assignment. When the parameter is assigned, it is filled on the right with blanks or truncated, if necessary, to fit into the field. Leading zeros are removed if the **–Z** flag is also set. The **–R** flag is turned off.

–R Right justify and fill with leading blanks. If *n* is non-zero, it defines the width of the field; otherwise, it is determined by the width of the value of first assignment. The field is left-filled with blanks or truncated from the end if the parameter is reassigned. The **L** flag is turned off.

–Z Right justify and fill with leading zeros if the first non-blank character is a digit and the **–L** flag has not been set. If *n* is non-zero, it defines the width of the field; otherwise, it is determined by the width of the value of first assignment.

–f The names refer to function names rather than parameter names. No assignments can be made, and the only other valid flags are **–t**, **–u** and **–x**. The flag **–t** turns on execution tracing for this function. The flag **–u** causes this function to be marked undefined. The **FPATH** variable is checked to find the function definition when the function is referenced. The flag **–x** allows the function definition to remain in effect across shell procedures invoked by name.

–i Parameter is an integer. This makes arithmetic faster. If *n* is non-zero, it defines the output arithmetic base; otherwise, the first assignment determines the output base.

–l All uppercase characters are converted to lowercase. The uppercase flag, **–u** is turned off.

–r The given *names* are marked read-only, and these names cannot be changed by subsequent assignment.

-t Tags the named parameters. Tags are user-definable and have no special meaning to the shell.

-u All lowercase characters are converted to uppercase characters. The lowercase flag, **-l**, is turned off.

-x The given *name*s are marked for automatic export to the *environment* of subsequently executed commands.

Using + rather than - causes these flags to be turned off. If no *name* arguments are given but flags are specified, a list of *names* (and optionally the *values*) of the *parameters* which have these flags set is printed. (Using + rather than - keeps the values from being printed.) If no *name*s and flags are given, the *names* and *attributes* of all *parameters* are printed.

ulimit [**-HSacdfmnpstvw**] [*limit*]

Set or display a resource limit. The available resource limits are listed below. Many systems do not contain one or more of these limits. The limit for a specified resource is set when *limit* is specifed. The value of *limit* can be a number in the unit specifed below with each resource or the value **unlimited**. The **H** and **S** flags specify whether the hard limit or the soft limit for the given resource is set. A hard limit cannot be increased once it is set. A soft limit can be increased up to the value of the hard limit. If neither the **H** nor **S** options is specified, the limit applies to both. The current resource limit is printed when *limit* is omitted. In this case, the soft limit is printed unless **H** is specified. When more that one resource is specifed, the limit name and unit is printed before the value.

-a Lists all of the current resource limits.

-c The number of 512-byte blocks on the size of core dumps.

-d The number of K-bytes on the size of the data area.

-f The number of 512-byte blocks on files written by child processes (files of any size may be read).

-m The number of K-bytes on the size of physical memory.

-n The number of file descriptors.

-p The number of 512-byte blocks for pipe buffering.

-s The number of K-bytes on the size of the stack area.

-t The number of seconds to be used by each process.

-v The number of K-bytes for virtual memory.

-w The number of K-bytes for the swap area.

If no option is given, **-f** is assumed.

umask [*mask*]

The user file-creation mask is set to *mask* (see **umask**(2)). *mask* can either be an octal number or a symbolic value as described in **chmod**(1). If a symbolic value is given, the new **umask** value is the complement of the result of applying *mask* to the complement of the previous **umask** value. If *mask* is omitted, the current value of the mask is printed.

unalias *name* ...

The parameters given by the list of *name*s are removed from the *alias* list.

unset [**–f**] *name* ...
The parameters given by the list of *name*s are unassigned, i. e., their values and attributes are erased. Read-only variables cannot be unset. If flag **–f** is set, the names refer to *function* names. Unsetting **ERRNO, LINENO, MAILCHECK, OPTARG, OPTIND, RANDOM, SECONDS, TMOUT,** and _ removes their special meaning even if they are subsequently assigned to.

† **wait** [*job*]
Wait for the specified *job,* and report its termination status. If *job* is not given, all currently active child processes are waited for. The exit status from this command is that of the process waited for. See *Jobs* for a description of the format of *job*.

whence [**–pv**] *name* ...
For each *name*, indicate how it would be interpreted if used as a command name. Flag **–v** produces a more verbose report. Flag **–p** does a path search for *name* even if name is an alias, a function, or a reserved word.

Invocation

If the shell is invoked by **exec (2)** and the first character of argument zero (**$0**) is –, the shell is assumed to be a *login* shell, and commands are read from */etc/profile* and then from either *.profile* in the current directory or *$HOME/.profile* if either file exists. Next, commands are read from the file named by performing parameter substitution on the value of the environment parameter **ENV** if the file exists. If the **–s** flag is not present and *arg* is, a path search is performed on the first *arg* to determine the name of the script to execute. The script *arg* must have read permission, and any *setuid* and *getgid* settings are ignored. Commands are then read as described below. The following flags are interpreted by the shell when it is invoked:

–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. **shell** output, except for the output of the *Special commands* listed above, 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 (as told by **ioctl**(2)), this shell is *interactive*. In this case, TERM is ignored (so that **kill 0** does not kill an interactive shell), and INTR 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.

rksh Only

rksh is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of **rksh** are identical to those of **sh**, except that the following are disallowed:

changing directory (see **cd**(1)),
setting the value of **SHELL, ENV,** or **PATH**,
specifying path or command names containing /,
redirecting output (**>**, **>|** , **<>** , and **>>**).

The restrictions above are enforced after *.profile* and the **ENV** files are interpreted.

When a command to be executed is found to be a shell procedure, **rksh** invokes **ksh** 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* 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/rbin*) that can be safely invoked by **rksh**. Some systems also provide a restricted editor **red**.

FILES

/etc/passwd
/etc/profile
$HOME/.profile
/tmp/sh*
/dev/null

SEE ALSO

cat(1), cd(1), chmod(1), cut(1), echo(1), emacs(1), env(1), gmacs(1), newgrp(1), stty(1), test(1), umask(1), vi(1), dup(2), exec(2), fork(2), ioctl(2), lseek(2), paste(1), pipe(2), signal(2), umask(2), ulimit(2), wait(2), rand(3), a.out(5), profile(5), environ(5).

Morris I. Bolsky and David G. Korn, *The KornShell Command and Programming Language*, Prentice Hall, 1989.

DIAGNOSTICS

Errors detected by the shell, such as syntax errors, cause the shell to return a non-zero exit status. Otherwise, the shell returns the exit status of the last command executed (see also the **exit** command above). If the shell is being used noninteractively, execution of the shell file is abandoned. Runtime errors detected by the shell are reported by printing the command or function name and the error condition. If the line number that the error occurred on is greater than one, the line number is also printed in square brackets ([]) after the command or function name.

WARNINGS

If a command which is a *tracked alias* is executed, and then 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 **–t** option of the **alias** command to correct this situation.

Some very old shell scripts contain a ^ as a synonym for the pipe character |.

Using the **fc** built-in command within a compound command causes the whole command to disappear from the history file.

The built-in command **.** *file* reads the whole file before any commands are executed. Therefore, **alias** and **unalias** commands in the file will not apply to

any functions defined in the file.

Traps are not processed while a job is waiting for a foreground process. Thus, a trap on **CHLD** won't be executed until the forground job terminates.

The current implemetation of **ksh** exectutes each element of a pipeline, except the last, in a subshell environment. Therefore, any side effects of built-ins, except in the last element of a pipeline, do not affect the current environment as built-ins normally do.

It is strongly recommeded that you enclose the last element of a pipeline in parentheses if it is a built-in. Because of the implementation of pipelines, unknown results (or even hazardous results) may occur when using built-ins as the last element of a pipeline. This behavior will likely be changed in future implementations.

AUTHOR

David Korn, ATT Bell Laboratories, 11/16/88

NAME

labelit – provide labels for filesystems

SYNOPSIS

/etc/labelit special [fsname volume [**–n**]]

DESCRIPTION

labelit can be used to provide labels for unmounted disk filesystems. The **–n** option provides for initial tape labeling only. Because few tape devices are random access, writing labels at the front of a tape will destroy the previous contents. Labels can also be added to tapes as part of the **volcopy** procedure.

With the optional arguments omitted, **labelit** prints current label values.

The *special* name should be the physical disk section (such as */dev/rdsk/zd0s6*), or the cartridge tape (such as */dev/rmt/ts0*). The device may not be on a remote machine.

The *fsname* argument represents the mounted name (such as **root**, **usr**, etc.) of the filesystem.

volume may be used to equate an internal name to a volume name applied externally to the disk pack or tape.

For filesystems on disk, *fsname* and *volume* are recorded in the superblock.

SEE ALSO

sh(1), volcopy(1M), fs(4)

NAME

lex – generate programs for simple lexical tasks

SYNOPSIS

lex [**–ctvn**] [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 newline. 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 newline, and the character **$** at the end of an expression requires a trailing newline. 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**(*c*) to replace a character read; and **output**(*c*) 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**(*p*) pushes back the portion of the string matched beginning at *p*, which should be between *yytext* and *yytext+yyleng*. The macros *input* and *output* use files *yyin* and *yyout* to read from and write to, defaulted to *stdin* and *stdout*, 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.

EXAMPLES

```
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[yyleng-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 **–c** flag 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** suppresses the printing of 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)

NAME

line – read one line

SYNOPSIS

line

DESCRIPTION

line copies one line (up to a newline) 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 newline. 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 filename 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

rm(1), link(2), unlink(2)

NOTES

These commands can be run only by the superuser.

NAME

locate – use keywords to identify an operating system command

SYNOPSIS

[**help**] **locate**
[**help**] **locate** [keyword1 [keyword2] ...]

DESCRIPTION

The **locate** command is part of the Help Facility and provides on-line assistance for identifying operating system commands.

Without arguments, the **locate** command causes the initial **locate** screen to display. On this screen, a user can enter keywords and receive a list of operating system commands whose functional attributes match those in the keyword list, or the user can exit to the shell by typing q (for "quit").

For example, if you wish to print the contents of a file, enter the keywords **print** and **file**. The **locate** command would then print the names of all commands related to these keywords.

Keywords may also be entered directly from the shell, as shown above. In this case, the initial screen is not displayed and the resulting command list is printed.

More detailed information for a command in the list produced by **locate** can be obtained by accessing the **usage** module of the Help Facility. Access is made by entering the appropriate menu choice after the command list is displayed.

From any screen in the Help Facility, a user may execute a command via the shell by typing an exclamation mark (!) and the command to be executed. The screen will be redrawn if the command was entered at a first-level prompt. If entered at any other prompt level, only the prompt will be redrawn.

By default, the Help Facility scrolls the data that is presented to the user. If you prefer to have the screen clear before printing the data (non-scrolling), the shell variable **SCROLL** must be set to **no** and exported so it will become part of your environment. This is done by adding the following line to your *.profile* file (see **profile**(4)): "`export SCROLL ; SCROLL=no`". If you later decide that scrolling is desired, **SCROLL** must be set to **yes**.

Information on each of the Help Facility commands (**starter**, **locate**, **usage**, **glossary**, and **help**) is located on their respective manual pages.

SEE ALSO

csh(1), glossary(1), help(1), ksh(1), sh(1), starter(1), usage(1), term(5)

WARNINGS

If the shell variable **TERM** is not set in the user's *.profile* file, then **TERM** will default to the terminal value type 450 (a hard-copy terminal). For a list of valid terminal types, refer to **term**(5).

NAME

login – sign on

SYNOPSIS

login [**-p**] [username]
login [**-p**] [**-h** hostname] [**-f**] [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 can be invoked either 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 Ctrl-D to indicate an "end-of-file." (See "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 **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.

If you make any mistake in the login procedure, you will receive the following message and a new login prompt will appear.

```
Login incorrect
```

If you make five incorrect login attempts, all five may be logged in */usr/adm/loginlog* (if it exists) and the line will be dropped.

If you do not complete the login successfully within a certain period of time (for example, one minute), you are likely to be silently disconnected.

After a successful login, accounting files are updated, the procedure */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 (for example, **sh**) is initialized, and the file *.profile* in the working directory is executed, if it exists. (For C shell users, the *.login* and *.cshrc* files are executed.) These specifications are found in the */etc/passwd* file entry for the user. The name of the command interpreter is – followed by the last component of the interpreter's pathname (for example **-sh**). If this field in the password file is empty, then the default command interpreter (**/bin/sh**) is used. If this field is "*", then the named directory becomes the root directory, the starting point for path searches for pathnames beginning with a /. At that point **login** is re-executed at the new level, which must have its own root structure, including */etc/login* and */etc/passwd*.

The basic environment is initialized as follows:

HOME=*your-login-directory*
PATH=:/bin:/usr/bin
SHELL=*last-field-of-passwd-entry*
MAIL=/usr/mail/*your-login-name*
TZ=*timezone-specification*

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 *xxx* or *xxx=yyy*. Arguments without an equal sign are placed in the environment as follows:

L*n*=xxx

n is a number starting at 0 and is incremented each time a new variable name is required. Variables containing an = 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 users 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. The '–p' argument causes the previous environment to be preserved, otherwise any previous environment is discarded.

There are several additional options to *login* for use at initial login. With one exception, these options are available only to the superuser. The **–h** option is used by *telnetd*(8C) and other servers to list the host from which the connection was received. The **–f** option is used with a username on the command line to indicate that proper authentication has already been done and that no password need be requested. This option may be used by the superuser or by the user specified on the command line.

FILES

/etc/utmp	Accounting
/etc/wtmp	Accounting
/usr/mail/*your-name*	Mailbox for user *your-name*
/usr/adm/loginlog	Record of failed login attempts
/etc/motd	Message-of-the-day
/etc/passwd	Password file
/etc/profile	System profile for sh and ksh
.profile	User's login profile for sh and ksh
/etc/csh.login	System profile equivalent for csh
.login	User's login profile equivalent for csh

SEE ALSO

csh(1), ksh(1), mail(1), newgrp(1), sh(1), su(1M), loginlog(4), passwd(4), profile(4), environ(5)

DIAGNOSTICS

login incorrect
The user name or the password cannot be matched.

No shell, cannot open password file, or *no directory*
Consult a system programming counselor.

No utmp entry. You must exec "login" from the lowest level "sh"
You attempted to execute **login** as a command without using the shell's **exec** internal command or from other than the initial shell.

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)

NAME

lp, cancel – send/cancel requests to the LP print service

SYNOPSIS

lp [*printing-options*] *files*

lp –i *request-ids printing-options*

cancel [*request-ids*] [*printers*]

DESCRIPTION

The first form of the **lp** shell command arranges for the named files and associated information (collectively called a *request*) to be printed. If no filenames are specified on the shell command line, the standard input is assumed. The standard input may be specified along with named *files* on the shell command line by entering the filename(s) and specifying a – for the standard input. The *files* will be printed in the order they appear on the shell command line.

The second form of **lp** is used to change the options for a request. The print request identified by the *request-id* is changed according to the printing options specified with this shell command. The printing options available are the same as those with the first form of the **lp** shell command. If the request has finished printing, the change is rejected. If the request is already printing, it will be stopped and restarted from the beginning (unless the **–P** option has been given).

lp associates a unique *request-id* with each request and prints it on the standard output. This *request-id* can be used later to cancel, change, or find the status of the request. (See the section on **cancel** for details about canceling a request, the previous paragraph for an explanation of how to change a request, and **lpstat**(1) for information about checking the status of a print request.)

Sending a Print Request

The first form of the **lp** command is used to send a print request to a particular printer or group of printers.

Options to **lp** must always precede filenames, but may be listed in any order. The following options are available for **lp**:

–c Make copies immediately of the *files* to be printed when **lp** is invoked. Normally, *files* will not be copied. 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.

–d *dest* Print this request using *dest* as the printer or class of printers. Under certain conditions (lack of printer availability, capabilities of printers, and so on), 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 from system to system (see **lpstat**(1)).

-f *form-name* [**-d any**]
Print the request on the form *form-name*. The LP print service ensures that the form is mounted on the printer. If *form-name* is requested with a printer destination that cannot support the form, the request is rejected. If *form-name* has not been defined for the system or if the user is not allowed to use the form, the request is rejected (see **lpforms**(1M)). When the **-d any** option is given, the request is printed on any printer that has the requested form mounted and that can handle any other needs of the print request.

-H *special-handling*
Print the request according to the value of *special-handling*. Acceptable values for *special-handling* are **hold**, **resume**, and **immediate**, as defined below:

hold Don't print the request until notified. If printing has already begun, stop it. Other print requests will go ahead of a held request until it is resumed.

resume
Resume a held request. If it had been printing when held, it will be the next request printed, unless subsequently bumped by an **immediate** request.

immediate (Available only to LP administrators)
Print the request next. If more than one request is specified for **immediate** printing, the requests are printed in the reverse order queued. If a request is currently printing on the desired printer, you must put it on hold to allow the immediate request to print.

-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.

-n *number* Print *number* copies of the output. (Default is 1.)

-o *option* Specify printer-dependent or class-dependent *options*. Several such options may be specified on a single command line either by using the **-o** keyletter more than once (**-o** $option_1$ **-o** $option_2$ ... **-o** $option_n$), or by specifying a list of options with a single **-o** keyletter (**-o** $option_1$, $option_2$, ... $option_n$). The standard interface recognizes the following options:

nobanner
Do not print a banner page with this request. (The administrator can disallow this option at any time.)

nofilebreak
If submitting a job to print more than one file, Do not insert a form feed between the files.

length=*scaled-decimal-number*
Print this request with pages *scaled-decimal-number* lines long. A *scaled-decimal-number* is an optionally scaled decimal number that gives a size in lines, columns, inches, or

centimeters, as appropriate. The scale is indicated by appending the letter **i** for inches, or the letter **c** for centimeters. For length or width settings, an unscaled number indicates lines or columns; for line pitch or character pitch settings, an unscaled number indicates lines per inch or characters per inch (the same as a number scaled with **i**). For example, **length=66** indicates a page length of 66 lines, **length=11i** indicates a page length of 11 inches, and **length=27.94c** indicates a page length of 27.94 centimeters.

This option cannot be used with the **–f** option.

width=*scaled-decimal-number*
Print this request with page-width set to *scaled-decimal-number* columns wide. (See the explanation of *scaled-decimal-numbers* in the discussion of **length**, above.) This option cannot be used with the **–f** option.

lpi=*scaled-decimal-number*
Print this request with the line pitch set to *scaled-decimal-number* lines per inch. This option cannot be used with the **–f** option.

cpi=*scaled-decimal-number*
Print this request with the character pitch set to *scaled-decimal-number* characters per inch. Character pitch can also be set to **pica** (representing 10 columns per inch) or **elite** (representing 12 columns per inch), or it can be **compressed** (representing as many columns as a printer can handle). There is no standard number of columns per inch for all printers; see the Terminfo database (**terminfo**(4)) for the default character pitch for your printer.

This option cannot be used with the **–f** option.

stty=*stty-option-list*
A list of options valid for the **stty** command; enclose the list in quotes if it contains blanks.

–P *page-list* Print the pages specified in *page-list*. This option can be used only if there is a filter available to handle it; otherwise, the print request will be rejected.

The *page-list* may consist of range(s) of numbers, single page numbers, or a combination of both. The pages will be printed in ascending order.

–q *priority-level*
Assign this request *priority-level* in the printing queue. The values of *priority-level* range from 0, the highest priority, to 39, the lowest priority. If a priority is not specified, the default for the print service is used, as assigned by the system administrator.

–s Suppress messages from the print service such as `request id is` *request-id*.

–S *character-set* [**–d any**]
–S *print-wheel* [**–d any**]
Print this request using the specified *character-set* or *print-wheel*. If a form was requested and it requires a character set or print wheel other than the one specified with the **–S** option, the request is rejected.

For printers that take print wheels: If the print wheel specified is not one listed by the administrator as acceptable for the printer specified in this request, the request is rejected unless the print wheel is already mounted on the printer.

For printers that use selectable or programmable character sets: If the *character-set* specified is not one defined in the Terminfo database for the printer (see **terminfo**(4)) or is not an alias defined by the administrator, the request is rejected.

When the **–d any** option is used, the request is printed on any printer that has the print wheel mounted or can select the character set and can also handle any other needs of the request.

–t *title* Print *title* on the banner page of the output. The default is no title.

–T *content-type* [**–r**]
Print the request on a printer that can support the specified *content-type*. If no printer accepts this type directly, a filter will be used to convert the content into an acceptable type. If the **–r** option is specified, a filter will not be used. If **–r** is specified but no printer accepts the *content-type* directly, the request is rejected. If the *content-type* is not acceptable to any printer, either directly or with a filter, the request is rejected.

–w Write a message on the user's terminal after the *files* have been printed. If the user is not logged in, mail will be sent instead.

–y *mode-list* Print this request according to the printing modes listed in *mode-list*. The allowed values for *mode-list* are locally defined. This option can be used only if there is a filter available to handle it; otherwise, the print request will be rejected.

Canceling a Print Request

The **cancel** command cancels printer requests that were made by the **lp**(1) shell command. The shell 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 that 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

enable(1), lpstat(1), mail(1), accept(1M), lpadmin(1M), lpfilter(1M), lpforms(1M), lpsched(1M), lpusers(1M), terminfo(4)

WARNINGS

For printers that take mountable print wheels or font cartridges, if you do not specify a particular print wheel or font with the **–S** option, whichever one happens to be mounted at the time your request is printed will be used. Use the **lpstat –p** *printer* **–l** command to see which print wheels are available on a particular printer, or the **lpstat –S –l** command to see the print wheels that are available and on which printers. For printers that have selectable character sets, you will get the standard character set if you don't use the **–S** option.

NOTES

Printers for which requests are not being accepted will not be considered when the destination is **any**. (Use the **lpstat –a** command to see which printers are accepting requests.) On the other hand, if a request is destined for a class of printers and the class itself is accepting requests, *all* printers in the class will be considered, regardless of their acceptance status, as long as the printer class is accepting requests.

NAME

lpadmin – configure the LP print service

SYNOPSIS

/usr/lib/lpadmin –p *printer options*
/usr/lib/lpadmin –x *dest*
/usr/lib/lpadmin –d [*dest*]
/usr/lib/lpadmin –S *print-wheel* **–A** *alert-type* [**–W** *integer1*] [**–Q** *integer2*]

DESCRIPTION

lpadmin configures the LP print service by defining printers and devices. It is used to add and change printers, to remove printers from the service, to set or change the system default destination, and to define alerts for print wheels.

Adding or Changing a Printer

The first form of the **lpadmin** command (**lpadmin –p** *printer options*) is used to configure a new printer or to change the configuration of an existing printer. The following *options* are used, and can appear in any order. For ease of discussion, the printer will be referred to as *P* below.

–F *fault-recovery*
Restore the LP print service after a printer fault, according to the value of *fault-recovery*:

continue
Continue printing on the top of the page where printing stopped. This requires a filter that waits for the fault to clear before automatically continuing.

beginning
Start printing the request again from the beginning.

wait
Disable printing on the printer and wait for the administrator or a user to enable printing again.

During the wait, the administrator or the user who submitted the stopped print request can issue a change request that specifies where printing should resume. If no change request is made before printing is enabled, printing will resume at the top of the page where stopped, if the filter allows; otherwise, the request will be printed from the beginning.

This option specifies the recovery method to be used for any print request that is stopped because of a printer fault.

–c *class*
Insert printer *P* into the specified *class*. *Class* will be created if it does not already exist.

–D *comment*
Save this *comment* for display whenever a user asks for a full description of the printer *P* (see **lpstat**(1)). The LP print service does not interpret this comment.

–e *printer*
Copy an existing *printer's* interface program. The copy will be used as the interface program for printer *P*.

-f allow:*form-list*
-f deny:*form-list*

Allows (**-f allow**) or denies (**-f deny**) the forms in *form-list* to be printed on printer *P*.

For each printer, the LP print service keeps two lists of forms: an "allow-list" of forms that can be used with the printer, and a "deny-list" of forms that shouldn't be used with the printer. With the **-f allow** option, the forms listed are added to the allow-list and removed from the deny-list. With the **-f deny** option, the forms listed are removed from the allow-list and added to the deny-list.

If the allow-list is not empty, only the forms in the list can be used with the printer, regardless of the content of the deny-list. If the allow-list is empty but the deny-list is not, the forms in the deny-list cannot be used with the printer. All forms can be excluded from a printer by having an empty allow-list and putting the word **any** in the deny-list. All forms can be used on a printer by having an empty deny-list and specifying **any** for the allow-list, provided the printer can handle all the characteristics of the forms.

The LP print service uses this information as a set of guidelines for determining where a form can be mounted. Administrators, however, are not restricted from mounting a form on any printer. If mounting a form on a particular printer is in disagreement with the information in the allow-list or deny-list, the administrator is warned but the mount is accepted. Nonetheless, if a user attempts to issue a print or change request for a form and printer combination that is in disagreement with the information, the request is accepted only if the form is currently mounted on the printer. If the form is later unmounted before the request can print, the request is canceled and the user is notified by mail.

If an administrator tries to name a form as acceptable for use on a printer that doesn't have the capabilities needed by the form, the command is rejected.

Note the other use of **-f** below.

-h Indicates that the device associated with *P* is hardwired. This option is assumed when adding a new printer unless the **-l** option is supplied.

-i *interface*

Establishes a new interface program for *P*. *interface* is the pathname of the new program.

-I *content-type-list*

Assigns *P* to handle print requests with content of a type listed in *content-type-list*.

The type **simple** is recognized as the default content-type of files in the operating system. Such a data stream contains only printable ASCII characters and the following control characters.

Control Character	Octal Value	Meaning
backspace	010	move back to previous column, except at beginning of line
tab	011	move to next tab stop
linefeed (newline)	012	move to beginning of next line
form feed	014	move to beginning of next page
carriage return	015	move to beginning of current line

To force the print service to not consider **simple** as a valid type for the printer, give an explicit value (for example, the printer type) in the *content-type-list*. Vice versa, if you do want **simple** included along with other types, you must include **simple** in the *content-type-list*.

Each printer automatically has its printer type included in the list of content types it will accept.

Except for **simple**, each *content-type* name is freely determined by the administrator. If names given as content types are also printer types, the names are accepted without comment, because the LP print service recognizes all printer types as potential content types as well.

–l 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 by using **lpadmin**.

–M –f *form-name* [**–a** [**–o filebreak**]]
Mounts the form *form-name* on *P*. Print requests to be printed with the pre-printed form *form-name* will be printed on *P*. If more than one printer has the form mounted and the user has specified `any` (with the **–d** option of the **lp** command) as the printer destination, the print request will be printed on the printer that also meets the other needs of the request.

The page length, page width, and character and line pitches needed by the form are compared with those allowed for the printer. This is done by checking the capabilities of that type of printer in the **terminfo**(4) database. If the form requires attributes that are not available with the printer, the administrator is warned but the mount is accepted. If the form lists a print wheel as mandatory but the print wheel mounted on the printer is different, the administrator is also warned but the mount is accepted.

If the **–a** option is given, an alignment pattern is printed, preceded by the same initialization of the physical printer that precedes a normal print request. There is one exception: no banner page is printed. Printing is assumed to start at the top of the first page of the form. After the pattern is printed, the administrator can adjust the mounted form in the printer and press Return for another alignment pattern (no initialization this time), and can continue printing as many alignment patterns as desired. The administrator can quit the printing alignment patterns by typing **q**.

If the **–o filebreak** option is given, a formfeed is inserted between each copy of the alignment pattern. By default, the alignment pattern is assumed to correctly fill a form, so no formfeed is added.

A form is "unmounted" by mounting a new form in its place using the **–f** option or the **–f none** option. By default, a new printer has no form mounted.

Note the other use of **–f** above.

–M –S *print-wheel*
Mounts the *print-wheel* on *P*. Print requests to be printed with *print-wheel* will be printed on *P*. If more than one printer has *print-wheel* mounted and the user has specified `any` (with the **–d** option of the **lp** command) as the printer destination, the print request will be printed on the printer that also meets the other needs of the request.

If the *print-wheel* is not listed as acceptable for the printer, the administrator is warned but the mount is accepted. If the printer does not take print wheels, the command is rejected.

A print wheel is "unmounted" by mounting a new print wheel in its place or by using the option **–S none**.

By default, a new printer has no special print wheel mounted. Until this is changed, a print request that asks for a specific print wheel will not be printed on *P*.

Note the other uses of the **–S** option described below.

–m *model*
Selects a model interface program, provided with the LP print service, for printer *P*.

–o *printing-option*
Each **–o** option in the list below is the default given to an interface program if the option is not taken from a preprinted form description or is not explicitly given by the user submitting a request (see **lp**(1)). The only **–o** options that can have defaults defined are listed below.

length=*scaled-decimal-number*
width=*scaled-decimal-number*
cpi=*scaled-decimal-number*
lpi=*scaled-decimal-number*
stty=*'stty-option-list'*

The term *scaled-decimal-number* refers to a non-negative number used to indicate a unit of size. The type of unit is shown by a trailing letter attached to the number. Three types of scaled decimal numbers can be used with the LP print service: numbers that show sizes in centimeters (marked with a trailing **c**); numbers that show sizes in inches (marked with a trailing **i**); and numbers that show sizes in units appropriate to use (without a trailing letter), such as lines, columns, lines per inch, or characters per inch.

The first four default option values should agree with the capabilities of the type of physical printer, as defined in the **terminfo**(4) database for

the printer type. If they do not, the command is rejected.

The *stty-option-list* is not checked for allowed values, but is passed directly to the **stty**(1) program by the standard interface program. Any error messages produced by **stty**(1) when a request is processed (by the standard interface program) are mailed to the user submitting the request.

For each printing option not specified, the defaults for the **length**, **width**, **cpi**, and **lpi** attributes are defined in the Terminfo entry for the specified printer type.

The default for **stty** is as follows:

stty=9600 cs8 –cstopb –parenb –paroff ixon
–ixany opost –olcuc –onlcr –ocrnl –onocr
–onlret –ofill nl0 cr0 tab0 bs0 vt0 ff0

You can set any of the **–o** options to the default values (which vary for different types of printers), by typing them without assigned values, as shown below.

length=
width=
cpi=
lpi=
stty=

–o nobanner
Allows users to submit a print request that asks that no banner page be printed.

–o banner
Forces a banner page to be printed with every print request, even when a user asks for no banner page. This is the default; you must specify **–o nobanner** if you want to allow users to specify **–o nobanner** with the **lp** command.

–r *class*
Removes printer *P* from the specified *class*. If *P* is the last member of the *class*, then *class* will be removed.

–S *list* Allows the print wheels or aliases for character sets named in *list* to be used with *P*.

If the printer is a type that takes print wheels, then *list* is a list of print wheel names. The names in the list are separated by commas or spaces. (Enclose the list in quotes if it contains blanks.) These will be the only print wheels considered mountable on the printer. (You can always force a different print wheel to be mounted, however.) Until the option is used to specify a list, no print wheels will be considered mountable on the printer, and print requests that ask for a particular print wheel with this printer will be rejected.

If the printer is a type that has selectable character sets, then *list* contains character set name *mappings* or aliases. The names in the list are separated by,commas or spaces. (Enclose the list in quotes if it

contains blanks.) Each mapping has this form:

known-name=alias

The *known-name* is either a character set number preceded by **cs** (such as **cs3** for character set three) or a character set name from the Terminfo database **csnm** entry. (See ***terminfo***(4).) If this option is not used to specify a list, only the names already known from the Terminfo database or numbers with a prefix of **cs** will be acceptable for the printer.

If *list* is the word **none**, any existing print wheel list or character set aliases will be removed.

Note the other uses of the **–S** option.

–T *printer-type*

Assigns the given *printer-type*, a representation of a physical printer of type *printer-type*. *printer-type* is used to extract data from **terminfo**(4). This data is used to initialize the printer before printing each user's request. Some filters may also use *printer-type* to convert content for the printer. If this option is not used, the default *printer-type* will be **unknown**; no useful information will be extracted from **terminfo**(4) and each user request will be printed without first initializing the printer. Also, this option must be used if the following are to work: the **–o cpi=**, **–o lpi=**, **–o width=**, and **–o length=** options of the **lpadmin** and **lp** commands, and the **–S** and **–f** options of the **lpadmin** command.

–u allow:*user-list*
–u deny:*user-list*

Allows (**–u allow**) or denies (**–u deny**) the users in *user-list* access to *P*.

For normal access to each printer, the LP print service keeps two lists of users: an "allow-list" of people allowed to use the printer, and a "deny-list" of people denied access to the printer. With the **–u allow** option, the users listed are added to the allow-list and removed from the deny-list. With the **–u deny** option, the users listed are removed from the allow-list and added to the deny-list.

If the allow-list is not empty, the users in the list are allowed access to the printer and all others are denied access, regardless of the content of the deny-list. If the allow-list is empty, but the deny-list is not, the users in the deny-list are denied access and all others are allowed. If both lists are empty, all users are allowed access. Access can be denied to all users, except the LP print service administrator, by putting **any** in the deny-list. To allow everyone access to *P* and effectively empty both lists, put **any** in the allow-list.

–U *dial-info*

Assign the dialing information *dial-info* to the printer. *Dial-info* is used with the **dial**(3) routine to call the printer. Any network connection supported by the Basic Networking Utilities will work. *Dial-info* can be either a phone number for a modem connection or a system

name for other kinds of connections. If **–U direct** is given, no dialing will take place, because the name **direct** is reserved for a printer that is directly connected. If a system name is given, it is used to search for connection details from the file */usr/lib/uucp/Systems* or related files. The Basic Networking Utilities are required to support this option. By default, **–U direct** is assumed.

–v *device*

Associates a *device* with printer *P*. *device* is the pathname of a file that is writeable by **lp**. Note that the same *device* can be associated with more than one printer.

–A *alert-type* [**–W** *integer*]

The **–A** option is used to define an alert-type to inform the administrator when a printer fault is detected, and periodically thereafter, until the printer fault is cleared by the administrator. The *alert-types* are as follows:

mail Send the alert message via mail (see **mail**(1)) to the administrator who issues this command.

write Write the message to the terminal on which the administrator is logged in. If the administrator is logged in on several terminals, one is chosen arbitrarily.

quiet Do not send messages for the current condition. An administrator can use this option to temporarily stop receiving further messages about a known problem. Once the fault has been cleared and printing resumes, messages will again be sent when another fault occurs with the printer.

none Do not send messages; any existing alert definition for the printer will be removed. No alert will be sent when the printer faults until a different alert-type (except **quiet**) is used.

shell-command

The *shell-command* is run each time the alert needs to be sent. The shell command should expect the message as standard input. If there are blanks embedded in the command, enclose the command in quotes. Note that the **mail** and **write** values for this option are equivalent to the values **mail** *user-name* and **write** *user-name* respectively, where *user-name* is the current name for the administrator. This will be the login name of the person submitting this command *unless* he or she has used the **su**(1) command to change to another user ID. If the **su**(1) command has been used to change the user ID, then the *user-name* for the new ID is used.

list The type of the alert for the printer fault is displayed on the standard output. No change is made to the alert.

The message appears as follows:

```
The printer printer-name has stopped printing for the reason
given below.  Fix the problem and bring the printer back on
```

```
line.  Printing has stopped, but will be restarted in a few
minutes; issue an enable command if you want to restart
sooner.  Unless someone issues a change request

        lp -i request-id -P ...

to change the page list to print, the current request will be
reprinted from the beginning.

The reason(s) it stopped (multiple reasons indicate reprinted
attempts): reason
```

The LP print service can detect printer faults only through an adequate fast filter and only when the standard interface program or a suitable customized interface program is used. Furthermore, the level of recovery after a fault depends on the capabilities of the filter.

If the *printer-name* is **all**, the alerting defined in this command applies to all existing printers.

If the **–W** option is not used to arrange fault alerting for a printer, the default procedure is to mail one message to the administrator of the printer per fault. Similarly, if *integer* is zero, only one message will be sent per fault. If *integer* is a non-zero number, an alert will be sent every *integer* minute(s).

Restrictions

When creating a new printer, either the **–v** or the **–U** option must be supplied. In addition, only one of the following may be supplied: **–e**, **–i**, or **–m**; if none of these three options is supplied, the model standard is used. The **–h** and **–l** keyletters are mutually exclusive. Printer and class names can be no longer than 14 characters and must consist entirely of the characters **A-Z**, **a-z**, **0-9** and _ (underscore).

Removing a Printer Destination

The **–x** *dest* option removes the destination *dest* from the LP print service. If *dest* is a printer and is the only member of a class, then the class will be deleted too. If *dest* is **all**, all printers and classes are removed. No other *options* are allowed with **–x**.

Setting/Changing the System Default Destination

The **–d** [*dest*] option makes *dest* (which must be an existing destination) the new system default destination. If *dest* is not supplied, then there is no system default destination. No other *options* are allowed with **–d**.

Setting an Alert for a Print Wheel

–S *print-wheel* **–A** *alert-type* [**–W** *integer1*] [**–Q** *integer2*]

The **–S** *print-wheel* option is used with the **–A** *alert-type* option to send the alert *alert-type* to the administrator to mount the print wheel when there is one or more jobs queued for it. If this command is not used to arrange alerting for a print wheel, no alert will be sent for the print wheel. The *alert-types* are as follows:

mail Send the alert message via mail (see **mail**(1)) to the administrator who issues this command.

write Write the message to the terminal on which the administrator is logged in. If the administrator is logged in on several terminals, one is arbitrarily chosen.

quiet Do not send messages for the current condition. An administrator can use this option to temporarily stop receiving further messages about a known problem. Once the *print-wheel* has been mounted and subsequently unmounted, messages will again be sent when the number of print requests again exceeds the threshold.

none Do not send messages until the **–A** option is given again with a different *alert-type* (other than **quiet**).

shell-command
The *shell-command* is run each time the alert needs to be sent. The shell command should expect the message as standard input. If there are blanks embedded in the command, enclose the command in quotes. Note that the **mail** and **write** values for this option are equivalent to the values **mail** *user-name* and **write** *user-name* respectively, where *user-name* is the current name for the administrator. This will be the login name of the person submitting this command *unless* he or she has used the **su**(1) command to change to another user ID. If the **su**(1) command has been used to change the user ID, then the *user-name* for the new ID is used.

list The type of the alert for the print wheel is displayed on the standard output. No change is made to the alert.

The message sent appears as follows:

```
The print wheel print-wheel needs to be mounted
on the printer(s):
printer  (integer3 requests)
integer4 print requests await this print wheel.
```

The printers listed are those that the administrator had earlier specified were candidates for this print wheel. The number (*integer3*) listed next to each printer is the number of requests eligible for the printer. The number (*integer4*) shown after the printer list is the total number of requests waiting for the print wheel. It will be less than the sum of the other numbers if some requests can be handled by more than one printer.

If the *print-wheel* is **all**, the alerting defined in this command applies to all print wheels already defined to have an alert.

If the **–W** option is not given or *integer1* is zero (which is interpreted as **once** and is also the default), only one message will be sent per need to mount a print wheel. If *integer1* is a non-zero number, an alert is sent every *integer1* minute(s).

If the **–Q** option is also given, the alert will be sent when *integer2* print requests that need the print wheel are waiting. If the **–Q** option is not given, or *integer2* is 1 or the word **any** (which are both the default), a message is sent as soon as anyone submits a print request for the print

wheel when it is not mounted.

The **–S** option has a different meaning when used with the **–p** option.

FILES

/usr/spool/lp/*

SEE ALSO

accept(1M), enable(1), lp(1), lpsched(1M) lpstat(1), stty(1), dial(3), terminfo(4)

NAME

lpfilter – administer filters used with the LP print service

SYNOPSIS

/usr/lib/lpfilter –f *filter-name* **–F** *pathname*
/usr/lib/lpfilter –f *filter-name* **–**
/usr/lib/lpfilter –f *filter-name* **–i**
/usr/lib/lpfilter –f *filter-name* **–x**
/usr/lib/lpfilter –f *filter-name* **–l**

DESCRIPTION

The **lpfilter** command is used to add, change, delete, and list filters used with the LP print service. These filters are used to convert the content type of a file to a content type acceptable to a given printer. One of the following options must be used with the **lpfilter** command: **–F** *pathname* (or – for standard input) to add or change a filter, **–i** to reset an original LP print service filter to its factory setting, **–x** to delete a filter, or **–l** to list a filter description.

The argument **all** can be used instead of a *filter-name* with any of these options. When **all** is specified with the **–F** or – option, the requested change is made to all filters. Using **all** with the **–i** option has the effect of restoring to their original settings all filters for which predefined settings were initially available. Using the **all** argument with the **–l** option produces a list of all filters, and using it with the **–x** option results in all filters being deleted.

Adding or Changing a Filter

The filter named in the **–f** option and described in the input is added to the filter table. If the filter already exists, its description is changed to reflect the new information in the input. Once added, a filter is available for use.

The filter description is taken from the *pathname* if the **–F** option is given, or from the standard input if the – option is given. One of the two must be given to define or change a filter. If the filter named is one originally delivered with the LP print service, the **–i** option will restore the original filter description.

When an existing filter is changed with the **–F** or – option, items that are not specified in the new information are left as they were. When a new filter is added with this command, unspecified items are given default values.

Note that a filter name and a command must be given. A filter with no input type value is assumed to work with any input type; this is also true for the output type, printer type, and printer values.

Filters are used to convert the content of a request into a data stream acceptable to a printer. For a given print request, the LP print service will know the following: the type of content in the request, the name of the printer, the type of the printer, the types of content acceptable to the printer, and the modes of printing asked for by the originator of the request. It will use this information to find a filter that will convert the content into a type acceptable to the printer.

Below is a list of items that provide input to this command and a description of each item. All lists are separated by commas or spaces.

Input types: *content-type-list*
Output types: *content-type-list*
Printer types: *printer-type-list*

Printers: *printer-list*
Filter type: *filter-type*
Command: *shell-command*
Options: *template-list*

Input types
This gives the types of content that can be accepted by the filter.

Output types
This gives the types of content that the filter can produce from any of the input content types.

Printer types
This gives the type of printers for which the filter can be used. The LP print service will restrict the use of the filter to these types of printers.

Printers
This gives the names of the printers for which the filter can be used. The LP print service will restrict the use of the filter to just the printers named.

Filter type
This marks the filter as a *slow* filter or a *fast* filter. Slow filters are generally those that take a long time to convert their input. They run unconnected to a printer, to keep the printers from being tied up while the filter is running. Fast filters are generally those that convert their input quickly or those that must be connected to the printer when run. These will be given to the interface program to run connected to the physical printer.

Command
This specifies the program to run to invoke the filter. The program name as well as fixed options are included in the *shell-command*; additional options are constructed, based on the characteristics of each print request and on the **Options** field.

The command must accept a data stream as standard input and produce the converted data stream on its standard output. This allows filter pipelines to be constructed to convert data not handled by a single filter.

Options
This is a list of templates, separated by commas, that is used by the LP print service to construct options to the filter from the print request characteristics listed in the following table.

In general, each template is of the following form:

keyword pattern = replacement

The *keyword* names the characteristic that the template attempts to map into a filter specific option; each valid *keyword* is listed in the table below. A *pattern* is either a literal pattern of one of the forms listed in the table or a single asterisk (*). If the *pattern* matches the value of the characteristic, the template fits and is used to generate a filter-specific option. A *pattern* of * matches any value. The *replacement* is a string used as a filter specific option. If an asterisk (*) is

used, it is replaced with the value of the characteristic.

lp Option	Characteristic	*keyword*	Possible *patterns*
–T	Content type (input)	**INPUT**	*content-type*
N/A	Content type (output)	**OUTPUT**	*content-type*
N/A	Printer type	**TERM**	*printer-type*
–f, –o cpi=	Character pitch	**CPI**	*integer*
–f, –o lpi=	Line pitch	**LPI**	*integer*
–f, –o length=	Page length	**LENGTH**	*integer*
–f, –o width=	Page width	**WIDTH**	*integer*
–P	Pages to print	**PAGES**	*page-list*
–S	Character set/ print wheel	**CHARSET**	*character-set-name*/ *print-wheel-name*
–f	Form name	**FORM**	*form-name*
–y	Modes	**MODES**	*mode*
–n	Number of copies	**COPIES**	*integer*

For example, the following template would show that if a print request was submitted with the **–y landscape** option, the filter should be given the option **–l**.

MODES landscape = –l

As another example, the following template would show that the filter should be given the option **–T** *printer-type* for whichever *printer-type* is associated with a print request using the filter.

TERM * = –T *

Deleting a Filter

The **–x** option is used to delete the filter specified in *filter-name* from the LP filter table.

Listing a Filter Description

The **–l** option is used to list the description of the filter named in *filter-name*. If the command is successful, the following message is sent to standard output:

```
Input types: content-type-list
Output types: content-type-list
Printer types: printer-type-list
Printers: printer-list
Filter type: filter-type
Command: shell-command
Options: template-list
```

If the command fails, an error message is sent to standard error.

SEE ALSO

lp(1), lpadmin(1M)

NAME

lpforms – administer forms used with the LP print service

SYNOPSIS

/usr/lib/lpforms –f *form-name options*

/usr/lib/lpforms –f *form-name* **–A** *alert-type* [–Q *integer*$_1$] [–W *integer*$_2$]

/usr/lib/lpforms –f *form-name* **–A list**

/usr/lib/lpforms –f *form-name* **–A quiet**

/usr/lib/lpforms –f *form-name* **–A none**

DESCRIPTION

The **lpforms** command is used to administer the use of preprinted forms such as company letterhead paper with the LP print service. A form is specified by the *form-name* given with the **lpforms** command. Users may request a form by *form-name* (see **lp**(1)). The argument **all** can be used instead of *form-name* with any of the five **lpforms** command lines shown above. The first command line allows the administrator to add, change, and delete forms, to list the attributes of an existing form, and to allow and deny users access to particular forms. The second command line is used to establish the method for alerting the administrator when a form must be mounted on a printer. The third command line lists the current alerting methods assigned to forms. The fourth command line is used to terminate an active alert, and the fifth command line is used to remove an alert definition.

With the first **lpforms** command line, one of the following options must be used:

–F *pathname*
: To add or change a form as specified by the information in *pathname*

–
: To add or change a form and supply information from standard input

–x
: To delete a form (this option must be used separately; it cannot be used with any other option)

–l
: To list the attributes of a form

–u allow:*user-list*
: To allow users to request a form (this option can be used with the **–F** or **–** option)

–u deny:*user-list*
: To deny users access to a form (this option can be used with the **–F** or **–** option)

Each option is explained below.

Adding or Changing a Form

The **–F** *pathname* option is used to add a new form to the LP print service or to change the attributes of an existing form. The form description is taken from *pathname* if the **–F** option is given, or the standard input if the **–** option is used. One of the two options must be used to define or change a form. *pathname* is the pathname of a file containing all or any subset of the following information about the form.

Page length: *scaled-decimal-number*$_1$
Page width: *scaled-decimal-number*$_2$
Number of pages: *integer*
Line pitch: *scaled-decimal-number*$_3$

Character pitch: *scaled-decimal-number*$_4$
Character set choice: *character-set*/*print-wheel*,[**mandatory**]
Ribbon color: *ribbon-color*
Comment:
comment
Alignment pattern: [*content-type*]
content

Except for the last two lines, the above lines can appear in any order. The **Comment:** and *comment* items must appear in consecutive order but can appear before the other items, and the **Alignment pattern:** and the *content* items must appear in consecutive order at the end of the file. Also, the *comment* item cannot contain a line that begins with any of the key phrases above, unless the key phrase is preceded with a > sign. Any leading > sign found in the *comment* will be removed when the comment is displayed. Case distinctions in the key phrases are ignored.

When this command is issued, the form specified by *form-name* is added to the list of forms. If the form already exists, its description is changed to reflect the new information in the input. Once added, a form is available for use in a print request, except where access to the form has been restricted as described under the **-u allow:** option. A form can also be restricted to certain printers only.

Following is a description of each form attribute:

Page length and page width

Before printing the content of a print request needing this form, the generic interface program provided with the LP print service will initialize the physical printer to handle pages *scaled-decimal-number*$_1$ long, and *scaled-decimal-number*$_2$ wide using the printer type as a key into the **terminfo**(4) database. A *scaled-decimal-number* is an optionally scaled decimal number that gives a size in lines, columns, inches, or centimeters, as appropriate. The scale is indicated by appending the letter **i** for inches or the letter **c** for centimeters. For length or width settings, an unscaled number indicates lines or columns; for line pitch or character pitch settings, an unscaled number indicates lines per inch or characters per inch (the same as a number scaled with **i**). For example, **length=66** indicates a page length of 66 lines, **length=11i** indicates a page length of 11 inches, and **length=27.94c** indicates a page length of 27.94 centimeters.

The page length and page width will also be passed, if possible, to each filter used in a request needing this form.

Number of pages

Each time the alignment pattern is printed, the LP print service will attempt to truncate the *content* to a single form by, if possible, passing to each filter the page subset of 1-*integer*.

Line pitch and character pitch

Before printing the content of a print request needing this form, the interface programs provided with the LP print service will initialize the physical printer to handle these pitches, using the printer type as a key into the **terminfo**(4) database. Also, the pitches will be passed, if possible, to each filter used in a request needing this form. *Scaled-decimal-number*$_3$ is in lines per centimeter if a **c** is appended, and lines per inch otherwise. Similarly, *scaled-decimal-number*$_4$ is in columns per centimeter if a **c** is appended, and columns per inch otherwise. The character pitch can also be given as **elite** (12 characters per inch), **pica** (10 characters per inch), or **compressed** (as many characters per inch as possible).

Character set choice

When the LP print service alerts an administrator to mount this form, it will also mention that *print-wheel* should be used on those printers that take print wheels. If printing with this form is to be done on a printer that has selectable or loadable character sets instead of print wheels, the interface programs provided with the LP print service will automatically select or load the correct character set. If **mandatory** is appended, a user is not allowed to select a different character set for use with the form. Otherwise, the character set or print wheel named is a suggestion and a default only.

Ribbon color

When the LP print service alerts an administrator to mount this form, it will also mention that the color of the ribbon should be *ribbon-color*.

Comment

The LP print service will display the *comment* unaltered when a user asks about this form (see **lpstat**(1)).

Alignment pattern

When mounting this form, an administrator can ask for the *content* to be printed repeatedly, as an aid in correctly positioning the preprinted form. The optional *content-type* defines the type of printer for which *content* had been generated. If *content-type* is not given, **simple** is assumed. Note that the *content* is stored as given, and will be readable only by the user **lp**.

When an existing form is changed with this command, items missing in the new information are left as they were. When a new form is added with this command, missing items will receive the following defaults:

Page Length: **66**
Page Width: **80**
Number of Pages: **1**
Line Pitch: **6**
Character Pitch: **10**
Character Set Choice: **any**
Ribbon Color: **any**
Comment: (no default)
Alignment Pattern: (no default)

Deleting a Form

The **-x** option is used to delete the form specified in *form-name* from the LP print service.

Listing Form Attributes

The **-l** option is used to list the attributes of the existing form specified by *form-name*. The attributes listed are those described under "Adding and Changing a Form," above. Because of the potentially sensitive nature of the alignment pattern, only the administrator can examine the form with this command. Other people can use the **lpstat**(1) command to examine the non-sensitive part of the form description.

Allowing/Denying Access to a Form

The LP print service keeps two lists of users for each form: an "allow-list" of people allowed to use the form, and a "deny-list" of people denied access to the form. With the **-u allow:** option, the users listed are added to the allow-list and removed from the deny-list. With the **-u deny:** option, the users listed are removed from the allow-list and added to the deny-list.

If the allow-list is not empty, the users in the list are allowed access to the form and all others are denied access, regardless of the content of the deny-list. If the allow-list is empty, but the deny-list is not, the users in the deny-list are denied access and all others are allowed. If both lists are empty, all users are allowed access. Access can be denied to all users, except the LP print service administrator, by putting **any** in the deny-list. To effectively empty both lists, allowing access for everyone, put **any** in the allow-list.

Alerting to Mount Forms

The second **lpforms** command line (shown under **SYNOPSIS**) is used to arrange for alerts to be sent to the administrator when forms need to be mounted on a printer.

When $integer_1$ print requests needing the preprinted form *form-name* become queued up because no printer satisfying all the needs of the requests has the form mounted (and for as long as this condition remains), an alert is sent to the administrator every $integer_2$ minutes until the form is mounted on a qualifying printer. If the *form-name* is **all**, the alerting defined in this command applies to all existing forms. If the administrator does not use this command, no alerting is done for a backlog of print requests needing a form.

The method by which the alert is sent depends on the value of the **-A** option.

write The message is sent via **write**(1) to the terminal on which the administrator is logged in when the alert arises. If the administrator is logged in on several terminals, one is arbitrarily chosen.

mail The message sent via **mail**(1) to the administrator who issues this command.

The message sent appears as follows:

```
The form form-name needs to be mounted on the printer(s)
printer (integer5 requests).
integer4 print requests await this form.
Use the ribbon-color ribbon.
Use the print-wheel print wheel, if appropriate.
```

The printers listed are those that the administrator had earlier specified were candidates for this form. The number (*integer*$_3$) listed next to each printer is the number of requests eligible for the printer. The number (*integer*$_4$) shown after the list of printers is the total number of requests awaiting the form. It will be less than the sum of the other numbers if some requests can be handled by more than one printer. The *ribbon-color* and *print-wheel* are those specified in the form description. The last line in the message is always sent, even if none of the printers listed use print wheels, because the administrator may choose to mount the form on a printer that does use a print wheel.

Where any color ribbon or any print wheel can be used, the statements above will read:

```
Use any ribbon.
Use any print-wheel.
```

shell-command

The *shell-command* is run each time the alert needs to be sent. The shell command should expect the message as standard input. Note that the **mail** and **write** values for the **–A** command are equivalent to the values **mail** *user-name* and **write** *user-name*, respectively, where *user-name* is the current name for the administrator. This will be the login name of the person submitting this command *unless* he or she has used the **su** command to change to another user ID. If the **su** command has been used to change the user ID, then the *user-name* for the new ID is used.

If the **–Q** option is not given or *integer*$_1$ is one or the word **any** (which are both the default), a message is sent as soon as anyone submits a print request for the form when it is not mounted.

If the **–W** option is not given or *integer*$_2$ is zero or the word **once** (which are both the default), only one message is sent when the queue size exceeds *integer*$_1$. If *integer*$_2$ is a non-zero number, an alert will be sent every *integer*$_2$ minute(s).

Listing the Current Alert

The third **lpforms** command line (shown under **SYNOPSIS**) is used to list the type of the alert for the specified form. No change is made to the alert. If *form-name* is recognized by the LP print service, one of the following lines is sent to the standard output, depending on the type of alert for the form.

- When *integer*$_1$ are queued: alert with "*shell-command*" every *integer*$_2$ minutes
- When *integer*$_1$ are queued: write to *user-name* every *integer*$_2$ minutes
- When *integer*$_1$ are queued: mail to *user-name* every *integer*$_2$ minutes
- No alert

The phrase every *integer* minutes is replaced with once if *integer*$_2$ (**–W** *integer*$_2$) is 0.

Terminating an Active Alert

The **–A quiet** option is used to stop messages for the current condition. An

administrator can use this option to temporarily stop receiving further messages about a known problem. Once the form has been mounted and then unmounted, messages will again be sent when the queue size reaches *integer*$_1$ pending requests.

Removing an Alert Definition

No messages will be sent after the **-A none** option is used until the **-A** option is given again with a different *alert-type*. This can be used to permanently stop further messages from being sent, as any existing alert definition for the form will be removed.

SEE ALSO

lp(1), lpadmin(1M), terminfo(4)

NAME

lpsched, lpshut, lpmove – start/stop the LP print service and move requests

SYNOPSIS

/usr/lib/lpsched
/usr/lib/lpshut
/usr/lib/lpmove *requests dest*
/usr/lib/lpmove *dest*$_1$ *dest*$_2$

DESCRIPTION

lpsched starts the LP print service; this can be done only by **root** or **lp**.

lpshut shuts down the print service. All printers that are printing at the time **lpshut** is invoked will stop printing. When **lpsched** is started again, requests that were printing at the time a printer was shut down will be reprinted from the beginning.

lpmove moves requests that were queued by **lp**(1) between LP destinations. The first form of the **lpmove** command shown above (under **SYNOPSIS**) moves the named *requests* to the LP destination *dest*. *Requests* are request-ids as returned by **lp**(1). The second form of the **lpmove** command moves all requests for destination *dest*$_1$ to destination *dest*$_2$; **lp**(1) will then reject any new requests for *dest*$_1$.

Note that when moving requests, **lpmove** never checks the acceptance status (see **accept**(1M)) of the new destination. Also, the request-ids of the moved request are not changed, so that users can still find their requests. The **lpmove** command will not move requests that have options (content type, form required, and so on) that cannot be handled by the new destination.

If a request was originally queued for a class or the special destination **any**, its destination will be changed to *new-destination*. A request thus affected will be printable only on *new-destination* and not on other members of the **class** or other acceptable printers if the original destination was **any**.

FILES

/usr/spool/lp/*

SEE ALSO

accept(1M), enable(1), lp(1), lpadmin(1M), lpstat(1)

NOTE

By default, the directory */usr/spool/lp* is used to hold all the files used by the LP print service. This can be changed by setting the **SPOOLDIR** environment variable to another directory before running **lpsched**. If you do this, you should populate the directory with the same files and directories found under */usr/spool/lp*; the LP print service will not automatically create them. Also, the **SPOOLDIR** variable must then be set before any of the other LP print service commands are run.

NAME

lpstat – print information about the status of the LP print service

SYNOPSIS

lpstat [options]

DESCRIPTION

lpstat prints information about the current status of the LP print service.

If no options are given, **lpstat** prints the status of all requests made to **lp**(1) by users. Any arguments that are not options are assumed to be *request-ids* (as returned by **lp**), printers, or printer classes. **lpstat** prints the status of such requests, printers, or printer classes. 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, as shown in the following example.

–u "user1, user2, user3"

Specifying **all** after a keyletter that takes *list* as an argument causes all information relevant to the keyletter to print. For example, the following command prints the status of all output requests.

lpstat –o all

The options are as follows:

–a [*list*] Display acceptance status (with respect to **lp**) of destinations for requests (see **accept**(1M)). *list* is a list of intermixed printer names and class names; the default is **all**.

–c [*list*] Display class names and their members. *list* is a list of class names; the default is **all**.

–d Display the system default destination for **lp**.

–f [*list*] [**–l**] Display a verification that the forms in *list* are recognized by the LP print service. *list* is a list of forms; the default is **all**. The **–l** option will list the form descriptions.

–o [*list*] [**–l**] Display the status of output requests. *list* is a list of intermixed printer names, class names, and request-ids; the default is **all**. The **–l** option gives a more detailed status of the request.

–p [*list*] [**–D**] [**–l**]
Display the status of printers named in *list*. *list* is a list of printers; the default is **all**. If the **–D** option is given, a brief description is printed for each printer in *list*. If the **–l** option is given, a full description of each printer's configuration is given, including the form mounted, the acceptable content and printer types, a printer description, the interface used, and so on.

–r Display the status of the LP request scheduler.

–s Display a status summary, including the system default destination, a list of class names and their members, a list of printers and their associated devices, a list of all forms currently mounted, and a list of all recognized character sets and print

wheels.

–S [*list*] [**–l**] Display a verification that the character sets or print wheels specified in *list* are recognized by the LP print service. Items in *list* can be character sets or print wheels; the default for the list is **all**. If the **–l** option is given, each line is appended by a list of printers that can handle the print wheel or character set. The list also shows whether the print wheel or character set is mounted or specifies the built-in character set into which it maps.

–t Display all status information.

–u [*list*] Display the status of output requests for users. *list* is a list of login names; the default is **all**.

–v [*list*] Display the names of printers and the pathnames of the devices associated with them. *list* is a list of printer names; the default is **all**.

FILES

/usr/spool/lp/*

SEE ALSO

enable(1), lp(1)

NAME

lpusers – set printing queue priorities

SYNOPSIS

/usr/lib/lpusers –d *priority-level*
/usr/lib/lpusers –q *priority-level* **–u** *user-list*
/usr/lib/lpusers –u *user-list*
/usr/lib/lpusers –q *priority-level*
/usr/lib/lpusers –l

DESCRIPTION

The **lpusers** command is used to set limits to the queue priority level that can be assigned to jobs submitted by users of the LP print service.

The first form of the command (with **–d**) sets the system-wide priority default to *priority-level*, where *priority-level* is a value of 0 to 39, with 0 being the highest priority. If a user does not specify a priority level for a print request (see **lp**(1)), the default priority is used. Initially, the default priority level is 20.

The second form of the command (with **–q** and **–u**) sets the default highest *priority-level* (0-39) that the users in *user-list* can request when submitting a print request. Users that have been given a limit cannot submit a print request with a higher priority level than the one assigned, nor can they change a request already submitted to have a higher priority. Any print requests with priority levels higher than allowed will be given the highest priority allowed.

The third form of the command (with **–u**) removes the users from any explicit priority level and returns them to the default priority level.

The fourth form of the command (with **–q**) sets the default highest priority level for all users not explicitly covered by the use of the second form of this command.

The last form of the command (with **–l**) lists the default priority level and the priority limits assigned to users.

SEE ALSO

lp(1)

NAME

ls – list contents of directory

SYNOPSIS

ls [**–RadCxmlnogrtucpFbqisf**] [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 **ioctl**(2) call is used to see if **cols** is set and uses this as the column width. Otherwise, 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 (.), that 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. If the file is a symbolic link, the filename is printed followed by a "->" and the pathname of the referenced file.

–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 inode (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, and put an @ if that file is a symbolic link.

–b Force printing of non-printable characters to be in the octal **\ddd** notation.

–q Force printing of non-printable characters in file names as the character question mark (**?**).

–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 10 characters. The first character can be one of the following:

d The entry is a directory
l The entry is a symbolic link
b The entry is a block special file
c The entry is a character special file
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 second set refers to permissions for the owner's group, and the last set refers 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 means 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
```

This horizontal configuration provides a good deal of information. Reading from right to left, 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 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 is off)
- **t** The 1000 (octal) bit, or sticky bit, is on (see **chmod**(1)), and execution is on
- **T** 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, **l** may occupy the third position. **l** 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 following example describes a file that is readable, writable, and executable by the user and readable by the group and others:

```
-rwxr--r--
```

The following example describes a file that is readable, writable, and executable by the user and readable and executable by the group and others. It also allows its user-ID to be assumed, during execution, by the user presently executing it.

```
-rwsr-xr-x
```

The next example describes a file that is readable and writable only by the user and the group and that can be locked during access.

```
-rw-rwl---
```

The following command line will print the names of all files in the current directory, including those that begin with a dot (.), which normally do not print.

ls –a

The next command line provides you with quite a bit of information including non-printing files (**a**), the **i**-number—the memory address of the i-node associated with the file—printed in the left-hand column (**i**), and 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 **n**umeric version of the long list, printing the UID (instead of user name) and GID (instead of group name) numbers associated with the files.

ls –aisn

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks, is printed.

FILES

/etc/passwd	User IDs for **ls –l** and **ls –o**
/etc/group	Group IDs for **ls –l** and **ls –g**
/usr/lib/terminfo/?/*	Terminal information database

SEE ALSO

chmod(1), find(1)

BUGS

Unprintable characters in filenames may confuse the columnar output options.