Reference Manual

REAL/IX® Operating System Sections 1, 1M, and 1R

Open Architecture Systems

211-863002-002

AEG

MODCOMP

Property of Logical Data Corporation RECEIVED APR 2 8 1993

AEG

Reference Manual

REAL/IX® Operating System Sections 1, 1M, and 1R

Open Architecture Systems

MODCOMP

CLASSIC, CLASSIC REAL/IX, MAX, MODCOMP, MODMate, GLS, PACE/32, PACE/IX, PRIDE, REAL/IX, REAL/STONE, REAL/VU, Tri-Dimensional, Tri-D, TRI-SIM, and Modular Computer Systems are registered trademarks of Modular Computer Systems, Inc. Development Server, REAL/MAX, and REAL/STAR are trademarks of Modular Computer Systems, Inc.

The following are trademarks or **registered** trademarks of their respective companies: Macintosh of Apple Computer, Inc.; **Documenter's Workbench, TELETYPE** of AT&T; VCL, EDT+ of Boston Business Computing, Ltd.; Belden of Cooper Industries, Inc.; Centronics of Data Computer Corporation; DEC, DECnet, PDP, VAX, VMS of Digital Equipment Corporation; **EXOS, Excelan** of Excelan, Inc.; HP, Vectra of Hewlett-Packard Company; VRTX/68000 of Hunter & Ready, Inc.; BITBUS, Intel-(followed by any number) of Intel Corporation; **IBM**, PC-DOS, **PC/AT, PC/XT, PS/2** of International Business Machines Corporation; X Window System of Massachusetts Institute of Technology; **Microsoft, Windows, MS-DOS, Xenix** of Microsoft Corporation; MODICON, Modbus, Modbus Plus of Modicon, Inc.; **Motorola**, System V/68, VME Delta Series, Delta Series, VMEexec, Business Assistant, OfficeLAN, VERSAbus, VERSAdos, 133Bug, 147BUG, 187Bug, HYPERmodule, VMEmodule, 188Bug, SSID of Motorola, Inc.; OSF, OSF/Motif, Motif of Open Software Foundation, Inc.; **ORACLE, SQL*Calc, SQL*Forms, SQL*Plus**, SQL*Graph, **SQL*Menu, SQL*Net**, Pro*SQL, Pro*ORACLE, Pro*C, Pro*COBOL, Easy*SQL of Oracle Corporation; Q-ONE of Quadretron Systems Inc.; FORTRIX of Rapitech Systems, Inc.; **PROBUG** of SBE, Inc.; **Sun**, NFS, ONC of Sun Microsystems, Inc.; **TEKTRONIX** of TEKTRONIX, Inc.; **TeleSoft**, Telegen2 of **TeleSoft; TeleVideo** of TeleVideo Systems, Inc.; **UNIX** of UNIX System Laboratories, Inc.; **DIABLO, Ethernet, Xerox** of Xerox Corporation; **88open**, BCS, OCS of 88open Consortium, Ltd..

Copyright © 1991, 1992, 1993 by Modular Computer Systems, Inc.
Printed in the United States of America. All rights reserved.

Contents subject to change without notice.

Manual History

211–863002

REAL/IX Operating System Sections 1, 1M, and 1R, Reference Manual

Revisions are listed in reverse chronological order. Detailed summaries are provided for the five most recent revisions.

Revision 002 (Reissue) 03/93

Incorporated the change package pages of Revision 001 into the base manual.

Revision 001 (Change Package) 08/92

The front matter has been updated to current standards.

The following reference manual pages for Section 1 are new or modified for this change package:

- as2(1)
- at(1)
- bru(1)
- cc(1)
- chown(1)
- conman(1)
- cpio(1)
- cputype(1)
- dbx(1)
- dnp(1)
- domainname(1N)
- egrep(1)
- ex(1)
- fdivfix(1)
- find(1)
- ftp(1N)
- ld(1)
- login(1)
- lp(1)
- ls(1)
- machid(1)
- mail(1)
- make(1)
- makekey(1)
- man(1)
- mformat(1)
- mkshlib(1)
- mt(1)
- pack(1)
- putbuf(1)
- rpcgen(1N)
- rtgroups(1)
- rtusers(1)
- ruptime(1N)
- sh(1)
- strip(1)
- stty(1)
- systat(1)
- tar(1)
- vi(1)
- xargs(1)
- ypcat(1N)
- ypmatch(1N)
- yppasswd(1N)
- ypwhich(1N)

The following reference manual pages for Sections 1M and 1R are new or modified for this change package:

- arp(1M)
- automount(1M)
- bootpd(1M)
- clri(1M)
- cpset(1M)
- crash(1M)
- dcopy(1M)
- ddefs(1M)
- dinit(1M)
- fsck(1M)
- fsinfo(1M)
- getty(1M)
- halt(1M)
- id(1M)
- ifconfig(1M)
- ifctl(1M)
- ifprobe(1M)
- ifstat(1M)
- igf(1M)
- inetd(1M)
- intro(1M)
- kdb(1M)
- lpadmin(1M)
- lpsched(1M)
- makedbm(1M)
- map2ascii(1M)
- mbs(1M)
- mkfs(1M)
- mount(1M)
- mountall(1M)
- netlp(1M)
- netlpd(1M)
- newgrp(1M)
- nfs(1M)
- nfsclnt(1M)
- nfsstat(1M)
- nlsadmin(1M)
- nmountall(1M)
- passmgmt(1M)
- portconfig(1M)
- profiler(1M)
- ptyfix(1M)
- pwck(1M)
- rexd(1M)
- rlogind(1M)
- route(1M)
- rpcinfo(1M)
- savecore(1M)
- savedump(1M)
- sckconf(1M)
- scsiconf(1M)
- scsiload(1M)
- setrtgroups(1M)
- setrtusers(1M)
- setup(1M)
- sledit(1M)
- su(1M)
- swap(1M)
- tapectl(1M)
- telnetd(1M)
- tftp(1M)
- tpid(1M)
- uucico(1M)
- ypmake(1M)
- yppassmgt(1M)
- yppasswdd(1M)
- yppush(1M)
- ypserv(1M)
- ypset(1M)
- prealloc(1R)
- resume(1R)
- setpri(1R)

Revision 000 (Initial Issue) 06/91

About This Manual

This is one of the three volumes that constitute the *REAL/IX Reference Manual Set*. This manual contains the following sections:

- **Section 1, Commands**, describes the commands available to all users, including all options and sometimes examples.
- **Section 1M, Administrative Commands**, describes administrative commands, most of which can be executed only by the superuser.
- **Section 1R, Realtime Commands**, describes the commands that can be executed by the superuser or a user with realtime privileges.

The pages in each section are arranged alphabetically. Use the permuted index when you know the topic but not the manual page name. The permuted index is alphabetized on the middle column, with one line for every significant word in the **NAME** field on each manual page. For ease of reference, each volume of the *REAL/IX Reference Manual Set* contains the permuted index for the entire set. All manual pages can be accessed online through the **man**(1) command.

The term "open architecture system", in its simplest form, implies that a user may add a variety of vendors' components to a single system. This is possible when certain industry-accepted standards have been implemented in the system. MODCOMP open architecture systems are based on such software and hardware standards as the UNIX System V operating system, VMEbus and SCSI bus interfaces, and CPUs built around standard microprocessors. By building on these standards, open architecture systems provide computer solutions that are portable and compatible.

The REAL/IX Operating System[1], which runs on all MODCOMP open architecture system hardware platforms, allows applications to be ported easily between traditional UNIX systems and MODCOMP open architecture systems. Furthermore, by using VMEbus and SCSI bus interfaces, MODCOMP open architecture systems ensure compatibility among a wide range of peripheral and I/O devices and the ability to expand as needs dictate. MODCOMP open architecture systems meet networking and communications needs with such industry standards as Ethernet and TCP/IP and have the flexibility to accommodate new standards as they are developed.

[1]The REAL/IX Operating System, featuring realtime and multiprocessing capabilities, is the MODCOMP implementation of the UNIX System Laboratories UNIX System V operating system.

Related Publications

Refer to the following manuals for additional information. Contact your MODCOMP Sales Representative to order.

Books for All System Users

Concepts and Characteristics
Gives an overview of the internals of the REAL/IX Operating System and an introduction to the tools and facilities that are available.

POSIX Conformance Guide
Describes conformance to IEEE Std 1003.1–1988. This document describes only those areas where the specification allows implementation-defined behavior, or where the behavior of an implementation may vary.

Reference Manual, Sections 1, 1M, and 1R
Contains manual pages for user commands (Section 1), administrative commands (Section 1M), and realtime commands (Section 1R).

Reference Manual, Sections 2, 3, and 5
Contains manual pages for system calls (Section 2), library routines (Sections 3C, 3M, 3N, 3S, and 3X), and miscellaneous information (Section 5).

Reference Manual, Sections 4, 7, and 7A
Contains manual pages for system files (Section 4), special device files for standard devices (Section 7), and special device files for add-on packages (Section 7A).

User's Guide
Discusses basic user procedures including the login procedure and getting around the file system. Information is included about general user tools; for example, the **vi** and **ed** text editors, electronic mail, the shell programming language, and the Korn shell.

Using UUCP and Usenet
Introduces UUCP communications, describes how to transfer files and execute remote commands over UUCP, how to check on UUCP requests, and how to access the Usenet electronic bulletin board.

Books for System Administrators

System Guide
Gives an overview of MODCOMP open architecture systems VMEbus-based computers and contains instructions for the installation and maintenance of these systems.

Guide to VME Modules, MVME188 RISC Board Set

Provides installation, hardware setup information, and firmware-level initialization information for VMEbus-based systems.

Software Installation Guide

Gives instructions for installing the operating system (either for the first time or as an upgrade) and initially setting up the system.

System Administrator's Guide

Gives instructions and background information about administering the REAL/IX Operating System. Topics covered include ensuring system security; creating and maintaining user and group IDs; working with file systems (creating, repairing, backing up); setting up terminals and printers; using the **sysgen**(1M) utility to modify tunable parameters and to configure or deconfigure standard system devices; and setting up and using the Job Accounting System. Appendixes discuss the system files that control system operations and the file naming conventions for special device files.

Open Network Computing for Open Architecture Systems User's Guide

Discusses Open Network Computing (ONC) as it applies to the REAL/IX Operating System, including procedures and instructions for installing, configuring, removing, and managing the Network File System (NFS) and Network Information Services (NIS). General programming guidelines are provided for the eXternal Data Representation (XDR) and Remote Procedure Call (RPC) facilities of ONC.

Trouble Analysis Guide

Gives guidelines for avoiding system problems and lists some common system problems with suggestions for solving them.

Software Engineering Release Notes

Gives an overview of the new features in this release of the REAL/IX Operating System and provides usage notes for the system.

Managing UUCP and Usenet

Provides background information about UUCP for administrators and gives instructions for setting up a UUCP link, verifying that the link works, administering UUCP communications, and setting up and administering the Usenet access. This information is supplemented by the *System Administrator's Guide*, which includes information for administering UUCP over the TCP/IP protocol, and the *Software Installation Guide*.

Books for Programmers

Languages and Support Tools Guide

Provides tutorials for many of the special purpose languages and the programming support tools available on the REAL/IX Operating System.

Languages and Support Tools Guide Supplement
Contains information that is specific to the released system as it operates in the native microprocessor environment of your hardware platform. Note that the *"x"* in the manual number represents a number specific to the supplement shipped with your system.

Programmer's Guide
Gives an overview of the REAL/IX Operating System and realtime computing, describes the REAL/IX programming environment and the operating system interface, and provides programming examples for using the realtime extensions of the REAL/IX Operating System as well as the standard UNIX operating system features.

GLS Programming Guide Host and Cross Development Environments
Describes how to install and execute each GLS compiler (C, FORTRAN, Pascal) in the host and cross environments.

The C Programming Language, First Edition
Describes the traditional UNIX C language compatible with the GLS C compiler.

The C Programming Language, Second Edition
Describes the ANSI C language compatible with the GLS C compiler.

Books for Kernel Programmers

Driver Development Guide
Introduces the process of writing device drivers for the REAL/IX Operating System, including detailed information about porting and installing drivers.

Driver Development Guide Supplement
Contains information that is specific to the released system as it operates in the native microprocessor environment of your hardware platform. Note that the *"x"* in the manual number represents a number specific to the supplement shipped with your system.

Kernel Programming Guide
Gives background information about topics of interest to programmers writing device drivers and system calls. Topics discussed include how drivers and system calls execute and how various types of I/O operations are implemented.

Kernel Programming Guide Supplement
Contains information that is specific to the released system as it operates in the native microprocessor environment of your hardware platform. Note that the *"x"* in the manual number represents a number specific to the supplement shipped with your system.

Kernel Reference Manual
Contains reference pages for driver entry-point routines (Section D2X), kernel functions and macros (Section D3X), and kernel data structures (Section D4X) used for coding system calls and device drivers.

Industry Standard Publications

The REAL/IX Operating System and its supported C programming language comply with the industry standards listed below. These standards are commercially available and can be obtained from the following sources. While an effort was made to ensure that the ordering information was complete and up-to-date at time of printing, we cannot guarantee its accuracy.

ANSI X3.159–1989 *Programming Language C Standard*
American National Standards Institute, Inc.
Sales Department
1430 Broadway
New York, NY 10018
Phone: (212) 642–4900
Fax: (212) 302–1286

IEEE Std 1003.1–1988 *Standard Portable Operating System Interface for Computer Environments (POSIX)*
The Institute of Electrical and Electronics Engineers, Inc.
Publications Sales, IEEE Service Center
P.O. Box 1331
445 Hoes Lane
Piscataway, NJ 08855–1331
Phone: 1–800–678–IEEE
Fax: (201) 981–9677

88open Binary Compatibility Standard (BCS)
88open Consortium Ltd.
Marketing Department
100 Homeland Court, Suite 800
San Jose, CA 95112
Phone: (408) 436–6600
Fax: (408) 436–0725

Service and Assistance

MODCOMP offers a variety of programs and services that demonstrate our commitment to customer satisfaction. Our Technical Education department provides comprehensive hands-on instruction either at our facilities or at customer-designated sites. Our worldwide field service organization is ready to give installation assistance, free service during the warranty period, and flexible service programs tailored to your requirements.

Your MODCOMP sales and service representatives can help you with any questions, problems, or suggestions you may have for our products and services. For your convenience, MODCOMP maintains toll-free telephone numbers at which we can be reached. Numbers you may find helpful are listed here.

For	Call	From
Questions, sales information, or suggestions	• 1-800-255-2066	• U.S.A. and Canada
	• 1-305-974-1380 extension 1800	• outside the U.S.A. and Canada
	or please call your regional support office	
Service	• 1-800-327-8928	• U.S.A.
	• 1-416-890-0666	• Canada
	Outside the U.S.A. and Canada, please call your regional service/support office.	
Technical Education Information	• 1-305-977-1708	• U.S.A.
	Outside the U.S.A., please call your regional support office.	

For comments about documentation, please use the response form at the back of this manual.

RECEIVED APR 2 8 1993

PERMUTED INDEX

canonical "server" process for testing network ..... sink(1)
names of hosts with equivalent users, i.e., "trusted" hosts ..... hosts.equiv(4N)
"trusted" hosts names of hosts with equivalent users, i.e., ... hosts.equiv(4N)
canonical server" process ..... for/"
read (copy) a MSDOS file to Unix ..... mread(1)
write (copy) a Unix file to MSDOS ..... mwrite(1)
generate cyclic redundancy checksums (crc) of files ..... crc(1)
file for framed access command environment (FACE) utilities alias ..... alias(4)
send and receive messages (message operations) ..... msgop(2)
text editor (variant of ex for casual users) ..... edit(1)
screen-oriented (visual) display editor based on ex ..... vi(1)
GNU project parser generator (yacc replacement) ..... bison(1)
and update /etc/shadow with information from /etc/passwd install ..... pwconv(1M)
install and update /etc/shadow with information from /etc/passwd pwconv(1M)
convert between 3-byte integers and long integers ..... l3tol(3C)
3-way differential file comparison ..... diff3(1)
handle special functions of/ 300 ..... 300(1)
handle special functions of DASI 300 and 300s terminals ..... 300(1)
handle special functions of/ 300s ..... 300(1)
handle special functions of DASI 300 and 300s terminals ..... 300(1)
paginator for the Tektronix/ 4014 ..... 4014(1)
paginator for the Tektronix 4014 terminal ..... 4014(1)
driver for the MODCOMP 4211 Peregrine FDDI Controller ..... fddi4211(7)
handle special functions of the/ 450 ..... 450(1)
handle special functions of the DASI 450 terminal ..... 450(1)
driver for the MVME355 9-Track Tape Controller ..... mvme355(7)
REAL/IX realtime SCSI/ general 9-track tape implementation details used by .. scsi9track(7)
remake the binary system and/ :mk ..... :mk(1M)
common assembler and link/ a.out ..... a.out(4)
merge an a.out file into another a.out file. ..... scsiload(1M)
merge an a.out file into another a.out file. ..... scsiload(1M)
convert between long integer/ a64l ..... a64l(3C)
generate an IOT fault abort ..... abort(3C)
return integer absolute value abs ..... abs(3C)
set the expiration time of a/ absinterval ..... absinterval(2)
return integer absolute value ..... abs(3C)
floor, ceiling, remainder, absolute value functions ..... floor(3M)
cancel one or more asynchronous/ acancel ..... acancel(2)
accept a connection on a socket accept ..... accept(2N)
allow or prevent LP requests accept ..... accept(1M)
accept a connect request ..... t_accept(3N)
accept a connection on a socket ..... accept(2N)
determine accessibility of a/ access ..... access(2)
set file access and modification times ..... utime(2)
update access and modification times of a file ..... touch(1)
alias file for framed access command environment (FACE) utilities alias(4)
system-wide framed access command environment variables ..... env(4)
get group access list ..... getgroups(3P)
initialize group access list ..... initgroups(3C)
set group access list ..... setgroups(2)
machine-independent fashion access long integer data in a ..... sputl(3X)
common object file access routines ..... ldfcn(4)
copy file systems for optimal access time ..... dcopy(1M)

permissions file used by the value-added disk	access utilities	filesys(4)
	access utmp file entry	getut(3C)
determine	accessibility of a file	access(2)
connect-time	accounting	acctcon(1M)
enable or disable process	accounting	acct(2)
process	accounting	acctprc(1M)
shell procedures for	accounting	acctsh(1M)
commands overview of	accounting and miscellaneous accounting	acct(1M)
overview of accounting and miscellaneous	accounting commands	acct(1M)
generate disk	accounting data by user ID	diskusg(1M)
per-process	accounting file format	acct(4)
search and print process	accounting file(s)	acctcom(1)
merge or add total	accounting files	acctmerg(1M)
command summary from per-process	accounting records	acctcms(1M)
manipulate connect	accounting records	fwtmp(1M)
enable or disable process/	acct	acct(2)
per-process accounting file/	acct	acct(4)
command summary from/	acctcms	acctcms(1M)
search and print process/	acctcom	acctcom(1)
connect-time accounting	acctcon1	acctcon(1M)
connect-time accounting	acctcon2	acctcon(1M)
overview of accounting and/	acctdisk	acct(1M)
overview of accounting and/	acctdusg	acct(1M)
merge or add total accounting/	acctmerg	acctmerg(1M)
overview of accounting and/	accton	acct(1M)
process accounting	acctprc1	acctprc(1M)
process accounting	acctprc2	acctprc(1M)
overview of accounting and/	acctwtmp	acct(1M)
indication	acknowledge receipt of an orderly release	t_rcvrel(3N)
trigonometric functions	acos	trig(3M)
examine or change signal	action	sigaction(2P)
kill all	active processes	killall(1M)
print current SCCS file editing	activity	sact(1)
ime a command; report process data and system	activity	timex(1)
system	activity report package	sar(1M)
system	activity reporter	sar(1)
merge or	add total accounting files	acctmerg(1M)
change or	add value to environment	putenv(3C)
Internet	address manipulation routines	inet(3N)
	Address Resolution Protocol	arp(7N)
	address resolution protocol maintenance	arp(1M)
bind an	address to a transport endpoint	t_bind(3N)
unmap previously mapped	addresses	munmap(2)
map process	addresses to a file	mmap(2)
correct the time to allow/	adjtime	adjtime(2)
create and administer SCCS files	admin	admin(1)
service	administer filters used with the LP print	lpfilter(1M)
	administer forms used with the LP print service	lpforms(1M)
create and	administer SCCS files	admin(1)
devices	administered by System Administration	sa(7)
devices administered by System	Administration	sa(7)
menu interface to do system	administration	sysadm(1M)
network listener service	administration	nlsadmin(1M)
	administrative control	uadmin(1M)
	administrative control	uadmin(2)
swap	administrative interface	swap(1M)
control block for asynchronous/	aiocb	aiocb(4)

set a process alarm clock	alarm	alarm(2)
set a process	alarm clock	alarm(2)
real device in the permissions file for a given	alias echo the	real(1)
environment (FACE) utilities	alias file for framed access command	alias(4)
	allocate a library structure	t_alloc(3N)
change data segment space	allocation	brk(2)
fast main memory	allocator	malloc(3C)
main memory	allocator	malloc(3X)
	allow or prevent LP requests	accept(1M)
correct the time to	allow synchronization of the system clock	adjtime(2)
displays or	alters byte content of files	bpatch(1)
file system block	analyzer	fsba(1M)
get yes/no response from user or check	answer to question	chkyn(1)
introduction to commands and	application programs	intro(1)
introduction to commands and	application programs	intro(1N)
introduction to maintenance commands and	application programs	intro(1M)
archive and library maintainer/	ar	ar(1)
common archive file format	ar	ar(4)
	arbitrary-precision arithmetic language	bc(1)
files that hold object	architecture information	ott(4)
a brief overview of the software	architecture of the Quadbus multiprocessor	realstar(5)
format of cpio	archive	cpio(4)
archives	archive and library maintainer for portable	ar(1)
read the archive header of a member of an	archive file	ldahread(3X)
common	archive file format	ar(4)
convert	archive files to common formats	convert(1)
read the	archive header of a member of an archive file	ldahread(3X)
tape file	archiver	tar(1)
archive and library maintainer for portable	archives	ar(1)
copy file	archives in and out	cpio(1)
read from file in an/	aread	aread(2)
handle variable	argument list	varargs(5)
print formatted output of a varargs	argument list	vprintf(3S)
construct	argument list(s) and execute command	xargs(1)
get option letter from	argument vector	getopt(3C)
echo	arguments	echo(1)
evaluate	arguments as an expression	expr(1)
initialize structures before/	arinit	arinit(2)
arbitrary-precision	arithmetic language	bc(1)
Address Resolution Protocol	arp	arp(7N)
address resolution protocol/	arp	arp(1M)
	ARPANET file transfer program	ftp(1N)
be notified if mail	arrives and who it is from	biff(1)
free internal resources for/	arwfree	arwfree(2)
assembler driver script	as	as(1)
assembler	as	as(1)
interpret ASA carriage control/	asa	asa(1)
interpret	ASA carriage control characters	asa(1)
convert date and time to string	ascftime	ctime(3C)
map of ASCII character set	ascii	ascii(5)
strings for converting times and dates to	ASCII language specific	cftime(4)
map of	ASCII character set	ascii(5)
convert between long integer and base-64	ASCII string	a64l(3C)
convert date and time to string	asctime	ctime(3C)
trigonometric functions	asin	trig(3M)
	assembler	as(1)
common	assembler and link editor output	a.out(4)

	assembler driver script	as(1)
preprocess MC88100	assembly language	sifilter(1)
verify program assertion	assert	assert(3X)
verify program	assertion	assert(3X)
	assign buffering to a stream	setbuf(3S)
assistance using REAL/IX/	assist	assist(1)
program for generating/modifying	ASSIST menus or command forms	astgen(1)
	assistance using REAL/IX commands	assist(1)
program for/	astgen	astgen(1)
log incoming SSP maintenance bus	async messages.	mblogger(1M)
print out incoming SSP maintenance bus	async messages.	mba(1M)
free internal resources for	asynchronous I/O from the process	arwfree(2)
control block for	asynchronous I/O operations	aiocb(4)
cancel one or more	asynchronous I/O requests	acancel(2)
read from file in an	asynchronous manner	aread(2)
write to file in an	asynchronous manner	awrite(2)
initialize structures before requesting an	asynchronous read	arinit(2)
initialize structures before requesting an	asynchronous write	awinit(2)
execute commands at a later time	at	at(1)
trigonometric functions	atan	trig(3M)
trigonometric functions	atan2	trig(3M)
convert string to/	atof	strtod(3C)
convert string to integer	atoi	strtol(3C)
convert string to integer	atol	strtol(3C)
log of failed login	attempts	loginlog(4)
change login password and password	attributes	passwd(1)
control write/execute	attributes of memory	memctl(2)
	autoconfig MVME187 SCSI controller interface	mvme187(7)
	autoconfig MVME327 SCSI controller interface	mvme327(7)
details used by REAL/IX realtime SCSI	autoconfiguration drivers /tape implementation	scsi9track(7)
details used by REAL/IX realtime SCSI	autoconfiguration drivers /tape implementation	scsitape(7)
	automatically mount NFS file systems	automount(1M)
automatically mount NFS file/	automount	automount(1M)
patch XC88100 COFF executables to	avoid fdiv deadlock	fdivfix(1)
	await completion of process	wait(1)
initialize structures before/	awinit	awinit(2)
pattern scanning and processing/	awk	awk(1)
write to file in an/	awrite	awrite(2)
fast incremental	backup	finc(1M)
	backup and restore utility	bru(1)
check file system	backup schedule	ckbupscd(1M)
recover files from a	backup tape	frec(1M)
convert	bad track list from one format to another	xformtrk(1M)
make posters	banner	banner(1)
host name data	base	hosts(4N)
protocol name data	base	protocols(4N)
rpc program number data	base	rpc(4N)
service name data	base	services(4N)
terminal capability data	base	terminfo(4)
convert between long integer and	base-64 ASCII string	a64l(3C)
deliver portions of pathnames	basename	basename(1)
execute commands at a later time	batch	at(1)
get or set the value of the output and input	baud rate	cfgetospeed(3P)
arbitrary-precision arithmetic/	bc	bc(1)
system initialization procedures	bcheckrc	bcheckrc(1M)
big diff	bdiff	bdiff(1)
C program	beautifier	cb(1)

	become super-user or another user	su(1M)
information about the system for	beginning users	starter(1)
	Bessel functions	bessel(3M)
summarize cache hit rate for/	bfree	bfree(1M)
big file scanner	bfs	bfs(1)
be notified if mail arrives and/	biff	biff(1)
	biff server	comsat(1M)
	big diff	bdiff(1)
	big file scanner	bfs(1)
force compliance with a	bill of materials	mkcomply(1M)
install object files in	binary directories	cpset(1M)
	binary input/output	fread(3S)
	binary search a sorted table	bsearch(3C)
manage	binary search trees	tsearch(3C)
conditionally lock a	binary semaphore	bslkc(3C)
free a	binary semaphore	bsfree(2)
get a	binary semaphore	bsget(2)
lock a	binary semaphore	bslk(3C)
unlock a	binary semaphore	bsunlk(3C)
remake the	binary system and commands from source code	:mk(1M)
bind a name to a socket	bind	bind(2N)
	bind a name to a socket	bind(2N)
	bind a socket to a privileged IP port	bindresvport(3N)
	bind an address to a transport endpoint	t_bind(3N)
NIS server and	binder processes	ypserv(1M)
bind a socket to a privileged/	bindresvport	bindresvport(3N)
NFS daemons	biod	nfsd(1M)
GNU project parser generator/	bison	bison(1)
update super	block	sync(2)
update the super	block	sync(1M)
file system	block analyzer	fsba(1M)
print checksum and	block count of a file	sum(1)
control	block for asynchronous I/O operations	aiocb(4)
convert track numbers into file system logical	block numbers	fblkgen(1M)
examine and change	blocked signals	sigprocmask(2P)
report number of free disk	blocks and i-nodes	df(1M)
bootstrap loader for QWB/	bo	bo(1M)
Environment Monitor	Board driver	emb(7)
print CPU	board information	cputype(1)
Network Interface Driver for MVME187	Boards	ifen596(7N)
get	boot inode	gbootinod(1M)
Internet	Boot Protocol server	()
Internet Boot Protocol server	bootpd	
	bootstrap loader for QWB multiprocessor	bo(1M)
the	bourne shell command programming language	bsh(1)
displays or alters byte content/	bpatch	bpatch(1)
back up and restore files/	br	br(1M)
the Quadbus multiprocessor a	brief overview of the software architecture of	realstar(5)
change data segment space/	brk	brk(2)
backup and restore utility	bru	bru(1)
binary search a sorted table	bsearch	bsearch(3C)
free a binary semaphore	bsfree	bsfree(2)
get a binary semaphore	bsget	bsget(2)
the bourne shell command/	bsh	bsh(1)
lock a binary semaphore	bslk	bslk(3C)
conditionally lock a binary/	bslkc	bslkc(3C)
unlock a binary semaphore	bsunlk	bsunlk(3C)

summarize cache hit rate for each — buffer size ..... bfree(1M)
standard — buffered input/output package ..... stdio(3S)
assign — buffering to a stream ..... setbuf(3S)
with **NULL** pointer dereference — bug patch program ..... dnp(1)
— build and install NIS database ..... ypinit(1M)
— build special file ..... mknod(1M)
a simple loopback test on the SSP maintenance — bus perform ..... mbloop(1M)
one sync transfer on the SSP maintenance — bus perform ..... mbs(1M)
log incoming SSP maintenance — bus async messages ..... mblogger(1M)
print out incoming SSP maintenance — bus async messages ..... mba(1M)
SSP maintenance — bus interface ..... sspmb(7)
turn SSP maintenance — bus polling on or off ..... mbpoll(1M)
displays or alters — byte content of files ..... bpatch(1)
convert values between host and network — byte order ..... byteorder(3N)
swap — bytes ..... swab(3C)
examine and change — bytes in a file ..... patch(1M)
print section sizes in — bytes of common object files ..... size(1)
— C compiler ..... cc(1)
GNU project — C Compiler ..... gcc(1)
generate — C flowgraph ..... cflow(1)
the — C language preprocessor ..... cpp(1)
— C Obscure Feature Finder ..... coff(1)
maintain a tags file for a — C program ..... ctags(1)
— C program beautifier ..... cb(1)
a — C program checker ..... lint(1)
generate — C program cross-reference ..... cxref(1)
— C program debugger ..... ctrace(1)
generate — C program makefile dependencies ..... depend(1)
from a common object file produce a — C source listing with line numbers ..... list(1)
summarize — cache hit rate for each buffer size ..... bfree(1M)
print calendar — cal ..... cal(1)
characters — calculate an integer value from a string of .... getnum(3X)
desk — calculator ..... dc(1)
print — calendar ..... cal(1)
reminder service — calendar ..... calendar(1)
implementation-defined system — call ..... sys_local(2)
— call another UNIX system ..... cu(1C)
display — call graph profile data ..... gprof(1)
suspend the — calling process ..... suspend(2)
target the — calling process to some cpu ..... targetcpu(2)
fast main memory allocator — calloc ..... malloc(3C)
main memory allocator — calloc ..... malloc(3X)
SSP specific system service — calls ..... sspsys(2)
data returned by stat(2) and estat(2) system — calls ..... stat(5)
library routines for remote procedure — calls ..... rpc(3N)
introduction to system — calls and error numbers ..... intro(2)
send/cancel requests to an LP/ — cancel ..... lp(1)
— cancel one or more asynchronous I/O requests — acancel(2)
network — canonical "server" process for testing ..... sink(1)
terminal — capability data base ..... terminfo(4)
convert a termcap description/ — captoinfo ..... captoinfo(1M)
interpret ASA — carriage control characters ..... asa(1)
text editor (variant of ex for — casual users) ..... edit(1)
concatenate and print files — cat ..... cat(1)
C program beautifier — cb ..... cb(1)
C compiler — cc ..... cc(1)
change working directory — cd ..... cd(1)

serial ports on the MVME187	cd2401	cd2401(7)
change the delta commentary of/	cdc	cdc(1)
floor, ceiling, remainder,/	ceil	floor(3M)
floor,	ceiling, remainder, absolute value functions	floor(3M)
QWB multiprocessor/	cf	cf(1M)
get or set the value of the/	cfgetispeed	cfgetospeed(3P)
get or set the value of the/	cfgetospeed	cfgetospeed(3P)
generate C flowgraph	cflow	cflow(1)
get or set the value of the/	cfsetispeed	cfgetospeed(3P)
get or set the value of the/	cfsetospeed	cfgetospeed(3P)
convert date and time to string	cftime	ctime(3C)
language specific strings for/	cftime	cftime(4)
examine and	change blocked signals	sigprocmask(2P)
examine and	change bytes in a file	patch(1M)
	change data segment space allocation	brk(2)
	change group	chgrp(1)
	change login password and password attributes	passwd(1)
	change login password in NIS	yppasswd(1N)
	change mode	chmod(1)
	change mode of file	chmod(2)
	change or add value to environment	putenv(3C)
	change owner	chown(1)
	change owner and group of a file	chown(2)
	change owner or group	chown(1)
	change performance tunings of file systems	tunefs(1M)
	change priority of a process	nice(2)
	change root directory	chroot(2)
	change root directory for a command	chroot(1M)
examine or	change signal action	sigaction(2P)
shut down system,	change system state	shutdown(1M)
	change the delta commentary of a SCCS delta	cdc(1)
	change the format of a text file	newform(1)
	change the name of a file	rename(2)
	change the qwbd daemon debug trace flags	qwbdflags(1M)
	change working directory	cd(1)
	change working directory	chdir(2)
force propagation of a	changed NIS map	yppush(1M)
make	changes to the help facility database	helpadm(1M)
create an interprocess	channel	pipe(2)
push	character back into input stream	ungetc(3S)
generate	character classification and conversion tables	chrtbl(1M)
	character classification macros	ctype(3C)
get	character login name of the user	cuserid(3S)
get	character or word from a stream	getc(3S)
put	character or word on a stream	putc(3S)
map of ASCII	character set	ascii(5)
search a file for a	character string	fgrep(1)
display	character strings in object files	strings(1)
calculate an integer value from a string of	characters	getnum(3X)
interpret ASA carriage control	characters	asa(1)
translate	characters	conv(3C)
translate	characters	tr(1)
shell procedures for accounting	chargefee	acctsh(1M)
change working directory	chdir	chdir(2)
	check a file system on a diskette	checkfsys(1M)
file system	check and interactive repair	chk(1)
	check and repair file systems	fsck(1M)

get yes/no response from user or	check answer to question	chkyn(1)
	check file system backup schedule	ckbupscd(1M)
	check the uucp directories and permissions file	uucheck(1M)
a C program	checker	lint(1)
password/group file	checkers	pwck(1M)
check a file system on a/	checkfsys	checkfsys(1M)
list of file systems processed/	checklist	checklist(4)
print	checksum and block count of a file	sum(1)
generate cyclic redundancy	checksums (crc) of files	crc(1)
change group	chgrp	chgrp(1)
change owner or group	chgrp	chown(1)
get process and	child process times	times(2)
wait for	child process to stop or terminate	wait(2)
wait for	child process to stop or terminate	waitpid(2)
file system check and/	chk	chk(1)
get yes/no response from user/	chkyn	chkyn(1)
change mode of file	chmod	chmod(2)
change mode	chmod	chmod(1)
change owner and group of a file	chown	chown(2)
change owner or group	chown	chown(1)
change owner	chown	chown(1)
change root directory for a/	chroot	chroot(1M)
change root directory	chroot	chroot(2)
generate character/	chrtbl	chrtbl(1M)
connected interrupt interface	cintrio	cintrio(7)
connected interrupt structure	cintrio	cintrio(4)
wait for a connected interrupt	cisema	cisema(2)
check file system backup/	ckbupscd	ckbupscd(1M)
shell procedures for accounting	ckpacct	acctsh(1M)
generate character	classification and conversion tables	chrtbl(1M)
character	classification macros	ctype(3C)
uucp spool directory	clean-up	uucleanup(1M)
STREAMS error logger	cleanup program	strclean(1M)
	clear i-node	clri(1M)
	clear QWB memory	cm(1M)
stream status inquiries	clearerr	ferror(3S)
create NFS	client handles	nfsclnt(1M)
create kernel RPC	client handles	kclt_create(3N)
NIS	client interface	ypclnt(3N)
directories to export to NFS	clients	exports(4N)
report CPU time used	clock	clock(3C)
set a process alarm	clock	alarm(2)
system clock	clock	clock(7)
system	clock	clock(7)
the time to allow synchronization of the system	clock correct	adjtime(2)
	clock daemon	cron(1M)
open any minor device on a/	clone	clone(7)
close a file descriptor	close	close(2)
	close a common object file	ldclose(3X)
	close a file descriptor	close(2)
	close a shared memory name server database	nsclose(3C)
	close a transport endpoint	t_close(3N)
	close or flush a stream	fclose(3S)
directory operations	closedir	directory(3X)
clear i-node	clri	clri(1M)
set/clear realtime privileges	clrrt	setrt(2)
clear QWB memory	cm	cm(1M)

compare two files — cmp ..... cmp(1)
the binary system and commands from source — code remake ..... :mk(1M)
object — code disassembler ..... dis(1)
C Obscure Feature Finder — coff ..... coff(1)
patch XC88100 — COFF executables to avoid fdiv deadlock .... fdivfix(1)
compare common object format/ — coffcmp ..... coffcmp(1)
dump common object format/ — coffdump ..... coffdump(1)
filter reverse linefeeds — col ..... col(1)
remove columns from a file — colrm ..... colrm(1)
remove — columns from a file ..... colrm(1)
combine SCCS deltas — comb ..... comb(1)
— combine SCCS deltas ..... comb(1)
select or reject lines common/ — comm ..... comm(1)
change root directory for a — command ..... chroot(1M)
condition evaluation — command ..... test(1)
construct argument list(s) and execute — command ..... xargs(1)
issue a shell — command ..... system(3S)
routines for returning a stream to a remote — command ..... rcmd(3N)
set process group id and execute — command ..... setpgrp(1)
time a — command ..... time(1)
shell, the standard/restricted — command and programming language ..... sh(1)
run a — command at low priority ..... nice(1)
retrieve a — command description and usage examples .... usage(1)
alias file for framed access — command environment (FACE) utilities ..... alias(4)
system-wide framed access — command environment variables ..... env(4)
UNIX-to-UNIX system — command execution ..... uux(1C)
set environment for — command execution ..... env(1)
for generating/modifying ASSIST menus or — command forms program ..... astgen(1)
run a — command immune to hangups and quits ..... nohup(1)
run a — command on a particular cpu ..... target(1R)
parse — command options ..... getopt(1)
parse — command options ..... getopts(1)
shell, the standard/restricted — command programming language ..... ksh(1)
the bourne shell — command programming language ..... bsh(1)
execute a — command remotely ..... on(1N)
execute remote — command requests ..... uuxqt(1M)
extracts menus from labels stored in — command shell scripts ..... mkmenus(1M)
records — command summary from per-process accounting acctcms(1M)
partition db — command to dump system memory image to dump sysdump(1M)
identify a — command using keywords ..... locate(1)
activity time a — command; report process data and system ... timex(1)
STREAMS ioctl — commands ..... streamio(7)
assistance using REAL/IX — commands ..... assist(1)
install — commands ..... install(1M)
introduction to realtime — commands ..... intro(1R)
of accounting and miscellaneous accounting — commands overview ..... acct(1M)
introduction to maintenance — commands and application programs ..... intro(1M)
introduction to — commands and application programs ..... intro(1)
introduction to — commands and application programs ..... intro(1N)
execute — commands at a later time ..... at(1)
remake the binary system and — commands from source code ..... :mk(1M)
run — commands performed for multi-user environment rc2(1M)
environment run — commands performed for the SSP multiuser .. rcqwb(1M)
run — commands performed to stop the operating system rc0(1M)
manipulate the object file — comment section ..... mcs(1)
change the delta — commentary of a SCCS delta ..... cdc(1)
— common archive file format ..... ar(4)

	common assembler and link editor output	a.out(4)
control structures for the	common event mechanism	evt(4)
convert archive files to	common formats	convert(1)
close a	common object file	ldclose(3X)
compress a	common object file	cprs(1)
compute the index of a symbol table entry of a	common object file	ldtbindex(3X)
line number entries in a	common object file	linenum(4)
print name list of	common object file	nm(1)
read an indexed symbol table entry of a	common object file	ldtbread(3X)
read an indexed/named section header of a	common object file	ldshread(3X)
read the file header of a	common object file	ldfhread(3X)
relocation information for a	common object file	reloc(4)
section header for a	common object file	scnhdr(4)
seek to an indexed/named section of a	common object file	ldsseek(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)
seek to the optional file header of a	common object file	ldohseek(3X)
seek to the symbol table of a	common object file	ldtbseek(3X)
trip symbol and line number information from a	common object file	strip(1)
	common object file access routines	ldfcn(4)
	common object file converter	conv(1)
open a	common object file for reading	ldopen(3X)
manipulate line number entries of a	common object file function	ldlread(3X)
with line numbers from a	common object file produce a C source listing	list(1)
retrieve symbol name for	common object file symbol table entry	ldgetname(3X)
	common object file symbol table format	syms(4)
file header for	common object files	filehdr(4)
link editor for	common object files	ld(1)
print section sizes in bytes of	common object files	size(1)
compare	common object format modules	coffcmp(1)
dump	common object format modules	coffdump(1)
select or reject lines	common to two sorted files	comm(1)
create an endpoint for	communication	socket(2N)
MVME332XT	communication controller VME module interface	mvme332xt(7)
report inter-process	communication facilities status	ipcs(1)
standard interprocess	communication package	stdipc(3C)
differential file and directory	comparator	diff(1)
differential file	comparator	diff(1)
	compare common object format modules	coffcmp(1)
	compare or print out terminfo descriptions	infocmp(1M)
	compare two files	cmp(1)
	compare two versions of an SCCS file	sccsdiff(1)
3-way differential file	comparison	diff3(1)
directory	comparison	dircmp(1)
regular expression	compile	regcmp(1)
	compile and execute regular expression	regcmp(3X)
regular expression	compile and match routines	regexp(5)
format of	compiled term file.	term(4)
C	compiler	cc(1)
GNU project C	Compiler	gcc(1)
an RPC protocol	compiler	rpcgen(1N)
terminfo	compiler	tic(1M)
yet another	compiler-compiler	yacc(1)
error function and	complementary error function	erf(3M)
await	completion of process	wait(1)
force	compliance with a bill of materials	mkcomply(1M)
	compress a common object file	cprs(1)

	compress and expand files	pack(1)
common object file	compute the index of a symbol table entry of a	ldtbindex(3X)
biff server	comsat	comsat(1M)
	concatenate and print files	cat(1)
	condition evaluation command	test(1)
	conditionally lock a binary semaphore	bslkc(3C)
get	configurable pathname variables	pathconf(2P)
get	configurable system variables	sysconf(2P)
network interface	configuration	ifconfig(1M)
print system	configuration	prtconf(1M)
show socket	configuration	sckconf(1M)
QWB multiprocessor	configuration database manager	cf(1M)
host name lookup	configuration file	svcorder(4N)
resolver	configuration file	resolv.conf(4N)
network	configuration program	netc(1M)
NFS, NIS, and REX start/stop and	configuration scripts	nfs(1M)
	configure and generate the operating system	sysgen(1M)
	configure or display *tty* ports	portconfig(1M)
	configure SCSI devices for realtime operation	scsiconf(1M)
	configure the LP print service	lpadmin(1M)
receive the	confirmation from a connect request	t_rcvconnect(3N)
pseudo shell layer, console/	conman	conman(1)
initiate a connection on a/	connect	connect(2N)
manipulate	connect accounting records	fwtmp(1M)
accept a	connect request	t_accept(3N)
listen for a	connect request	t_listen(3N)
receive the confirmation from a	connect request	t_rcvconnect(3N)
	connect-time accounting	acctcon(1M)
wait for a	connected interrupt	cisema(2)
	connected interrupt interface	cintrio(7)
	connected interrupt structure	cintrio(4)
get name of	connected peer	getpeername(2N)
gets name of	connected peer	getpeername(3N)
establish an out-going terminal line	connection	dial(3C)
receive data or expedited data sent over a	connection	t_rcv(3N)
send data or expedited data over a	connection	t_snd(3N)
shut down part of a full-duplex	connection	shutdown(2N)
accept a	connection on a socket	accept(2N)
initiate a	connection on a socket	connect(2N)
establish a	connection with another transport user	t_connect(3N)
listen for	connections on a socket	listen(2N)
console interface	console	console(7)
	console interface	console(7)
Quadbus	console interface	qwbcon(7)
pseudo shell layer,	console manager	conman(1)
control dual	console operation	dcon(1M)
kernel	console printf logger	putbuf(1)
file header for implementation-specific	constants	limits(4)
file header for symbolic	constants	unistd(4)
math functions and	constants	math(5)
	construct a file system	fs(1)
	construct a file system	mkfs(1M)
	construct argument list(s) and execute command	xargs(1)
remove nroff/troff, tbl, and eqn	constructs	deroff(1)
try to	contact remote system with debugging on	uutry(1M)
displays or alters byte	content of files	bpatch(1)
display	contents of a MSDOS file	mtype(1)

list contents of directory ..... ls(1)
context split ..... csplit(1)
preallocate contiguous file space ..... prealloc(1R)
preallocate contiguous file space ..... prealloc(2)
administrative control ..... uadmin(1M)
administrative control ..... uadmin(2)
file control ..... fcntl(2)
magnetic tape control ..... mt(1)
uucp status inquiry and job control ..... uustat(1C)
version control ..... vc(1)
control block for asynchronous I/O operations aiocb(4)
interpret ASA carriage control characters ..... asa(1)
control device ..... ioctl(2)
control dual console operation ..... dcon(1M)
line control functions ..... tcdrain(3P)
multiprocessor process control initialization ..... mpinit(1M)
process control initialization ..... init(1M)
event control operations ..... evctl(2)
message control operations ..... msgctl(2)
semaphore control operations ..... semctl(2)
shared memory control operations ..... shmctl(2)
file control options ..... fcntl(5)
excersise control over IP network interfaces ..... ifctl(1M)
MVME332XT control program ..... m332xctl(1M)
MVME336 control program ..... m336ctl(1M)
MVME350 control program ..... m350ctl(1M)
MVME355 control program ..... m355ctl(1M)
tape device control program ..... tapectl(1M)
Internet Transmission Control Protocol ..... tcp(7)
control structures for messages ..... msg(4)
control structures for shared memory ..... shm(4)
mechanism control structures for the common event ..... evt(4)
control structures for user-level semaphores ... sem(4)
control write/execute attributes of memory ... memctl(2)
driver for the MODCOMP 4211 Peregrine FDDI Controller ..... fddi4211(7)
driver for the MVME355 9-Track Tape Controller ..... mvme355(7)
r the MODCOMP VLAN-E Intelligent Ethernet Controller driver ..... vlan-e(7)
autoconfig MVME187 SCSI controller interface ..... mvme187(7)
autoconfig MVME327 SCSI controller interface ..... mvme327(7)
MVME332XT communication controller VME module interface ..... mvme332xt(7)
MVME350 Streamer Tape Controller VME Module Interface ..... mvme350(7)
controlling terminal interface ..... tty(7)
common object file converter conv ..... conv(1)
conventional names for terminals ..... term(5)
conversion program ..... units(1)
NIS database conversion routines ..... map2ascii(1M)
generate character classification and conversion tables ..... chrtbl(1M)
convert archive files to common/ convert ..... convert(1)
description convert a termcap description into a terminfo . captoinfo(1M)
convert a virtual to a physical cpu number ... vcpuno(2)
convert and copy a file ..... dd(1M)
convert and copy a file ..... tt(1)
convert archive files to common formats ..... convert(1)
another convert bad track list from one format to .... xformtrk(1M)
integers convert between 3-byte integers and long ..... l3tol(3C)
string convert between long integer and base-64 ASCII a64l(3C)
convert date and time to string ..... ctime(3C)

	convert date and time to string	strftime(3C)
	convert floating-point number to string	ecvt(3C)
	convert formatted input	scanf(3S)
	convert string to double-precision number	strtod(3C)
	convert string to integer	strtol(3C)
block numbers	convert track numbers into file system logical	fblkgen(1M)
order	convert values between host and network byte	byteorder(3N)
common object file	converter	conv(1)
language specific strings for	converting times and dates to ASCII	cftime(4)
scheme.	converts from a two- to a one-password file	pwunconv(1M)
Transport Interface	cooperating STREAMS module	timod(7)
	copy	cp(1)
UNIX-to-UNIX system	copy	uucp(1C)
public UNIX-to-UNIX system file	copy	uuto(1C)
remote file	copy	rcp(1N)
convert and	copy a file	dd(1M)
convert and	copy a file	tt(1)
	copy file archives in and out	cpio(1)
	copy file systems for optimal access time	dcopy(1M)
make literal	copy of file system	volcopy(1M)
	copy removeable media	dcpy(1)
	copy, link, or move files	cp(1)
format of core image file	core	core(4)
save kernel	core dumps	savedump(1M)
format of	core image file	core(4)
save the	core image of a system dump in a file	savecore(1M)
	core memory	mem(7)
the system clock	correct the time to allow synchronization of	adjtime(2)
database map a name to a	corresponding id in a shared memory name server	nsntoi(3C)
server database map an id to a	corresponding name in a shared memory name	nsiton(3C)
trigonometric functions	cos	trig(3M)
hyperbolic functions	cosh	sinh(3M)
word	count	wc(1)
print checksum and block	count of a file	sum(1)
copy	cp	cp(1)
copy, link, or move files	cp	cp(1)
copy file archives in and out	cpio	cpio(1)
format of cpio archive	cpio	cpio(4)
format of	cpio archive	cpio(4)
the C language preprocessor	cpp	cpp(1)
compress a common object file	cprs	cprs(1)
install object files in binary/	cpset	cpset(1M)
run a command on a particular	cpu	target(1R)
target the calling process to some	cpu	targetcpu(2)
voluntarily give up	CPU	relinquish(2)
print	CPU board information	cputype(1)
convert a virtual to a physical	cpu number	vcpuno(2)
return the number of the	cpu this process is on	cpuno(2)
report	CPU time used	clock(3C)
some hardware and software data on a particular	cpu. return	getcpudata(2)
return the number of the cpu/	cpuno	cpuno(2)
print CPU board information	cputype	cputype(1)
examine system images	crash	crash(1M)
generate cyclic redundancy/	crc	crc(1)
create a new file or rewrite an/	creat	creat(2)
	create a file system on a diskette	makefsys(1M)
	create a name for a temporary file	tmpnam(3S)

	create a new file or rewrite an existing one	creat(2)
	create a new process	fork(2)
	create a new session	setsid(2)
	create a shared library	mkshlib(1)
	create a shared memory name server database	nscreat(3C)
	create a temporary file	tmpfile(3S)
	create an endpoint for communication	socket(2N)
	create an interprocess channel	pipe(2)
	create and administer SCCS files	admin(1)
	create IPC key from file name	ftok(3C)
	create kernel RPC client handles	kclt_create(3N)
	create NFS client handles	nfsclnt(1M)
	create shared memory segment	shmget(1)
set and get file	creation mask	umask(2)
clock daemon	cron	cron(1M)
user crontab file	crontab	crontab(1)
user	crontab file	crontab(1)
generate C program	cross-reference	cxref(1)
display system statistics on a	crt	systat(1)
file perusal filter for	crt viewing	more(1)
file perusal filter for	CRTs	pg(1)
generate hashing encryption	crypt	crypt(3C)
password and file encryption/	crypt	crypt(3X)
context split	csplit	csplit(1)
spawn getty to a remote terminal	ct	ct(1C)
maintain a tags file for a C/	ctags	ctags(1)
generate file name for terminal	ctermid	ctermid(3S)
convert date and time to string	ctime	ctime(3C)
C program debugger	ctrace	ctrace(1)
call another UNIX system	cu	cu(1C)
look at the	current event on a transport endpoint	t_look(3N)
get unique identifier of	current host	gethostid(3N)
get/set name of	current host	gethostname(3N)
set or print name of	current host system	hostname(1N)
set or query	current locale	setlocale(3C)
get/set name of	current NIS domain	getdomainname(3N)
set or display name of	current NIS domain	domainname(1N)
print	current SCCS file editing activity	sact(1)
get the	current state	t_getstate(3N)
get name of	current system	uname(2)
print name of	current system	uname(1)
find the slot in the utmp file of the	current user	ttyslot(3C)
print effective	current user id	whoami(1)
get the	current value for a process interval timer	getinterval(2)
get the	current value for a system-wide realtime timer	gettimer(2)
set the	current value for a system-wide realtime timer	settimer(2)
get path-name of	current working directory	getcwd(3C)
terminal screen handling and/	curses	curses(3X)
format of	curses screen image file	scr_dump(4)
get character login name of the/	cuserid	cuserid(3S)
cut out selected fields of each/	cut	cut(1)
	cut out selected fields of each line of a file	cut(1)
generate C program/	cxref	cxref(1)
generate	cyclic redundancy checksums (crc) of files	crc(1)
STREAMS error logger	daemon	strerr(1M)
Transport Provider Interface	daemon	tpid(1M)
clock	daemon	cron(1M)

error-logging	daemon	errdemon(1M)
network printer	daemon	netlpd(1N)
change the qwbd	daemon debug trace flags	qwbdflags(1M)
null modem	daemon pair	nmd(1M)
NFS	daemons	nfsd(1M)
qwb	daemons	qwbd(1M)
initialization file for the qwbd	daemons.	qwbtab(4)
	DARPA Internet File Transfer Protocol server	ftpd(1M)
	DARPA port to RPC program number mapper	portmap(1M)
	DARPA TELNET protocol server	telnetd(1M)
	DARPA Trivial File Transfer Protocol server	tftpd(1M)
handle special functions of	DASI 300 and 300s terminals	300(1)
handle special functions of the	DASI 450 terminal	450(1)
display call graph profile	data	gprof(1)
display profile	data	prof(1)
return some systemwide hardware and software	data	getsysdata(2)
time a command; report process	data and system activity	timex(1)
host name	data base	hosts(4N)
protocol name	data base	protocols(4N)
rpc program number	data base	rpc(4N)
service name	data base	services(4N)
terminal capability	data base	terminfo(4)
generate disk accounting	data by user ID	diskusg(1M)
receive a unit	data error indication	t_rcvuderr(3N)
receive	data from a socket	recv(2N)
access long integer	data in a machine-independent fashion	sputl(3X)
lock process, text, or	data in memory	plock(2)
return some hardware and software	data on a particular cpu.	getcpudata(2)
send	data or expedited data over a connection	t_snd(3N)
receive	data or expedited data sent over a connection	t_rcv(3N)
send data or expedited	data over a connection	t_snd(3N)
send	data over a socket	send(2N)
library routines for external	data representation	xdr(3N)
calls	data returned by stat(2) and estat(2) system	stat(5)
expand the stack region of the	data segment	stkexp(2)
change	data segment space allocation	brk(2)
receive data or expedited	data sent over a connection	t_rcv(3N)
primitive system	data types	types(5)
receive a	data unit	t_rcvudata(3N)
send a	data unit	t_sndudata(3N)
a record from a shared memory name server	database delete	nsdelete(3C)
a record into a shared memory name server	database insert	nsinsert(3C)
build and install NIS	database	ypinit(1M)
close a shared memory name server	database	nsclose(3C)
corresponding id in a shared memory name server	database map a name to a	nsntoi(3C)
create a shared memory name server	database	nscreat(3C)
get a record from a shared memory name server	database	nsget(3C)
initialize a terminal or query terminfo	database	tput(1)
make changes to the help facility	database	helpadm(1M)
name in a shared memory name server	database map an id to a corresponding	nsiton(3C)
network name	database	networks(4N)
open a shared memory name server	database	nsopen(3C)
print values in a NIS	database	ypcat(1N)
rebuild NIS	database	ypmake(1M)
remove a shared memory name server	database	nsremove(3C)
the NIS	database and directory structure	ypfiles(4N)
NIS	database conversion routines	map2ascii(1M)

QWB multiprocessor configuration — database manager ..... cf(1M)
relational — database operator ..... join(1)
— database subroutines ..... dbm(3N)
Internet User — Datagram Protocol ..... udp(7)
print and set the date — date ..... date(1)
print and set the — date ..... date(1)
convert — date and time to string ..... ctime(3C)
convert — date and time to string ..... strftime(3C)
specific strings for converting times and — dates to ASCII language ..... cftime(4)
print the number of — days since the Epoch ..... ypday(1M)
partition — db command to dump system memory image to dump sysdump(1M)
make an NIS — dbm file ..... makedbm(1M)
database subroutines — dbmclose ..... dbm(3N)
database subroutines — dbminit ..... dbm(3N)
debugger — dbx ..... dbx(1)
desk calculator — dc ..... dc(1)
control dual console operation — dcon ..... dcon(1M)
copy file systems for optimal/ — dcopy ..... dcopy(1M)
copy removeable media — dcpy ..... dcpy(1)
convert and copy a file — dd ..... dd(1M)
disk definition information/ — ddefs ..... ddefs(1M)
patch XC88100 COFF executables to avoid fdiv — deadlock ..... fdivfix(1)
change the qwbd daemon — debug trace flags ..... qwbdflags(1M)
— debugger ..... dbx(1)
C program — debugger ..... ctrace(1)
SSP interface to the multiprocessor system — debugger ..... nmi(1M)
file system — debugger ..... fsdb(1M)
kernel — debugger ..... kdb(1M)
symbolic — debugger ..... sdb(1)
kernel — debugger symbol table generator ..... unixsyms(1)
try to contact remote system with — debugging on ..... uutry(1M)
— decode semaphore processor instructions ..... smp(1M)
set — default system time zone ..... timezone(4)
disk — definition information manager ..... ddefs(1M)
— definitions of terms and symbols ..... glossary(1)
database subroutines — delete ..... dbm(3N)
— delete a MSDOS file ..... mdel(1)
server database — delete a record from a shared memory name .. nsdelete(3C)
— deliver portions of pathnames ..... basename(1)
— deliver the last part of a file ..... tail(1)
change the delta commentary of a SCCS — delta ..... cdc(1)
make a delta to a SCCS file — delta ..... delta(1)
change the — delta commentary of a SCCS delta ..... cdc(1)
remove a — delta from an SCCS file ..... rmdel(1)
make a — delta to a SCCS file ..... delta(1)
combine SCCS — deltas ..... comb(1)
permit or — deny messages ..... mesg(1)
generate C program makefile/ — depend ..... depend(1)
generate C program makefile — dependencies ..... depend(1)
patch program with **NULL** pointer — dereference bug ..... dnp(1)
remove nroff/troff, tbl, and/ — deroff ..... deroff(1)
convert a termcap description into a terminfo — description ..... captoinfo(1M)
retrieve a command — description and usage examples ..... usage(1)
convert a termcap — description into a terminfo description ..... captoinfo(1M)
compare or print out terminfo — descriptions ..... infocmp(1M)
close a file — descriptor ..... close(2)
duplicate an open file — descriptor ..... dup(2)

duplicate an open file	descriptor	dup2(3C)
	desk calculator	dc(1)
autoconfiguration/ general tape implementation	details used by REAL/IX realtime SCSI	scsitape(7)
general 9-track tape implementation	details used by REAL/IX realtime SCSI/	scsi9track(7)
	determine accessibility of a file	access(2)
	determine file type	file(1)
control	device	ioctl(2)
fold long lines for finite width output	device	fold(1)
tape	device control program	tapectl(1M)
Quadbus	device driver interface to SSP devices	qwbsys(7)
SSP	device driver interface to the MP system	qwbserv(7)
find	device filename from major and minor numbers	getdevname(1M)
alias echo the real	device in the permissions file for a given	real(1)
	device name	devnm(1M)
open any minor	device on a STREAMS driver	clone(7)
Quadbus device driver interface to SSP	devices	qwbsys(7)
recondition pty slave	devices	ptyfix(1M)
	devices administered by System Administration	sa(7)
configure SCSI	devices for realtime operation	scsiconf(1M)
device name	devnm	devnm(1M)
report number of free disk/	df	df(1M)
check and repair file systems	dfsck	fsck(1M)
VLAN-E	diagnostic utility	vlandiag(1M)
establish an out-going terminal/	dial	dial(3C)
big	diff	bdiff(1)
differential file and directory/	diff	diff(1)
differential file comparator	diff	diff(1)
3-way differential file/	diff3	diff3(1)
side-by-side	difference program	sdiff(1)
	differential file and directory comparator	diff(1)
	differential file comparator	diff(1)
3-way	differential file comparison	diff3(1)
disk initializer	dinit	dinit(1M)
format of directories	dir	dir(4)
directory comparison	dircmp	dircmp(1)
format of	directories	dir(4)
install object files in binary	directories	cpset(1M)
link and unlink files and	directories	link(1M)
make	directories	mkdir(1)
remove files or	directories	rm(1)
check the uucp	directories and permissions file	uucheck(1M)
	directories to export to NFS clients	exports(4N)
change root	directory	chroot(2)
change working	directory	cd(1)
change working	directory	chdir(2)
display a MSDOS	directory	mdir(1)
get path-name of current working	directory	getcwd(3C)
list contents of	directory	ls(1)
make a MSDOS sub	directory	mmd(1)
make a	directory	mkdir(2)
move a	directory	mvdir(1M)
remove a MSDOS sub	directory	mrd(1)
remove a	directory	rmdir(2)
uucp spool	directory clean-up	uucleanup(1M)
differential file and	directory comparator	diff(1)
	directory comparison	dircmp(1)
independent format read	directory entries and put in a file system	getdents(2)

file system independent	directory entry	dirent(4)
remove	directory entry	unlink(2)
change root	directory for a command	chroot(1M)
make a lost+found	directory for fsck	mklost+found(1M)
working	directory name	pwd(1)
	directory operations	directory(3X)
the NIS database and	directory structure	ypfiles(4N)
make a	directory, or a special or ordinary file	mknod(2)
file system independent/	dirent	dirent(4)
deliver portions of pathnames	dirname	basename(1)
object code disassembler	dis	dis(1)
enable/disable LP printers	disable	enable(1)
	disable a transport endpoint	t_unbind(3N)
enable or	disable process accounting	acct(2)
object code	disassembler	dis(1)
set terminal type, modes, speed, and line	discipline	getty(1M)
set terminal type, modes, speed, and line	discipline	uugetty(1M)
retrieve information from	disconnect	t_rcvdis(3N)
send user-initiated	disconnect request	t_snddis(3N)
format a MSDOS floppy	disk	mformat(1)
synchronize a file's in-core state with that on	disk	fsync(2)
permissions file used by the value-added	disk access utilities	filesys(4)
generate	disk accounting data by user ID	diskusg(1M)
report number of free	disk blocks and i-nodes	df(1M)
	disk definition information manager	ddefs(1M)
	disk initializer	dinit(1M)
	disk initializer	fmt(1)
summarize	disk usage	du(1M)
check a file system on a	diskette	checkfsys(1M)
create a file system on a	diskette	makefsys(1M)
mount, unmount a	diskette file system	mountfsys(1M)
generate disk accounting data/	diskusg	diskusg(1M)
mount and	dismount file system	mnt(1)
	display a MSDOS directory	mdir(1)
	display call graph profile data	gprof(1)
	display character strings in object files	strings(1)
	display contents of a MSDOS file	mtype(1)
screen-oriented (visual)	display editor based on ex	vi(1)
	display file system name	fstyp(1M)
	display lock and semaphore statistics	semstat(1M)
set or	display name of current NIS domain	domainname(1N)
	display profile data	prof(1)
	display statistics for supported IP interfaces	ifstat(1M)
	display system statistics on a crt	systat(1)
configure or	display *tty* ports	portconfig(1M)
	displays or alters byte content of files	bpatch(1)
Euclidean	distance function	hypot(3M)
get	distinguished process group ID	tcgetpgrp(3P)
set	distinguished process group ID	tcsetpgrp(3P)
generate uniformly	distributed pseudo-random numbers	drand48(3C)
patch program with/	dnp	dnp(1)
resolver routines	dn_comp	resolver(3N)
resolver routines	dn_expand	resolver(3N)
shell procedures for accounting	dodisk	acctsh(1M)
who is	doing what	whodo(1M)
get/set name of current NIS	domain	getdomainname(3N)
set or display name of current NIS	domain	domainname(1N)

set or display name of current/ domainname ..... domainname(1N)
convert string to double-precision number ..... strtod(3C)
generate uniformly distributed/ drand48 ..... drand48(3C)
Environment Monitor Board driver ..... emb(7)
open any minor device on a STREAMS driver ..... clone(7)
pseudo-device driver ..... sxt(7)
pseudo-terminal driver ..... pty(7)
Network Interface Driver for MVME187 Boards ..... ifen596(7N)
Controller driver for the MODCOMP 4211 Peregrine FDDI fddi4211(7)
Ethernet Controller driver for the MODCOMP VLAN-E Intelligent vlan-e(7)
driver for the MVME355 9-Track Tape Controller mvme355(7)
general non-volatile RAM driver for the REAL/IX Operating System ... nvram(7)
general ramdisk driver for the REAL/IX System ..... ramdisk(7)
Quadbus device driver interface to SSP devices ..... qwbsys(7)
SSP device driver interface to the MP system ..... qwbserv(7)
assembler driver script ..... as(1)
used by REAL/IX realtime SCSI autoconfiguration drivers /9-track tape implementation details . scsi9track(7)
used by REAL/IX realtime SCSI autoconfiguration drivers general tape implementation details .. scsitape(7)
summarize disk usage du ..... du(1M)
control dual console operation ..... dcon(1M)
dump selected parts of an/ dump ..... dump(1)
extract error records from dump ..... errdead(1M)
octal dump ..... od(1)
dump common object format modules ..... coffdump(1)
save the core image of a system dump in a file ..... savecore(1M)
db command to dump system memory image to dump partition ..... sysdump(1M)
dump selected parts of an object file ..... dump(1)
db command to dump system memory image to dump partition sysdump(1M)
save kernel core dumps ..... savedump(1M)
duplicate an open file/ dup ..... dup(2)
duplicate an open file/ dup2 ..... dup2(3C)
duplicate an open file descriptor ..... dup(2)
duplicate an open file descriptor ..... dup2(3C)
echo arguments echo ..... echo(1)
echo arguments ..... echo(1)
for a given alias echo the real device in the permissions file .... real(1)
send ICMP ECHO_REQUEST packets to network hosts . ping(1M)
convert floating-point number/ ecvt ..... ecvt(3C)
text editor ed ..... ed(1)
last locations in program edata ..... end(3C)
text editor (variant of ex for/ edit ..... edit(1)
print current SCCS file editing activity ..... sact(1)
slice table editor ..... sledit(1M)
stream editor ..... sed(1)
text editor ..... ed(1)
text editor ..... ex(1)
text editor (variant of ex for casual users) ..... edit(1)
screen-oriented (visual) display editor based on ex ..... vi(1)
link editor for common object files ..... ld(1)
common assembler and link editor output ..... a.out(4)
print effective current user id ..... whoami(1)
get real user, effective user, real group, and effective group IDs ..... getuid(2)
IDs get real user, effective user, real group, and effective group . getuid(2)
get extended file status efstat ..... estat(2)
search a file for a pattern/ egrep ..... egrep(1)
Environment Monitor Board driver emb ..... emb(7)
enable/disable LP printers enable ..... enable(1)

	enable or disable process accounting	acct(2)
	enable/disable LP printers	enable(1)
generate hashing encryption	encrypt	crypt(3C)
generate hashing	encryption	crypt(3C)
password and file	encryption functions	crypt(3X)
generate	encryption key	makekey(1)
last locations in program	end	end(3C)
get group file entry	endgrent	getgrent(3C)
get network host entry	endhostent	gethostent(3N)
get network entry	endnetent	getnetent(3N)
get network group entry	endnetgrent	getnetgrent(3N)
bind an address to a transport	endpoint	t_bind(3N)
close a transport	endpoint	t_close(3N)
disable a transport	endpoint	t_unbind(3N)
establish a transport	endpoint	t_open(3N)
look at the current event on a transport	endpoint	t_look(3N)
manage options for a transport	endpoint	t_optmgmt(3N)
create an	endpoint for communication	socket(2N)
get protocol entry	endprotoent	getprotoent(3N)
get password file entry	endpwent	getpwent(3C)
get service entry	endservent	getservent(3N)
get shadow password file entry	endspent	getspent(3X)
access utmp file entry	endutent	getut(3C)
Motorola	enhanced file tree walk	motftw(3C)
format read directory	entries and put in a file system independent	getdents(2)
get	entries from name list	nlist(3C)
line number	entries in a common object file	linenum(4)
print	entries in this manual	man(1)
manipulate line number	entries of a common object file function	ldlread(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)
access utmp file	entry	getut(3C)
file system independent directory	entry	dirent(4)
get RPC	entry	getrpcent(3N)
get group file	entry	getgrent(3C)
get network group	entry	getnetgrent(3N)
get network host	entry	gethostent(3N)
get network	entry	getnetent(3N)
get password file	entry	getpwent(3C)
get protocol	entry	getprotoent(3N)
get service	entry	getservent(3N)
get shadow password file	entry	getspent(3X)
remove directory	entry	unlink(2)
/mbol name for common object file symbol table	entry retrieve	ldgetname(3X)
write password file	entry	putpwent(3C)
write shadow password file	entry	putspent(3X)
utmp and wtmp	entry formats	utmp(4)
compute the index of a symbol table	entry of a common object file	ldtbindex(3X)
read an indexed symbol table	entry of a common object file	ldtbread(3X)
set environment for command/	env	env(1)
system-wide framed access/	environ	env(4)
user environment	environ	environ(5)
change or add value to	environment	putenv(3C)
run commands performed for multi-user	environment	rc2(1M)
run commands performed for the SSP multiuser	environment	rcqwb(1M)
user	environment	environ(5)
alias file for framed access command	environment (FACE) utilities	alias(4)

setting up an	environment at login time	profile(4)
set	environment for command execution	env(1)
	Environment Monitor Board driver	emb(7)
return value for	environment name	getenv(3C)
system-wide framed access command	environment variables	env(4)
print the number of days since the	Epoch	ypday(1M)
remove nroff/troff, tbl, and	eqn constructs	deroff(1)
user-specified file of	equivalent hosts and users	rhosts(4)
names of hosts with	equivalent users, i.e., "trusted" hosts	hosts.equiv(4N)
generate uniformly distributed/	erand48	drand48(3C)
error function and/	erf	erf(3M)
error function and/	erfc	erf(3M)
extract error records from dump	errdead	errdead(1M)
error-logging daemon	errdemon	errdemon(1M)
error-log file format	errfile	errfile(4)
system error messages	errno	perror(3C)
error function and complementary	error function	erf(3M)
	error function and complementary error function	erf(3M)
receive a unit data	error indication	t_rcvuderr(3N)
STREAMS	error logger cleanup program	strclean(1M)
STREAMS	error logger daemon	strerr(1M)
interface to STREAMS	error logging and event tracing	log(7)
produce	error message	t_error(3N)
system	error messages	perror(3C)
introduction to system calls and	error numbers	intro(2)
extract	error records from dump	errdead(1M)
	error-handling function	matherr(3M)
	error-log file format	errfile(4)
	error-logging daemon	errdemon(1M)
find spelling	errors	spell(1)
process a report of logged	errors	errpt(1M)
process a report of logged/	errpt	errpt(1M)
user	establish a connection with another transport	t_connect(3N)
	establish a transport endpoint	t_open(3N)
	establish an out-going terminal line connection	dial(3C)
	establish mount table	setmnt(1M)
get extended file status	estat	estat(2)
data returned by stat(2) and	estat(2) system calls	stat(5)
last locations in program	etext	end(3C)
driver for the MODCOMP VLAN-E Intelligent	Ethernet Controller	vlan-e(7)
	Euclidean distance function	hypot(3M)
	evaluate arguments as an expression	expr(1)
condition	evaluation command	test(1)
event control operations	evctl	evctl(2)
receive any queued	event	evrcv(2)
	event control operations	evctl(2)
receive any queued	event from a specified list	evrcvl(2)
get an	event identifier	evget(2)
release an	event identifier	evrel(2)
control structures for the common	event mechanism	evt(4)
look at the current	event on a transport endpoint	t_look(3N)
post an	event to a process	evpost(2)
interface to STREAMS error logging and	event tracing	log(7)
receive multiple queued	events	evrcvm(2)
control structures for the/	event_t	evt(4)
get an event identifier	evget	evget(2)
post an event to a process	evpost	evpost(2)

receive any queued event	evrcv	evrcv(2)
receive any queued event from a/	evrcvl	evrcvl(2)
receive multiple queued events	evrcvm	evrcvm(2)
release an event identifier	evrel	evrel(2)
control structures for the/	evt_t	evt(4)
(visual) display editor based on	ex screen-oriented	vi(1)
text editor	ex	ex(1)
text editor (variant of	ex for casual users)	edit(1)
	examine and change blocked signals	sigprocmask(2P)
	examine and change bytes in a file	patch(1M)
	examine or change signal action	sigaction(2P)
	examine pending signals	sigpending(2P)
	examine system images	crash(1M)
retrieve a command description and usage	examples	usage(1)
	excersise control over IP network interfaces	ifctl(1M)
execute a file	execl	exec(2)
execute a file	execle	exec(2)
execute a file	execlp	exec(2)
patch XC88100 COFF	executables to avoid fdiv deadlock	fdivfix(1)
	execute a command remotely	on(1N)
	execute a file	exec(2)
construct argument list(s) and	execute command	xargs(1)
set process group id and	execute command	setpgrp(1)
	execute commands at a later time	at(1)
compile and	execute regular expression	regcmp(3X)
	execute remote command requests	uuxqt(1M)
UNIX-to-UNIX system command	execution	uux(1C)
set environment for command	execution	env(1)
suspend	execution for an interval	sleep(1)
suspend	execution for interval	sleep(3C)
prepare	execution profile	monitor(3C)
remote	execution protocol	rex(3N)
RPC-based remote	execution server	rexd(1M)
	execution time profile	profil(2)
execute a file	execv	exec(2)
execute a file	execve	exec(2)
execute a file	execvp	exec(2)
rename an	existing MSDOS file	mren(1)
create a new file or rewrite an	existing one	creat(2)
terminate process	exit	exit(2)
exponential, logarithm, power,/	exp	exp(3M)
expand tabs to spaces, and vice/	expand	expand(1)
compress and	expand files	pack(1)
	expand tabs to spaces, and vice versa	expand(1)
	expand the stack region of the data segment	stkexp(2)
send data or	expedited data over a connection	t_snd(3N)
receive data or	expedited data sent over a connection	t_rcv(3N)
set the	expiration time of a process interval timer	absinterval(2)
functions	exponential, logarithm, power, square root	exp(3M)
directories to	export to NFS clients	exports(4N)
directories to export to NFS/	exports	exports(4N)
evaluate arguments as an/	expr	expr(1)
compile and execute regular	expression	regcmp(3X)
evaluate arguments as an	expression	expr(1)
regular	expression compile	regcmp(1)
regular	expression compile and match routines	regexp(5)
search a file for a pattern using full regular	expressions	egrep(1)

get	extended file status	estat(2)
library routines for	external data representation	xdr(3N)
	extract error records from dump	errdead(1M)
software management package	extraction facility	ixf(1M)
shell scripts	extracts menus from labels stored in command	mkmenus(1M)
shutdown the system with	extreme prejudice	halt(1M)
floor, ceiling, remainder,/	fabs	floor(3M)
report inter-process communication	facilities status	ipcs(1)
help	facility	help(1)
software management package extraction	facility	ixf(1M)
software management package generation	facility	igf(1M)
make changes to the help	facility database	helpadm(1M)
obtain the prime factors of a/	factor	factor(1)
obtain the prime	factors of a number	factor(1)
log of	failed login attempts	loginlog(4)
provide truth values	false	true(1)
Internet protocol	family	inet(7N)
long integer data in a machine-independent	fashion access	sputl(3X)
	fast incremental backup	finc(1M)
	fast main memory allocator	malloc(3C)
generate an IOT	fault	abort(3C)
convert track numbers into file/	fblkgen	fblkgen(1M)
change mode of file	fchmod	chmod(2)
change owner and group of a file	fchown	chown(2)
close or flush a stream	fclose	fclose(3S)
file control options	fcntl	fcntl(5)
file control	fcntl	fcntl(2)
convert floating-point number/	fcvt	ecvt(3C)
driver for the MODCOMP 4211 Peregrine	FDDI Controller	fddi4211(7)
driver for the MODCOMP 4211/	fddi4211	fddi4211(7)
patch XC88100 COFF executables to avoid	fdiv deadlock	fdivfix(1)
patch XC88100 COFF executables/	fdivfix	fdivfix(1)
open a stream	fdopen	fopen(3S)
C Obscure	Feature Finder	coff(1)
stream status inquiries	feof	ferror(3S)
stream status inquiries	ferror	ferror(3S)
database subroutines	fetch	dbm(3N)
	fetch system load	getsysload(2)
give first	few lines	head(1)
list file names and statistics/	ff	ff(1M)
close or flush a stream	fflush	fclose(3S)
get character or word from a/	fgetc	getc(3S)
get group file entry	fgetgrent	getgrent(3C)
get password file entry	fgetpwent	getpwent(3C)
get a string from a stream	fgets	gets(3S)
get shadow password file entry	fgetspent	getspent(3X)
search a file for a character/	fgrep	fgrep(1)
build special	file	mknod(1M)
change mode of	file	chmod(2)
change owner and group of a	file	chown(2)
change the format of a text	file	newform(1)
change the name of a	file	rename(2)
check the uucp directories and permissions	file	uucheck(1M)
close a common object	file	ldclose(3X)
compare two versions of an SCCS	file	sccsdiff(1)
compress a common object	file	cprs(1)
convert and copy a	file	dd(1M)

convert and copy a file ..... tt(1)
create a name for a temporary file ..... tmpnam(3S)
create a temporary file ..... tmpfile(3S)
cut out selected fields of each line of a file ..... cut(1)
delete a MSDOS file ..... mdel(1)
deliver the last part of a file ..... tail(1)
determine accessibility of a file ..... access(2)
determine file type file ..... file(1)
display contents of a MSDOS file ..... mtype(1)
do multiple reads from a file ..... readv(2)
do multiple writes from a file ..... writev(2)
dump selected parts of an object file ..... dump(1)
entries of a section of a common object file seek to relocation ..... ldrseek(3X)
examine and change bytes in a file ..... patch(1M)
execute a file ..... exec(2)
format of SCCS file ..... sccsfile(4)
format of core image file ..... core(4)
format of curses screen image file ..... scr_dump(4)
get a version of an SCCS file ..... get(1)
group file ..... group(4)
host name lookup configuration file ..... svcorder(4N)
indexed symbol table entry of a common object file read an ..... ldtbread(3X)
lexed/named section header of a common object file read an ..... ldshread(3X)
issue identification file ..... issue(4)
line number entries in a common object file ..... linenum(4)
line number information from a common object file strip symbol and ..... strip(1)
link to a file ..... link(2)
make a delta to a SCCS file ..... delta(1)
make a directory, or a special or ordinary file ..... mknod(2)
make an NIS dbm file ..... makedbm(1M)
make symbolic link to a file ..... symlink(2)
map process addresses to a file ..... mmap(2)
number entries of a section of a common object file seek to line ..... ldlseek(3X)
of a symbol table entry of a common object file compute the index ..... ldtbindex(3X)
of several files or subsequent lines of one file merge same lines ..... paste(1)
password file ..... passwd(4)
print an SCCS file ..... prs(1)
print checksum and block count of a file ..... sum(1)
print name list of common object file ..... nm(1)
read from file ..... read(2)
read the file header of a common object file ..... ldfhread(3X)
read the permissions file ..... getpermsintro(3X2)
relocation information for a common object file ..... reloc(4)
remove a delta from an SCCS file ..... rmdel(1)
remove columns from a file ..... colrm(1)
rename an existing MSDOS file ..... mren(1)
report repeated lines in a file ..... uniq(1)
resolver configuration file ..... resolv.conf(4N)
save the core image of a system dump in a file ..... savecore(1M)
section header for a common object file ..... scnhdr(4)
seek to the symbol table of a common object file ..... ldtbseek(3X)
server for modifying NIS password file ..... yppasswdd(1M)
the archive header of a member of an archive file read ..... ldahread(3X)
the null file ..... null(7)
o an indexed/named section of a common object file seek ..... ldsseek(3X)
to the optional file header of a common object file seek ..... ldohseek(3X)
undo a previous get of an SCCS file ..... unget(1)

update access and modification times of a file ..... touch(1)
user crontab file ..... crontab(1)
validate SCCS file ..... val(1)
write to a file ..... write(2)
set file access and modification times ..... utime(2)
common object file access routines ..... ldfcn(4)
differential file and directory comparator ..... diff(1)
tape file archiver ..... tar(1)
copy file archives in and out ..... cpio(1)
password/group file checkers ..... pwck(1M)
manipulate the object file comment section ..... mcs(1)
differential file comparator ..... diff(1)
3-way differential file comparison ..... diff3(1)
file control ..... fcntl(2)
file control options ..... fcntl(5)
common object file converter ..... conv(1)
public UNIX-to-UNIX system file copy ..... uuto(1C)
remote file copy ..... rcp(1N)
set and get file creation mask ..... umask(2)
close a file descriptor ..... close(2)
duplicate an open file descriptor ..... dup(2)
duplicate an open file descriptor ..... dup2(3C)
print current SCCS file editing activity ..... sact(1)
password and file encryption functions ..... crypt(3X)
access utmp file entry ..... getut(3C)
get group file entry ..... getgrent(3C)
get password file entry ..... getpwent(3C)
get shadow password file entry ..... getspent(3X)
write password file entry ..... putpwent(3C)
write shadow password file entry ..... putspent(3X)
maintain a tags file for a C program ..... ctags(1)
search a file for a character string ..... fgrep(1)
echo the real device in the permissions file for a given alias ..... real(1)
search a file for a pattern ..... grep(1)
expressions search a file for a pattern using full regular ..... egrep(1)
(FACE) utilities alias file for framed access command environment . alias(4)
open a common object file for reading ..... ldopen(3X)
initialization file for the qwbd daemons. ..... qwbtab(4)
common archive file format ..... ar(4)
error-log file format ..... errfile(4)
per-process accounting file format ..... acct(4)
introduction to networking file formats ..... intro(4N)
introduction to file formats ..... intro(4)
line number entries of a common object file function manipulate ..... ldlread(3X)
file header for common object files ..... filehdr(4)
constants file header for implementation-specific ..... limits(4)
file header for symbolic constants ..... unistd(4)
read the file header of a common object file ..... ldfhread(3X)
seek to the optional file header of a common object file ..... ldohseek(3X)
read from file in an asynchronous manner ..... aread(2)
write to file in an asynchronous manner ..... awrite(2)
merge an a.out file into another a.out file. ..... scsiload(1M)
split a file into pieces ..... split(1)
password file management for the NIS ..... yppassmgmt(1M)
create IPC key from file name ..... ftok(3C)
make a unique file name ..... mktemp(3C)
generate file name for terminal ..... ctermid(3S)

list	file names and statistics for a file system	ff(1M)
user-specified	file of equivalent hosts and users	rhosts(4)
find the slot in the utmp	file of the current user	ttyslot(3C)
identify processes using a	file or file structure	fuser(1M)
create a new	file or rewrite an existing one	creat(2)
	file perusal filter for crt viewing	more(1)
	file perusal filter for CRTs	pg(1)
move read/write	file pointer	lseek(2)
reposition a	file pointer in a stream	fseek(3S)
numbers from a common object	file produce a C source listing with line	list(1)
big	file scanner	bfs(1)
converts from a two- to a one-password	file scheme.	pwunconv(1M)
summarize	file size usage by file system	icount(1M)
preallocate contiguous	file space	prealloc(1R)
preallocate contiguous	file space	prealloc(2)
truncate	file space	ftruncate(2)
truncate	file space	trunc(2)
get extended	file status	estat(2)
get	file status	stat(2)
identify processes using a file or	file structure	fuser(1M)
retrieve symbol name for common object	file symbol table entry	ldgetname(3X)
common object	file symbol table format	syms(4)
construct a	file system	fs(1)
construct a	file system	mkfs(1M)
list file names and statistics for a	file system	ff(1M)
make literal copy of	file system	volcopy(1M)
mount a	file system	()
mount a	file system	mount(2)
mount and dismount	file system	mnt(1)
mount, unmount a diskette	file system	mountfsys(1M)
summarize file size usage by	file system	icount(1M)
unmount a	file system	umount(2)
check	file system backup schedule	ckbupscd(1M)
	file system block analyzer	fsba(1M)
	file system check and interactive repair	chk(1)
	file system debugger	fsdb(1M)
	file system independent directory entry	dirent(4)
read directory entries and put in a	file system independent format	getdents(2)
get	file system information	statfs(2)
print various	file system information	fsinfo(1M)
convert track numbers into	file system logical block numbers	fblkgen(1M)
display	file system name	fstyp(1M)
check a	file system on a diskette	checkfsys(1M)
create a	file system on a diskette	makefsys(1M)
get	file system statistics	ustat(2)
report	file system status	fsstat(1M)
NFS remotely mounted	file system table	rmtab(4N)
mounted	file system table	mnttab(4)
get	file system type information	sysfs(2)
automatically mount NFS	file systems	automount(1M)
back up and restore files and/or	file systems	br(1M)
change performance tunings of	file systems	tunefs(1M)
check and repair	file systems	fsck(1M)
keep track of remotely mounted	file systems	mount(3N)
mount, unmount multiple	file systems	mountall(1M)
mount, unmount multiple	file systems	nmountall(1M)
provide labels for	file systems	labelit(1M)

mount and unmount	file systems and remote resources	mount(1M)
copy	file systems for optimal access time	dcopy(1M)
list of	file systems processed by fsck and ncheck	checklist(4)
write (copy) a Unix	file to MSDOS	mwrite(1)
read (copy) a MSDOS	file to Unix	mread(1)
ARPANET	file transfer program	ftp(1N)
trivial	file transfer program	tftp(1M)
DARPA Internet	File Transfer Protocol server	ftpd(1M)
DARPA Trivial	File Transfer Protocol server	tftpd(1M)
the scheduler for the uucp	file transport program	uusched(1M)
	file transport program for the uucp system	uucico(1M)
walk a	file tree	ftw(3C)
Motorola enhanced	file tree walk	motftw(3C)
determine	file type	file(1)
utilities permissions	file used by the value-added disk access	filesys(4)
synchronize a	file's in-core state with that on disk	fsync(2)
search and print process accounting	file(s)	acctcom(1)
set	file-creation mode mask	umask(1)
	file-system-table	fstab(4)
format of compiled term	file.	term(4)
merge an a.out file into another a.out	file.	scsiload(1M)
file header for common object/	filehdr	filehdr(4)
find device	filename from major and minor numbers	getdevname(1M)
stream status inquiries	fileno	ferror(3S)
compare two	files	cmp(1)
compress and expand	files	pack(1)
concatenate and print	files	cat(1)
copy, link, or move	files	cp(1)
create and administer SCCS	files	admin(1)
display character strings in object	files	strings(1)
displays or alters byte content of	files	bpatch(1)
file header for common object	files	filehdr(4)
find	files	find(1)
format of manual page translation	files	man(4)
format specification in text	files	fspec(4)
generate cyclic redundancy checksums (crc) of	files	crc(1)
identify SCCS	files	what(1)
introduction to special	files	intro(7)
link editor for common object	files	ld(1)
merge or add total accounting	files	acctmerg(1M)
move or rename	files	mv(1)
print section sizes in bytes of common object	files	size(1)
print	files	pr(1)
record locking on	files	lockf(3C)
select or reject lines common to two sorted	files	comm(1)
sort and/or merge	files	sort(1)
link and unlink	files and directories	link(1M)
back up and restore	files and/or file systems	br(1M)
recover	files from a backup tape	frec(1M)
install object	files in binary directories	cpset(1M)
password	files management	passmgmt(1M)
remove	files or directories	rm(1)
merge same lines of several	files or subsequent lines of one file	paste(1)
	files that hold object archi ecture information	ott(4)
convert archive	files to common formats	convert(1)
permissions file used by the/	filesys	filesys(4)
line numbering	filter	nl(1)

select terminal	filter	greek(1)
file perusal	filter for crt viewing	more(1)
file perusal	filter for CRTs	pg(1)
	filter reverse linefeeds	col(1)
administer	filters used with the LP print service	lpfilter(1M)
fast incremental backup	finc	finc(1M)
find files	find	find(1)
numbers	find device filename from major and minor	getdevname(1M)
	find files	find(1)
	find name of a terminal	ttyname(3C)
	find ordering relation for an object library	lorder(1)
	find spelling errors	spell(1)
user	find the slot in the utmp file of the current	ttyslot(3C)
C Obscure Feature	Finder	coff(1)
user information lookup program	finger	finger(1)
remote user information server	fingerd	fingerd(1M)
fold long lines for	finite width output device	fold(1)
database subroutines	firstkey	dbm(3N)
change the qwbd daemon debug trace	flags	qwbdflags(1M)
convert	floating-point number to string	ecvt(3C)
manipulate parts of	floating-point numbers	frexp(3C)
floor, ceiling, remainder,/	floor	floor(3M)
functions	floor, ceiling, remainder, absolute value	floor(3M)
format a MSDOS	floppy disk	mformat(1)
generate C	flowgraph	cflow(1)
close or	flush a stream	fclose(3S)
floor, ceiling, remainder,/	fmod	floor(3M)
disk initializer	fmt	fmt(1)
fold long lines for finite/	fold	fold(1)
	fold long lines for finite width output device	fold(1)
open a stream	fopen	fopen(3S)
	force compliance with a bill of materials	mkcomply(1M)
	force propagation of a changed NIS map	yppush(1M)
create a new process	fork	fork(2)
common archive file	format	ar(4)
common object file symbol table	format	syms(4)
entries and put in a file system independent	format read directory	getdents(2)
error-log file	format	errfile(4)
per-process accounting file	format	acct(4)
	format a MSDOS floppy disk	mformat(1)
compare common object	format modules	coffcmp(1)
dump common object	format modules	coffdump(1)
change the	format of a text file	newform(1)
	format of an inode	inode(4)
	format of compiled term file.	term(4)
	format of core image file	core(4)
	format of cpio archive	cpio(4)
	format of curses screen image file	scr_dump(4)
	format of directories	dir(4)
	format of manual page translation files	man(4)
	format of SCCS file	sccsfile(4)
	format of system volume	fs(4)
	format of time for system-wide realtime timers	timestruc(4)
timers	format of time values for process interval	itimerstruc(4)
	format specification in text files	fspec(4)
convert bad track list from one	format to another	xformtrk(1M)
convert archive files to common	formats	convert(1)

introduction to file	formats	intro(4)
introduction to networking file	formats	intro(4N)
utmp and wtmp entry	formats	utmp(4)
convert	formatted input	scanf(3S)
print	formatted output	printf(3S)
print	formatted output of a varargs argument list	vprintf(3S)
generating/modifying ASSIST menus or command	forms program for	astgen(1)
administer	forms used with the LP print service	lpforms(1M)
get configurable pathname/	fpathconf	pathconf(2P)
print formatted output	fprintf	printf(3S)
put character or word on a/	fputc	putc(3S)
put a string on a stream	fputs	puts(3S)
utilities alias file for	framed access command environment (FACE)	alias(4)
system-wide	framed access command environment variables	env(4)
binary input/output	fread	fread(3S)
recover files from a backup tape	frec	frec(1M)
fast main memory allocator	free	malloc(3C)
main memory allocator	free	malloc(3X)
	free a binary semaphore	bsfree(2)
	free a library structure	t_free(3N)
report number of	free disk blocks and i-nodes	df(1M)
I/O from the process	free internal resources for asynchronous	arwfree(2)
open a stream	freopen	fopen(3S)
manipulate parts of/	frexp	frexp(3C)
who is my mail from?	from	from(1)
who is my mail	from?	from(1)
construct a file system	fs	fs(1)
format of system volume	fs	fs(4)
file system block analyzer	fsba	fsba(1M)
convert formatted input	fscanf	scanf(3S)
check and repair file systems	fsck	fsck(1M)
make a lost+found directory for	fsck	mklost+found(1M)
list of file systems processed by	fsck and ncheck	checklist(4)
file system debugger	fsdb	fsdb(1M)
reposition a file pointer in a/	fseek	fseek(3S)
print various file system/	fsinfo	fsinfo(1M)
install random inode generation/	fsirand	fsirand(1M)
format specification in text/	fspec	fspec(4)
report file system status	fsstat	fsstat(1M)
file-system-table	fstab	fstab(4)
get file status	fstat	stat(2)
get file system information	fstatfs	statfs(2)
display file system name	fstyp	fstyp(1M)
synchronize a file's in-core/	fsync	fsync(2)
reposition a file pointer in a/	ftell	fseek(3S)
create IPC key from file name	ftok	ftok(3C)
standard interprocess/	ftok	stdipc(3C)
ARPANET file transfer program	ftp	ftp(1N)
DARPA Internet File Transfer/	ftpd	ftpd(1M)
truncate file space	ftruncate	ftruncate(2)
walk a file tree	ftw	ftw(3C)
search a file for a pattern using	full regular expressions	egrep(1)
shut down part of a	full-duplex connection	shutdown(2N)
Euclidean distance	function	hypot(3M)
error function and complementary error	function	erf(3M)
error-handling	function	matherr(3M)
line number entries of a common object file	function manipulate	ldlread(3X)

log gamma function ..... gamma(3M)
profile within a function ..... prof(5)
error function and complementary error function ... erf(3M)
Bessel functions ..... bessel(3M)
exponential, logarithm, power, square root functions ..... exp(3M)
floor, ceiling, remainder, absolute value functions ..... floor(3M)
hyperbolic functions ..... sinh(3M)
line control functions ..... tcdrain(3P)
machine-specific functions ..... sysmot(2)
password and file encryption functions ..... crypt(3X)
trigonometric functions ..... trig(3M)
various REAL/IX specific functions ..... sysrealix(2)
math functions and constants ..... math(5)
introduction to functions and libraries ..... intro(3)
introduction to functions and libraries ..... intro(3N)
handle special functions of DASI 300 and 300s terminals .... 300(1)
handle special functions of the DASI 450 terminal ..... 450(1)
identify processes using a file/ fuser ..... fuser(1M)
binary input/output fwrite ..... fread(3S)
manipulate connect accounting/ fwtmp ..... fwtmp(1M)
log gamma function gamma ..... gamma(3M)
log gamma function ..... gamma(3M)
get boot inode gbootinod ..... gbootinod(1M)
GNU project C Compiler gcc ..... gcc(1)
convert floating-point number/ gcvt ..... ecvt(3C)
used by REAL/IX realtime SCSI/ general 9-track tape implementation details .. scsi9track(7)
Operating System general non-volatile RAM driver for the REAL/IX nvram(7)
general ramdisk driver for the REAL/IX System ramdisk(7)
REAL/IX realtime SCSI autoconfiguration/ general tape implementation details used by .. scsitape(7)
general terminal interface ..... termio(7)
general terminal interface ..... termios(7)
generate an IOT fault ..... abort(3C)
generate C flowgraph ..... cflow(1)
generate C program cross-reference ..... cxref(1)
generate C program makefile dependencies ... depend(1)
conversion tables generate character classification and ..... chrtbl(1M)
files generate cyclic redundancy checksums (crc) of crc(1)
generate disk accounting data by user ID .... diskusg(1M)
generate encryption key ..... makekey(1)
generate file name for terminal ..... ctermid(3S)
generate hashing encryption ..... crypt(3C)
generate path names from i-numbers ..... ncheck(1M)
generate programs for simple lexical tasks .... lex(1)
configure and generate the operating system ..... sysgen(1M)
numbers generate uniformly distributed pseudo-random drand48(3C)
forms program for generating/modifying ASSIST menus or command astgen(1)
software management package generation facility ..... igf(1M)
install random inode generation numbers ..... fsirand(1M)
kernel debugger symbol table generator ..... unixsyms(1)
simple random-number generator ..... rand(3C)
GNU project parser generator (yacc replacement) ..... bison(1)
get a version of an SCCS file get ..... get(1)
get/set name of current host ..... gethostname(3N)
get/set name of current NIS domain ..... getdomainname(3N)
get character or word from a/ getc ..... getc(3S)
get character or word from a/ getchar ..... getc(3S)
return some hardware and/ getcpudata ..... getcpudata(2)

get path-name of current/	getcwd	getcwd(3C)
read directory entries and put/	getdents	getdents(2)
find device filename from major/	getdevname	getdevname(1M)
get/set name of current NIS/	getdomainname	getdomainname(3N)
get real user, effective user,/	getegid	getuid(2)
return value for environment/	getenv	getenv(3C)
get real user, effective user,/	geteuid	getuid(2)
get real user, effective user,/	getgid	getuid(2)
get group file entry	getgrent	getgrent(3C)
get group file entry	getgrgid	getgrent(3C)
get group file entry	getgrnam	getgrent(3C)
get group access list	getgroups	getgroups(3P)
get network host entry	gethostbyaddr	gethostent(3N)
get network host entry	gethostbyname	gethostent(3N)
get network host entry	gethostent	gethostent(3N)
get unique identifier of/	gethostid	gethostid(3N)
get/set name of current host	gethostname	gethostname(3N)
get the current value for a/	getinterval	getinterval(2)
get value of interval timer	getitimer	getitimer(2)
get login name	getlogin	getlogin(3C)
get next message off a stream	getmsg	getmsg(2)
get network entry	getnetbyaddr	getnetent(3N)
get network entry	getnetbyname	getnetent(3N)
get network entry	getnetent	getnetent(3N)
get network group entry	getnetgrent	getnetgrent(3N)
calculate an integer value from/	getnum	getnum(3X)
get option letter from argument/	getopt	getopt(3C)
parse command options	getopt	getopt(1)
parse command options	getoptcvt	getopts(1)
parse command options	getopts	getopts(1)
read a password	getpass	getpass(3C)
get name of connected peer	getpeername	getpeername(2N)
gets name of connected peer	getpeername	getpeername(3N)
read the permissions file	getperms	getpermsintro(3X2)
get process, process group, and/	getpgrp	getpid(2)
get process, process group, and/	getpid	getpid(2)
get process, process group, and/	getppid	getpid(2)
get scheduling priority	getpri	getpri(2)
get protocol entry	getprotobyname	getprotoent(3N)
get protocol entry	getprotobynumber	getprotoent(3N)
get protocol entry	getprotoent	getprotoent(3N)
set/get Processor Status/	getpsr	setpsr(2)
get name from UID	getpw	getpw(3C)
get password file entry	getpwent	getpwent(3C)
get password file entry	getpwnam	getpwent(3C)
get password file entry	getpwuid	getpwent(3C)
get RPC entry	getrpcbyname	getrpcent(3N)
get RPC entry	getrpcbynumber	getrpcent(3N)
get RPC entry	getrpcent	getrpcent(3N)
get RPC port number	getrpcport	getrpcport(3N)
get a string from a stream	gets	gets(3S)
	gets name of connected peer	getpeername(3N)
get service entry	getservbyname	getservent(3N)
get service entry	getservbyport	getservent(3N)
get service entry	getservent	getservent(3N)
get socket name	getsockname	getsockname(2N)
get and set options on sockets	getsockopt	getsockopt(2N)

get shadow password file entry — getspent ..... getspent(3X)
get shadow password file entry — getspnam ..... getspent(3X)
return some systemwide hardware/ — getsysdata ..... getsysdata(2)
fetch system load — getsysload ..... getsysload(2)
get time of day — gettimeofday ..... gettimeofday(3C)
get the current value for a/ — gettimer ..... gettimer(2)
get a unique identifier for a/ — gettimerid ..... gettimerid(2)
set terminal type, modes,/ — getty ..... getty(1M)
speed and terminal settings used by — getty ..... gettydefs(4)
spawn — getty to a remote terminal ..... ct(1C)
speed and terminal settings/ — gettydefs ..... gettydefs(4)
get real user, effective user,/ — getuid ..... getuid(2)
access utmp file entry — getutent ..... getut(3C)
access utmp file entry — getutid ..... getut(3C)
access utmp file entry — getutline ..... getut(3C)
get character or word from a/ — getw ..... getc(3S)
— give first few lines ..... head(1)
voluntarily — give up CPU ..... relinquish(2)
the real device in the permissions file for a — given alias echo ..... real(1)
definitions of terms and symbols — glossary ..... glossary(1)
convert date and time to string — gmtime ..... ctime(3C)
— GNU project C Compiler ..... gcc(1)
— GNU project parser generator (yacc replacement) bison(1)
GNU project C Compiler — gnucc ..... gcc(1)
GNU project C Compiler — gnugcc ..... gcc(1)
non-local — goto ..... setjmp(3C)
display call graph profile data — gprof ..... gprof(1)
display call — graph profile data ..... gprof(1)
select terminal filter — greek ..... greek(1)
search a file for a pattern — grep ..... grep(1)
change owner or — group ..... chown(1)
change — group ..... chgrp(1)
group file — group ..... group(4)
log in to a new — group ..... newgrp(1M)
set process — group ..... setpgid(2P)
get — group access list ..... getgroups(3P)
initialize — group access list ..... initgroups(3C)
set — group access list ..... setgroups(2)
get network — group entry ..... getnetgrent(3N)
— group file ..... group(4)
get — group file entry ..... getgrent(3C)
get distinguished process — group ID ..... tcgetpgrp(3P)
set distinguished process — group ID ..... tcsetpgrp(3P)
set process — group ID ..... setpgrp(2)
set process — group id and execute command ..... setpgrp(1)
set user and — group IDs ..... setuid(2)
change owner and — group of a file ..... chown(2)
send a signal to a process or a — group of processes ..... kill(2)
user, effective user, real group, and effective — group IDs get real ..... getuid(2)
print user and — group IDs and names ..... id(1M)
get real user, effective user, real — group, and effective group IDs ..... getuid(2)
get process, process — group, and parent process IDs ..... getpid(2)
list of network — groups ..... netgroup(4N)
set realtime privileged — groups list ..... setrtgroups(1M)
set realtime privileged — groups list ..... setrtgroups(2)
view the realtime privileged — groups list ..... rtgroups(1)
maintain, update, and regenerate — groups of programs ..... make(1)

password/group file checkers grpck ..... pwck(1M)
software signals gsignal ..... ssignal(3C)
shutdown the system with/ halt ..... halt(1M)
translation table to handle long map names ..... ypmapxlate(4N)
terminals handle special functions of DASI 300 and 300s 300(1)
terminal handle special functions of the DASI 450 ..... 450(1)
handle variable argument list ..... varargs(5)
create NFS client handles ..... nfsclnt(1M)
create kernel RPC client handles ..... kclt_create(3N)
terminal screen handling and optimization package ..... curses(3X)
run a command immune to hangups and quits ..... nohup(1)
return some systemwide hardware and software data ..... getsysdata(2)
return some hardware and software data on a particular cpu. getcpudata(2)
manage hash search tables ..... hsearch(3C)
find spelling errors hashcheck ..... spell(1)
generate hashing encryption ..... crypt(3C)
find spelling errors hashmake ..... spell(1)
manage hash search tables hcreate ..... hsearch(3C)
manage hash search tables hdestroy ..... hsearch(3C)
give first few lines head ..... head(1)
section header for a common object file ..... scnhdr(4)
file header for common object files ..... filehdr(4)
file header for implementation-specific constants . limits(4)
file header for symbolic constants ..... unistd(4)
read an indexed/named section header of a common object file ..... ldshread(3X)
read the file header of a common object file ..... ldfhread(3X)
seek to the optional file header of a common object file ..... ldohseek(3X)
read the archive header of a member of an archive file ..... ldahread(3X)
help facility help ..... help(1)
help facility ..... help(1)
make changes to the help facility database ..... helpadm(1M)
make changes to the help/ helpadm ..... helpadm(1M)
high resolution sleep ..... nanosleep(2)
summarize cache hit rate for each buffer size ..... bfree(1M)
files that hold object architecture information ..... ott(4)
hold serial lines open ..... port_hold(1M)
get unique identifier of current host ..... gethostid(3N)
get/set name of current host ..... gethostname(3N)
send network printing requests to a remote host ..... netlp(1N)
what version of a YP map is at a YP server host ..... yppoll(1M)
convert values between host and network byte order ..... byteorder(3N)
get network host entry ..... gethostent(3N)
which host is the NIS server or map master? ..... ypwhich(1N)
host name data base ..... hosts(4N)
host name lookup configuration file ..... svcorder(4N)
show host status of local machines ..... ruptime(1N)
set or print name of current host system ..... hostname(1N)
set or print name of current/ hostname ..... hostname(1N)
host name data base hosts ..... hosts(4N)
send ICMP ECHO_REQUEST packets to network hosts ..... ping(1M)
user-specified file of equivalent hosts and users ..... rhosts(4)
hosts names of hosts with equivalent users, i.e., "trusted" .... hosts.equiv(4N)
names of hosts with equivalent/ hosts.equiv ..... hosts.equiv(4N)
manage hash search tables hsearch ..... hsearch(3C)
convert values between host and/ htonl ..... byteorder(3N)
convert values between host and/ htons ..... byteorder(3N)
hyperbolic functions ..... sinh(3M)

Euclidean distance function hypot ..... hypot(3M)
clear i-node ..... clri(1M)
report number of free disk blocks and i-nodes ..... df(1M)
generate path names from i-numbers ..... ncheck(1M)
names of hosts with equivalent users, i.e., "trusted" hosts ..... hosts.equiv(4N)
synchronous I/O multiplexing ..... select(2)
control block for asynchronous I/O operations ..... aiocb(4)
cancel one or more asynchronous I/O requests ..... acancel(2)
get processor type truth value i386 ..... machid(1)
send ICMP ECHO_REQUEST packets to network hosts ping(1M)
summarize file size usage by/ icount ..... icount(1M)
generate disk accounting data by user ID ..... diskusg(1M)
get distinguished process group ID ..... tcgetpgrp(3P)
nessage queue, semaphore set, or shared memory ID remove a ..... ipcrm(1)
print effective current user id ..... whoami(1)
print user and group/ id ..... id(1M)
set distinguished process group ID ..... tcsetpgrp(3P)
set process group ID ..... setpgrp(2)
set process group id and execute command ..... setpgrp(1)
map a name to a corresponding id in a shared memory name server database . nsntoi(3C)
name server database map an id to a corresponding name in a shared memory nsiton(3C)
issue identification file ..... issue(4)
get an event identifier ..... evget(2)
get shared memory segment identifier ..... shmget(2)
release a process interval timer identifier ..... reltimerid(2)
release an event identifier ..... evrel(2)
get a unique identifier for a process interval timer ..... gettimerid(2)
get unique identifier of current host ..... gethostid(3N)
identify a command using keywords ..... locate(1)
structure identify processes using a file or file ..... fuser(1M)
identify SCCS files ..... what(1)
set user and group IDs ..... setuid(2)
network interface configuration ifconfig ..... ifconfig(1M)
excersise control over IP/ ifctl ..... ifctl(1M)
remove ifdef'ed lines ..... unifdef(1)
Network Interface Driver for/ ifen596 ..... ifen596(7N)
Network Interface Probe ifprobe ..... ifprobe(1M)
display statistics for/ ifstat ..... ifstat(1M)
software management package/ igf ..... igf(1M)
format of core image file ..... core(4)
format of curses screen image file ..... scr_dump(4)
save the core image of a system dump in a file ..... savecore(1M)
db command to dump system memory image to dump partition ..... sysdump(1M)
examine system images ..... crash(1M)
run a command immune to hangups and quits ..... nohup(1)
SCSI autoconfiguration drivers general tape implementation details used by REAL/IX realtime scsitape(7)
SCSI autoconfiguration/ general 9-track tape implementation details used by REAL/IX realtime scsi9track(7)
implementation-defined system call ..... sys_local(2)
file header for implementation-specific constants ..... limits(4)
synchronize a file's in-core state with that on disk ..... fsync(2)
set the expiration time of a/ incinterval ..... absinterval(2)
log incoming SSP maintenance bus async messages. mblogger(1M)
print out incoming SSP maintenance bus async messages. mba(1M)
fast incremental backup ..... finc(1M)
file system independent directory entry ..... dirent(4)
read directory entries and put in a file system independent format ..... getdents(2)
object file compute the index of a symbol table entry of a common ... ldtbindex(3X)

file read an indexed symbol table entry of a common object ldtbread(3X)
file read an indexed/named section header of a common object ldshread(3X)
seek to an indexed/named section of a common object file ldsseek(3X)
acknowledge receipt of an orderly release indication ..... t_rcvrel(3N)
receive a unit data error indication ..... t_rcvuderr(3N)
Internet protocol family inet ..... inet(7N)
internet super-server inetd ..... inetd(1M)
Internet address manipulation/ inet_addr ..... inet(3N)
Internet address manipulation/ inet_lnaof ..... inet(3N)
Internet address manipulation/ inet_makeaddr ..... inet(3N)
Internet address manipulation/ inet_netof ..... inet(3N)
Internet address manipulation/ inet_network ..... inet(3N)
Internet address manipulation/ inet_ntoa ..... inet(3N)
compare or print out terminfo/ infocmp ..... infocmp(1M)
files that hold object architecture information ..... ott(4)
get file system type information ..... sysfs(2)
get file system information ..... statfs(2)
get protocol-specific service information ..... t_getinfo(3N)
print CPU board information ..... cputype(1)
print various file system information ..... fsinfo(1M)
report RPC information ..... rpcinfo(1M)
service print information about the status of the LP print . lpstat(1)
users information about the system for beginning .. starter(1)
relocation information for a common object file ..... reloc(4)
install and update /etc/shadow with information from /etc/passwd ..... pwconv(1M)
strip symbol and line number information from a common object file ..... strip(1)
retrieve information from disconnect ..... t_rcvdis(3N)
user information lookup program ..... finger(1)
disk definition information manager ..... ddefs(1M)
remote user information server ..... fingerd(1M)
process control initialization init ..... init(1M)
script for the init process ..... inittab(4)
initialize group access list initgroups ..... initgroups(3C)
QWB multiprocessor system initialization ..... si(1M)
multiprocessor process control initialization ..... mpinit(1M)
process control initialization ..... init(1M)
initialization file for the qwbd daemons. ..... qwbtab(4)
system initialization procedures ..... bcheckrc(1M)
database initialize a terminal or query terminfo ..... tput(1)
initialize group access list ..... initgroups(3C)
asynchronous read initialize structures before requesting an ..... arinit(2)
asynchronous write initialize structures before requesting an ..... awinit(2)
initialize system for first user ..... setup(1M)
disk initializer ..... dinit(1M)
disk initializer ..... fmt(1)
initiate a connection on a socket ..... connect(2N)
initiate an orderly release ..... t_sndrel(3N)
initiate pipe to/from a process ..... popen(3S)
script for the init process inittab ..... inittab(4)
get network group entry innetgr ..... getnetgrent(3N)
format of an inode inode ..... inode(4)
format of an inode ..... inode(4)
get boot inode ..... gbootinod(1M)
install random inode generation numbers ..... fsirand(1M)
convert formatted input ..... scanf(3S)
get or set the value of the output and input baud rate ..... cfgetospeed(3P)
push character back into input stream ..... ungetc(3S)

binary	input/output	fread(3S)
STREAMS	input/output multiplexing	poll(2)
standard buffered	input/output package	stdio(3S)
stream status	inquiries	ferror(3S)
uucp status	inquiry and job control	uustat(1C)
server database	insert a record into a shared memory name	nsinsert(3C)
install commands	install	install(1M)
from /etc/passwd	install and update /etc/shadow with information	pwconv(1M)
	install commands	install(1M)
build and	install NIS database	ypinit(1M)
	install object files in binary directories	cpset(1M)
	install random inode generation numbers	fsirand(1M)
decode semaphore processor	instructions	smp(1M)
convert string to	integer	strtol(3C)
return	integer absolute value	abs(3C)
convert between long	integer and base-64 ASCII string	a64l(3C)
access long	integer data in a machine-independent fashion	sputl(3X)
calculate an	integer value from a string of characters	getnum(3X)
convert between 3-byte integers and long	integers	l3tol(3C)
convert between 3-byte	integers and long integers	l3tol(3C)
driver for the MODCOMP VLAN-E	Intelligent Ethernet Controller	vlan-e(7)
report	inter-process communication facilities status	ipcs(1)
	interactive message processing system	mailx(1)
file system check and	interactive repair	chk(1)
communication controller VME module	interface	mvme332XT
/ME350 Streamer Tape Controller VME Module	Interface	mvme350(7)
NIS client	interface	ypclnt(3N)
Quadbus console	interface	qwbcon(7)
SSP maintenance bus	interface	sspmb(7)
autoconfig MVME187 SCSI controller	interface	mvme187(7)
autoconfig MVME327 SCSI controller	interface	mvme327(7)
connected interrupt	interface	cintrio(7)
console	interface	console(7)
controlling terminal	interface	tty(7)
general terminal	interface	termio(7)
general terminal	interface	termios(7)
line printer	interface	lp(7)
software loopback network	interface	lo(7)
swap administrative	interface	swap(1M)
network	interface configuration	ifconfig(1M)
Transport	Interface cooperating STREAMS module	timod(7)
Transport Provider	Interface daemon	tpid(1M)
Network	Interface Driver for MVME187 Boards	ifen596(7N)
Network	Interface Probe	ifprobe(1M)
Transport	Interface read/write interface STREAMS module	tirdwr(7)
Transport Interface read/write	interface STREAMS module	tirdwr(7)
menu	interface to do system administration	sysadm(1M)
Quadbus device driver	interface to SSP devices	qwbsys(7)
tracing	interface to STREAMS error logging and event	log(7)
SSP device driver	interface to the MP system	qwbserv(7)
SSP	interface to the multiprocessor system debugger	nmi(1M)
QuadBus system	interface to the SSP monitor	sspmon(1M)
user	interface to the TELNET protocol	telnet(1N)
display statistics for supported IP	interfaces	ifstat(1M)
excersise control over IP network	interfaces	ifctl(1M)
I/O from the process free	internal resources for asynchronous	arwfree(2)
	Internet address manipulation routines	inet(3N)

	Internet Boot Protocol server	()
DARPA	Internet File Transfer Protocol server	ftpd(1M)
	Internet IP protocol streams multiplexer module	ip(7N)
	Internet protocol family	inet(7N)
	internet super-server	inetd(1M)
	Internet Transmission Control Protocol	tcp(7)
	Internet User Datagram Protocol	udp(7)
	interpret ASA carriage control characters	asa(1)
create an	interprocess channel	pipe(2)
standard	interprocess communication package	stdipc(3C)
wait for a connected	interrupt	cisema(2)
connected	interrupt interface	cintrio(7)
multiprocessor	interrupt routing	intrctl(2)
connected	interrupt structure	cintrio(4)
suspend execution for an	interval	sleep(1)
suspend execution for	interval	sleep(3C)
get a unique identifier for a process	interval timer	gettimerid(2)
get the current value for a process	interval timer	getinterval(2)
get value of	interval timer	getitimer(2)
resolution and maximum time value of process	interval timer get	resabs(2)
set the expiration time of a process	interval timer	absinterval(2)
set value of	interval timer	setitimer(2)
release a process	interval timer identifier	reltimerid(2)
format of time values for process	interval timers	itimerstruc(4)
multiprocessor interrupt routing	intrctl	intrctl(2)
introduction to commands and/	intro	intro(1)
introduction to commands and/	intro	intro(1N)
introduction to file formats	intro	intro(4)
introduction to functions and/	intro	intro(3)
introduction to functions and/	intro	intro(3N)
introduction to maintenance/	intro	intro(1M)
introduction to miscellany	intro	intro(5)
introduction to networking file/	intro	intro(4N)
introduction to realtime/	intro	intro(1R)
introduction to special files	intro	intro(7)
introduction to system calls/	intro	intro(2)
programs	introduction to commands and application	intro(1)
programs	introduction to commands and application	intro(1N)
	introduction to file formats	intro(4)
	introduction to functions and libraries	intro(3)
	introduction to functions and libraries	intro(3N)
application programs	introduction to maintenance commands and	intro(1M)
	introduction to miscellany	intro(5)
	introduction to networking file formats	intro(4N)
	introduction to realtime commands	intro(1R)
	introduction to special files	intro(7)
	introduction to system calls and error numbers	intro(2)
control device	ioctl	ioctl(2)
STREAMS	ioctl commands	streamio(7)
	ioctl requests for sockets	sockio(7N)
generate an	IOT fault	abort(3C)
Internet IP protocol streams/	ip	ip(7N)
display statistics for supported	IP interfaces	ifstat(1M)
excersise control over	IP network interfaces	ifctl(1M)
bind a socket to a privileged	IP port	bindresvport(3N)
Internet	IP protocol streams multiplexer module	ip(7N)
create	IPC key from file name	ftok(3C)

remove a message queue,/	ipcrm	ipcrm(1)
report inter-process/	ipcs	ipcs(1)
character classification macros	isalnum	ctype(3C)
character classification macros	isalpha	ctype(3C)
character classification macros	isascii	ctype(3C)
find name of a terminal	isatty	ttyname(3C)
character classification macros	iscntrl	ctype(3C)
character classification macros	isdigit	ctype(3C)
character classification macros	isgraph	ctype(3C)
character classification macros	islower	ctype(3C)
character classification macros	isprint	ctype(3C)
character classification macros	ispunct	ctype(3C)
character classification macros	isspace	ctype(3C)
issue identification file	issue	issue(4)
	issue a shell command	system(3S)
	issue identification file	issue(4)
character classification macros	isupper	ctype(3C)
character classification macros	isxdigit	ctype(3C)
print news	items	news(1)
format of time values for/	itimerstruc	itimerstruc(4)
software management package/	ixf	ixf(1M)
Bessel functions	j0	bessel(3M)
Bessel functions	j1	bessel(3M)
Bessel functions	jn	bessel(3M)
uucp status inquiry and	job control	uustat(1C)
relational database operator	join	join(1)
generate uniformly distributed/	jrand48	drand48(3C)
non-local	jumps	sigsetjmp(3P)
create kernel RPC client handles	kclt_create	kclt_create(3N)
kernel debugger	kdb	kdb(1M)
	keep track of remotely mounted file systems	mount(3N)
	kernel console printf logger	putbuf(1)
save	kernel core dumps	savedump(1M)
	kernel debugger	kdb(1M)
	kernel debugger symbol table generator	unixsyms(1)
start, stop, sample, or reset	kernel profiling	kgmon(1M)
create	kernel RPC client handles	kclt_create(3N)
generate encryption	key	makekey(1)
create IPC	key from file name	ftok(3C)
print the value of one or more	keys from a NIS map	ypmatch(1N)
identify a command using	keywords	locate(1)
start, stop, sample, or reset/	kgmon	kgmon(1M)
send a signal to a process or a/	kill	kill(2)
terminate a process	kill	kill(1)
	kill all active processes	killall(1M)
kill all active processes	killall	killall(1M)
core memory	kmem	mem(7)
shell, the standard/restricted/	ksh	ksh(1)
shell, the standard/restricted/	ksh	sh(1)
convert between 3-byte integers/	l3tol	l3tol(3C)
convert between long integer/	l64a	a64l(3C)
provide labels for file systems	labelit	labelit(1M)
provide	labels for file systems	labelit(1M)
extracts menus from	labels stored in command shell scripts	mkmenus(1M)
arbitrary-precision arithmetic	language	bc(1)
pattern scanning and processing	language	awk(1)
pattern scanning and processing	language	oawk(1)

preprocess MC88100 assembly	language	sifilter(1)
the bourne shell command programming	language	bsh(1)
the standard/restricted command and programming	language shell,	sh(1)
the standard/restricted command programming	language shell,	ksh(1)
the C	language preprocessor	cpp(1)
and dates to ASCII	language specific strings for converting times	cftime(4)
shell procedures for accounting	lastlogin	acctsh(1M)
execute commands at a	later time	at(1)
shell	layer manager	shl(1)
pseudo shell	layer, console manager	conman(1)
change owner and group of a file	lchown	chown(2)
get shadow password file entry	lckpwdf	getspent(3X)
generate uniformly distributed/	lcong48	drand48(3C)
link editor for common object/	ld	ld(1)
close a common object file	ldaclose	ldclose(3X)
read the archive header of a/	ldahread	ldahread(3X)
open a common object file for/	ldaopen	ldopen(3X)
close a common object file	ldclose	ldclose(3X)
manipulate parts of/	ldexp	frexp(3C)
common object file access/	ldfcn	ldfcn(4)
read the file header of a/	ldfhread	ldfhread(3X)
retrieve symbol name for common/	ldgetname	ldgetname(3X)
manipulate line number entries/	ldlinit	ldlread(3X)
manipulate line number entries/	ldlitem	ldlread(3X)
manipulate line number entries/	ldlread	ldlread(3X)
seek to line number entries of/	ldlseek	ldlseek(3X)
seek to line number entries of/	ldnlseek	ldlseek(3X)
seek to relocation entries of a/	ldnrseek	ldrseek(3X)
read an indexed/named section/	ldnshread	ldshread(3X)
seek to an indexed/named/	ldnsseek	ldsseek(3X)
seek to the optional file/	ldohseek	ldohseek(3X)
open a common object file for/	ldopen	ldopen(3X)
seek to relocation entries of a/	ldrseek	ldrseek(3X)
read an indexed/named section/	ldshread	ldshread(3X)
seek to an indexed/named/	ldsseek	ldsseek(3X)
compute the index of a symbol/	ldtbindex	ldtbindex(3X)
read an indexed symbol table/	ldtbread	ldtbread(3X)
seek to the symbol table of a/	ldtbseek	ldtbseek(3X)
get option	letter from argument vector	getopt(3C)
generate programs for simple/	lex	lex(1)
generate programs for simple	lexical tasks	lex(1)
linear search and update	lfind	lsearch(3C)
introduction to functions and	libraries	intro(3)
introduction to functions and	libraries	intro(3N)
create a shared	library	mkshlib(1)
find ordering relation for an object	library	lorder(1)
synchronize transport	library	t_sync(3N)
archive and	library maintainer for portable archives	ar(1)
representation	library routines for external data	xdr(3N)
	library routines for remote procedure calls	rpc(3N)
allocate a	library structure	t_alloc(3N)
free a	library structure	t_free(3N)
file header for/	limits	limits(4)
get and set user	limits	ulimit(2)
read one line	line	line(1)
read one	line	line(1)
establish an out-going terminal	line connection	dial(3C)

	line control functions	tcdrain(3P)
set terminal type, modes, speed, and	line discipline	getty(1M)
set terminal type, modes, speed, and	line discipline	uugetty(1M)
	line number entries in a common object file	linenum(4)
function manipulate	line number entries of a common object file	ldlread(3X)
object file seek to	line number entries of a section of a common	ldlseek(3X)
file strip symbol and	line number information from a common object	strip(1)
	line numbering filter	nl(1)
object file produce a C source listing with	line numbers from a common	list(1)
cut out selected fields of each	line of a file	cut(1)
	line printer interface	lp(7)
	linear search and update	lsearch(3C)
filter reverse	linefeeds	col(1)
line number entries in a common/	linenum	linenum(4)
give first few	lines	head(1)
remove ifdef'ed	lines	unifdef(1)
select or reject	lines common to two sorted files	comm(1)
fold long	lines for finite width output device	fold(1)
report repeated	lines in a file	uniq(1)
merge same lines of several files or subsequent	lines of one file	paste(1)
one file merge same	lines of several files or subsequent lines of	paste(1)
hold serial	lines open	port_hold(1M)
link and unlink files and/	link	link(1M)
link to a file	link	link(2)
read value of a symbolic	link	readlink(2)
	link and unlink files and directories	link(1M)
	link editor for common object files	ld(1)
common assembler and	link editor output	a.out(4)
	link to a file	link(2)
make symbolic	link to a file	symlink(2)
copy,	link, or move files	cp(1)
a C program checker	lint	lint(1)
from a common object file/	list	list(1)
get entries from name	list	nlist(3C)
get group access	list	getgroups(3P)
handle variable argument	list	varargs(5)
initialize group access	list	initgroups(3C)
print formatted output of a varargs argument	list	vprintf(3S)
receive any queued event from a specified	list	evrcvl(2)
set group access	list	setgroups(2)
set realtime privileged groups	list	setrtgroups(1M)
set realtime privileged groups	list	setrtgroups(2)
set realtime privileged users	list	setrtusers(1M)
set realtime privileged users	list	setrtusers(2)
view the realtime privileged groups	list	rtgroups(1)
view the realtime privileged users	list	rtusers(1)
	list contents of directory	ls(1)
system	list file names and statistics for a file	ff(1M)
convert bad track	list from one format to another	xformtrk(1M)
print name	list of common object file	nm(1)
ncheck	list of file systems processed by fsck and	checklist(4)
	list of network groups	netgroup(4N)
construct argument	list(s) and execute command	xargs(1)
listen for connections on a/	listen	listen(2N)
	listen for a connect request	t_listen(3N)
	listen for connections on a socket	listen(2N)
network	listener service administration	nlsadmin(1M)

from a common object file produce a C source	listing with line numbers	list(1)
make	literal copy of file system	volcopy(1M)
copy, link, or move files	ln	cp(1)
software loopback network/	lo	lo(7)
fetch system	load	getsysload(2)
bootstrap	loader for QWB multiprocessor	bo(1M)
show host status of	local machines	ruptime(1N)
who's logged in on	local machines	rwho(1N)
set or query current	locale	setlocale(3C)
convert date and time to string	localtime	ctime(3C)
identify a command using/	locate	locate(1)
last	locations in program	end(3C)
reserve a terminal	lock	lock(1)
	lock a binary semaphore	bslk(3C)
conditionally	lock a binary semaphore	bslkc(3C)
display	lock and semaphore statistics	semstat(1M)
	lock process, text, or data in memory	plock(2)
make	locked segments resident in memory	resident(2)
record locking on files	lockf	lockf(3C)
record	locking on files	lockf(3C)
exponential, logarithm, power,/	log	exp(3M)
interface to STREAMS error/	log	log(7)
	log gamma function	gamma(3M)
	log in to a new group	newgrp(1M)
messages.	log incoming SSP maintenance bus async	mblogger(1M)
	log of failed login attempts	loginlog(4)
exponential, logarithm, power,/	log10	exp(3M)
exponential,	logarithm, power, square root functions	exp(3M)
process a report of	logged errors	errpt(1M)
who's	logged in on local machines	rwho(1N)
kernel console printf	logger	putbuf(1)
STREAMS error	logger cleanup program	strclean(1M)
STREAMS error	logger daemon	strerr(1M)
interface to STREAMS error	logging and event tracing	log(7)
convert track numbers into file system	logical block numbers	fblkgen(1M)
remote	login	rlogin(1)
sign on	login	login(1)
log of failed	login attempts	loginlog(4)
get	login name	getlogin(3C)
get	login name	logname(1)
get character	login name of the user	cuserid(3S)
return	login name of user	logname(3X)
change	login password and password attributes	passwd(1)
change	login password in NIS	yppasswd(1N)
remote	login server	rlogind(1M)
setting up an environment at	login time	profile(4)
log of failed login attempts	loginlog	loginlog(4)
get login name	logname	logname(1)
return login name of user	logname	logname(3X)
non-local goto	longjmp	setjmp(3C)
endpoint	look at the current event on a transport	t_look(3N)
host name	lookup configuration file	svcorder(4N)
user information	lookup program	finger(1)
software	loopback network interface	lo(7)
perform a simple	loopback test on the SSP maintenance bus	mbloop(1M)
find ordering relation for an/	lorder	lorder(1)
make a	lost+found directory for fsck	mklost+found(1M)

run a command at	low priority	nice(1)
line printer interface	lp	lp(7)
send/cancel requests to an LP/	lp	lp(1)
administer filters used with the	LP print service	lpfilter(1M)
administer forms used with the	LP print service	lpforms(1M)
configure the	LP print service	lpadmin(1M)
print information about the status of the	LP print service	lpstat(1)
send/cancel requests to an	LP print service	lp(1)
start/stop the	LP print service and move requests	lpsched(1M)
enable/disable	LP printers	enable(1)
allow or prevent	LP requests	accept(1M)
line printer interface	lp147	lp(7)
line printer interface	lp187	lp(7)
configure the LP print service	lpadmin	lpadmin(1M)
administer filters used with/	lpfilter	lpfilter(1M)
administer forms used with the/	lpforms	lpforms(1M)
start/stop the LP print service/	lpmove	lpsched(1M)
start/stop the LP print service/	lpsched	lpsched(1M)
start/stop the LP print service/	lpshut	lpsched(1M)
print information about the/	lpstat	lpstat(1)
set printing queue priorities	lpusers	lpusers(1M)
generate uniformly distributed/	lrand48	drand48(3C)
list contents of directory	ls	ls(1)
linear search and update	lsearch	lsearch(3C)
move read/write file pointer	lseek	lseek(2)
get file status	lstat	stat(2)
convert between 3-byte integers/	ltol3	l3tol(3C)
MVME332XT control program	m332xctl	m332xctl(1M)
MVME336 control program	m336ctl	m336ctl(1M)
MVME350 control program	m350ctl	m350ctl(1M)
MVME355 control program	m355ctl	m355ctl(1M)
macro processor	m4	m4(1)
serial ports on the/	m681	m681(7)
get processor type truth value	m68k	machid(1)
get processor type truth value	m88k	machid(1)
	machine-dependent values	values(5)
access long integer data in a	machine-independent fashion	sputl(3X)
	machine-specific functions	sysmot(2)
show host status of local	machines	ruptime(1N)
who's logged in on local	machines	rwho(1N)
	macro processor	m4(1)
character classification	macros	ctype(3C)
	magnetic tape control	mt(1)
send mail to users or read mail	mail	mail(1)
send mail to users or read	mail	mail(1)
be notified if	mail arrives and who it is from	biff(1)
who is my	mail from?	from(1)
send	mail to users or read mail	mail(1)
interactive message processing/	mailx	mailx(1)
	main memory allocator	malloc(3X)
fast	main memory allocator	malloc(3C)
	maintain a tags file for a C program	ctags(1)
programs	maintain, update, and regenerate groups of	make(1)
archive and library	maintainer for portable archives	ar(1)
address resolution protocol	maintenance	arp(1M)
perform a simple loopback test on the SSP	maintenance bus	mbloop(1M)
perform one sync transfer on the SSP	maintenance bus	mbs(1M)

log incoming SSP	maintenance bus async messages.	mblogger(1M)
print out incoming SSP	maintenance bus async messages.	mba(1M)
SSP	maintenance bus interface	sspmb(7)
turn SSP	maintenance bus polling on or off	mbpoll(1M)
introduction to	maintenance commands and application programs	intro(1M)
find device filename from	major and minor numbers	getdevname(1M)
maintain, update, and/	make	make(1)
make an NIS dbm file	makedbm	makedbm(1M)
generate C program	makefile dependencies	depend(1)
create a file system on a/	makefsys	makefsys(1M)
generate encryption key	makekey	makekey(1)
fast main memory allocator	mallinfo	malloc(3C)
fast main memory allocator	malloc	malloc(3C)
main memory allocator	malloc	malloc(3X)
fast main memory allocator	mallopt	malloc(3C)
format of manual page/	man	man(4)
print entries in this manual	man	man(1)
	manage binary search trees	tsearch(3C)
	manage hash search tables	hsearch(3C)
	manage options for a transport endpoint	t_optmgmt(3N)
password files	management	passmgmt(1M)
signal	management	sigset(2)
password file	management for the NIS	yppassmgmt(1M)
software	management package extraction facility	ixf(1M)
software	management package generation facility	igf(1M)
QWB multiprocessor configuration database	manager	cf(1M)
disk definition information	manager	ddefs(1M)
pseudo shell layer, console	manager	conman(1)
shell layer	manager	shl(1)
	manipulate connect accounting records	fwtmp(1M)
object file function	manipulate line number entries of a common	ldlread(3X)
	manipulate parts of floating-point numbers	frexp(3C)
	manipulate signal sets	sigsetops(2P)
	manipulate the object file comment section	mcs(1)
manually	manipulate the routing tables	route(1M)
Internet address	manipulation routines	inet(3N)
read from file in an asynchronous	manner	aread(2)
write to file in an asynchronous	manner	awrite(2)
print entries in this	manual	man(1)
format of	manual page translation files	man(4)
	manually manipulate the routing tables	route(1M)
force propagation of a changed NIS	map	yppush(1M)
print the value of one or more keys from a NIS	map	ypmatch(1N)
memory name server database	map a name to a corresponding id in a shared	nsntoi(3C)
memory name server database	map an id to a corresponding name in a shared	nsiton(3C)
transfer an NIS	map from some NIS server to here	ypxfr(1M)
what version of a YP	map is at a YP server host	yppoll(1M)
which host is the NIS server or	map master?	ypwhich(1N)
translation table to handle long	map names	ypmapxlate(4N)
	map of ASCII character set	ascii(5)
	map process addresses to a file	mmap(2)
NIS database conversion routines	map2ascii	map2ascii(1M)
unmap previously	mapped addresses	munmap(2)
DARPA port to RPC program number	mapper	portmap(1M)
set and get file creation	mask	umask(2)
set file-creation mode	mask	umask(1)
which host is the NIS server or map	master?	ypwhich(1N)

regular expression compile and — match routines ..... regexp(5)
force compliance with a bill of — materials ..... mkcomply(1M)
math functions and constants — math ..... math(5)
— math functions and constants ..... math(5)
error-handling function — matherr ..... matherr(3M)
get resolution and — maximum time value of process interval timer resabs(2)
timer get the resolution and — maximum time values for a system-wide realtime restimer(2)
print out incoming SSP/ — mba ..... mba(1M)
log incoming SSP maintenance/ — mblogger ..... mblogger(1M)
perform a simple loopback test/ — mbloop ..... mbloop(1M)
turn SSP maintenance bus/ — mbpoll ..... mbpoll(1M)
perform one sync transfer on/ — mbs ..... mbs(1M)
preprocess — MC88100 assembly language ..... sifilter(1)
manipulate the object file/ — mcs ..... mcs(1)
delete a MSDOS file — mdel ..... mdel(1)
display a MSDOS directory — mdir ..... mdir(1)
control structures for the common event — mechanism ..... evt(4)
copy removeable — media ..... dcpy(1)
core memory — mem ..... mem(7)
read the archive header of a — member of an archive file ..... ldahread(3X)
memory operations — memccpy ..... memory(3C)
memory operations — memchr ..... memory(3C)
memory operations — memcmp ..... memory(3C)
memory operations — memcpy ..... memory(3C)
control write/execute/ — memctl ..... memctl(2)
clear QWB — memory ..... cm(1M)
control structures for shared — memory ..... shm(4)
control write/execute attributes of — memory ..... memctl(2)
core — memory ..... mem(7)
lock process, text, or data in — memory ..... plock(2)
make locked segments resident in — memory ..... resident(2)
fast main — memory allocator ..... malloc(3C)
main — memory allocator ..... malloc(3X)
shared — memory control operations ..... shmctl(2)
a message queue, semaphore set, or shared — memory ID remove ..... ipcrm(1)
db command to dump system — memory image to dump partition ..... sysdump(1M)
close a shared — memory name server database ..... nsclose(3C)
create a shared — memory name server database ..... nscreat(3C)
delete a record from a shared — memory name server database ..... nsdelete(3C)
get a record from a shared — memory name server database ..... nsget(3C)
insert a record into a shared — memory name server database ..... nsinsert(3C)
map a name to a corresponding id in a shared — memory name server database ..... nsntoi(3C)
map an id to a corresponding name in a shared — memory name server database ..... nsiton(3C)
open a shared — memory name server database ..... nsopen(3C)
remove a shared — memory name server database ..... nsremove(3C)
synchronise a — memory object ..... msync(2)
— memory operations ..... memory(3C)
shared — memory operations ..... shmop(2)
create shared — memory segment ..... shmget(1)
get shared — memory segment identifier ..... shmget(2)
memory operations — memset ..... memory(3C)
— menu interface to do system administration .. sysadm(1M)
scripts extracts — menus from labels stored in command shell .. mkmenus(1M)
program for generating/modifying ASSIST — menus or command forms ..... astgen(1)
— merge an a.out file into another a.out file. .... scsiload(1M)
sort and/or — merge files ..... sort(1)
— merge or add total accounting files ..... acctmerg(1M)

lines of one file	merge same lines of several files or subsequent	paste(1)
permit or deny messages	mesg	mesg(1)
produce error	message	t_error(3N)
	message control operations	msgctl(2)
get next	message off a stream	getmsg(2)
send a	message on a stream	putmsg(2)
interactive	message processing system	mailx(1)
get	message queue	msgget(2)
ID remove a	message queue, semaphore set, or shared memory	ipcrm(1)
control structures for	messages	msg(4)
permit or deny	messages	mesg(1)
print STREAMS trace	messages	strace(1M)
system error	messages	perror(3C)
send and receive	messages (message operations)	msgop(2)
log incoming SSP maintenance bus async	messages.	mblogger(1M)
print out incoming SSP maintenance bus async	messages.	mba(1M)
format a MSDOS floppy disk	mformat	mformat(1)
open any	minor device on a STREAMS driver	clone(7)
find device filename from major and	minor numbers	getdevname(1M)
overview of accounting and	miscellaneous accounting commands	acct(1M)
introduction to	miscellany	intro(5)
force compliance with a bill of/	mkcomply	mkcomply(1M)
make a directory	mkdir	mkdir(2)
make directories	mkdir	mkdir(1)
construct a file system	mkfs	mkfs(1M)
make a lost+found directory for/	mklost+found	mklost+found(1M)
extracts menus from labels/	mkmenus	mkmenus(1M)
build special file	mknod	mknod(1M)
make a directory, or a special/	mknod	mknod(2)
create a shared library	mkshlib	mkshlib(1)
make a unique file name	mktemp	mktemp(3C)
map process addresses to a file	mmap	mmap(2)
make a MSDOS sub directory	mmd	mmd(1)
mount and dismount file system	mnt	mnt(1)
mounted file system table	mnttab	mnttab(4)
driver for the	MODCOMP 4211 Peregrine FDDI Controller	fddi4211(7)
driver for the	MODCOMP VLAN-E Intelligent Ethernet Controller	vlan-e(7)
change	mode	chmod(1)
set file-creation	mode mask	umask(1)
change	mode of file	chmod(2)
null	modem daemon pair	nmd(1M)
set terminal type,	modes, speed, and line discipline	getty(1M)
set terminal type,	modes, speed, and line discipline	uugetty(1M)
manipulate parts of/	modf	frexp(3C)
set file access and	modification times	utime(2)
update access and	modification times of a file	touch(1)
server for	modifying NIS password file	yppasswdd(1M)
Interface read/write interface STREAMS	module Transport	tirdwr(7)
Internet IP protocol streams multiplexer	module	ip(7N)
Transport Interface cooperating STREAMS	module	timod(7)
MVME332XT communication controller VME	module interface	mvme332xt(7)
MVME350 Streamer Tape Controller VME	Module Interface	mvme350(7)
compare common object format	modules	coffcmp(1)
dump common object format	modules	coffdump(1)
shell procedures for accounting	monacct	acctsh(1M)
QuadBus system interface to the SSP	monitor	sspmon(1M)
prepare execution profile	monitor	monitor(3C)

Environment	Monitor Board driver	emb(7)
file perusal filter for crt/	more	more(1)
Motorola enhanced file tree walk	motftw	motftw(3C)
	Motorola enhanced file tree walk	motftw(3C)
keep track of remotely mounted/	mount	mount(3N)
mount a file system	mount	
mount a file system	mount	mount(2)
mount and unmount file systems/	mount	mount(1M)
	mount a file system	()
	mount a file system	mount(2)
	mount and dismount file system	mnt(1)
resources	mount and unmount file systems and remote	mount(1M)
automatically	mount NFS file systems	automount(1M)
NFS	mount request server	mountd(1M)
establish	mount table	setmnt(1M)
	mount, unmount a diskette file system	mountfsys(1M)
	mount, unmount multiple file systems	mountall(1M)
	mount, unmount multiple file systems	nmountall(1M)
mount, unmount multiple file/	mountall	mountall(1M)
NFS mount request server	mountd	mountd(1M)
	mounted file system table	mnttab(4)
NFS remotely	mounted file system table	rmtab(4N)
keep track of remotely	mounted file systems	mount(3N)
mount, unmount a diskette file/	mountfsys	mountfsys(1M)
show all remote	mounts	showmount(1M)
	move a directory	mvdir(1M)
copy, link, or	move files	cp(1)
	move or rename files	mv(1)
	move read/write file pointer	lseek(2)
start/stop the LP print service and	move requests	lpsched(1M)
SSP device driver interface to the	MP system	qwbserv(7)
multiprocessor process control/	mpinit	mpinit(1M)
generate uniformly distributed/	mrand48	drand48(3C)
remove a MSDOS sub directory	mrd	mrd(1)
read (copy) a MSDOS file to Unix	mread	mread(1)
rename an existing MSDOS file	mren	mren(1)
write (copy) a Unix file to	MSDOS	mwrite(1)
display a	MSDOS directory	mdir(1)
delete a	MSDOS file	mdel(1)
display contents of a	MSDOS file	mtype(1)
rename an existing	MSDOS file	mren(1)
read (copy) a	MSDOS file to Unix	mread(1)
format a	MSDOS floppy disk	mformat(1)
make a	MSDOS sub directory	mmd(1)
remove a	MSDOS sub directory	mrd(1)
control structures for messages	msg	msg(4)
message control operations	msgctl	msgctl(2)
get message queue	msgget	msgget(2)
send and receive messages/	msgrcv	msgop(2)
send and receive messages/	msgsnd	msgop(2)
slice table editor	msledit	sledit(1M)
synchronise a memory object	msync	msync(2)
magnetic tape control	mt	mt(1)
display contents of a MSDOS file	mtype	mtype(1)
run commands performed for	multi-user environment	rc2(1M)
mount, unmount	multiple file systems	mountall(1M)
mount, unmount	multiple file systems	nmountall(1M)

receive multiple queued events ..... evrcvm(2)
do multiple reads from a file ..... readv(2)
do multiple writes from a file ..... writev(2)
Internet IP protocol streams multiplexer module ..... ip(7N)
STREAMS input/output multiplexing ..... poll(2)
synchronous I/O multiplexing ..... select(2)
bootstrap loader for QWB multiprocessor ..... bo(1M)
of the software architecture of the Quadbus multiprocessor a brief overview ..... realstar(5)
QWB multiprocessor configuration database manager cf(1M)
multiprocessor interrupt routing ..... intrctl(2)
multiprocessor process control initialization .. mpinit(1M)
SSP interface to the multiprocessor system debugger ..... nmi(1M)
QWB multiprocessor system initialization ..... si(1M)
run commands performed for the SSP multiuser environment ..... rcqwb(1M)
unmap previously mapped/ munmap ..... munmap(2)
copy, link, or move files mv ..... cp(1)
move or rename files mv ..... mv(1)
move a directory mvdir ..... mvdir(1M)
serial ports on the MVME181/MVME188 ..... m681(7)
autoconfig MVME187 SCSI/ mvme187 ..... mvme187(7)
serial ports on the MVME187 ..... cd2401(7)
Network Interface Driver for MVME187 Boards ..... ifen596(7N)
autoconfig MVME187 SCSI controller interface ..... mvme187(7)
autoconfig MVME327 SCSI/ mvme327 ..... mvme327(7)
autoconfig MVME327 SCSI controller interface ..... mvme327(7)
MVME332XT communication/ mvme332xt ..... mvme332xt(7)
interface MVME332XT communication controller VME module mvme332xt(7)
MVME332XT control program ..... m332xctl(1M)
MVME336 control program ..... m336ctl(1M)
MVME350 Streamer Tape/ mvme350 ..... mvme350(7)
MVME350 control program ..... m350ctl(1M)
Interface MVME350 Streamer Tape Controller VME Module mvme350(7)
driver for the MVME355 9-Track/ mvme355 ..... mvme355(7)
driver for the MVME355 9-Track Tape Controller ..... mvme355(7)
MVME355 control program ..... m355ctl(1M)
write (copy) a Unix file to/ mwrite ..... mwrite(1)
create IPC key from file name ..... ftok(3C)
device name ..... devnm(1M)
display file system name ..... fstyp(1M)
get login name ..... getlogin(3C)
get login name ..... logname(1)
get socket name ..... getsockname(2N)
make a unique file name ..... mktemp(3C)
return value for environment name ..... getenv(3C)
working directory name ..... pwd(1)
host name data base ..... hosts(4N)
protocol name data base ..... protocols(4N)
service name data base ..... services(4N)
network name database ..... networks(4N)
create a name for a temporary file ..... tmpnam(3S)
retrieve symbol name for common object file symbol table entry ldgetname(3X)
generate file name for terminal ..... ctermid(3S)
get name from UID ..... getpw(3C)
map an id to a corresponding name in a shared memory name server database nsiton(3C)
get entries from name list ..... nlist(3C)
print name list of common object file ..... nm(1)
host name lookup configuration file ..... svcorder(4N)

change the	name of a file	rename(2)
find	name of a terminal	ttyname(3C)
get	name of connected peer	getpeername(2N)
gets	name of connected peer	getpeername(3N)
get/set	name of current host	gethostname(3N)
set or print	name of current host system	hostname(1N)
get/set	name of current NIS domain	getdomainname(3N)
set or display	name of current NIS domain	domainname(1N)
get	name of current system	uname(2)
print	name of current system	uname(1)
get the	name of the terminal	tty(1)
get character login	name of the user	cuserid(3S)
return login	name of user	logname(3X)
name to a corresponding id in a shared memory	name server database map	nsntoi(3C)
close a shared memory	name server database	nsclose(3C)
create a shared memory	name server database	nscreat(3C)
delete a record from a shared memory	name server database	nsdelete(3C)
get a record from a shared memory	name server database	nsget(3C)
id to a corresponding name in a shared memory	name server database map an	nsiton(3C)
insert a record into a shared memory	name server database	nsinsert(3C)
open a shared memory	name server database	nsopen(3C)
remove a shared memory	name server database	nsremove(3C)
name server database map a	name to a corresponding id in a shared memory	nsntoi(3C)
bind a	name to a socket	bind(2N)
print user and group IDs and	names	id(1M)
translation table to handle long map	names	ypmapxlate(4N)
list file	names and statistics for a file system	ff(1M)
conventional	names for terminals	term(5)
generate path	names from i-numbers	ncheck(1M)
high resolution sleep	nanosleep	nanosleep(2)
get resolution supported by	nanosleep(2)	nanosleep_getres(2)
get resolution supported by/	nanosleep_getres	nanosleep_getres(2)
generate path names from/	ncheck	ncheck(1M)
list of file systems processed by fsck and	ncheck	checklist(4)
network configuration program	netc	netc(1M)
list of network groups	netgroup	netgroup(4N)
send network printing requests/	netlp	netlp(1N)
network printer daemon	netlpd	netlpd(1N)
show network status	netstat	netstat(1M)
convert values between host and	network byte order	byteorder(3N)
	network configuration program	netc(1M)
get	network entry	getnetent(3N)
get	network group entry	getnetgrent(3N)
list of	network groups	netgroup(4N)
get	network host entry	gethostent(3N)
send ICMP ECHO_REQUEST packets to	network hosts	ping(1M)
software loopback	network interface	lo(7)
	network interface configuration	ifconfig(1M)
	Network Interface Driver for MVME187 Boards	ifen596(7N)
	Network Interface Probe	ifprobe(1M)
excersise control over IP	network interfaces	ifctl(1M)
	network listener service administration	nlsadmin(1M)
	network name database	networks(4N)
	network printer daemon	netlpd(1N)
send	network printing requests to a remote host	netlp(1N)
write to all users over a	network running the TCP/IP-based software	rwall(1M)
	network rwall server	rwalld(1M)

show	network status	netstat(1M)
introduction to	networking file formats	intro(4N)
network name database	networks	networks(4N)
change the format of a text file	newform	newform(1)
log in to a new group	newgrp	newgrp(1M)
print news items	news	news(1)
print	news items	news(1)
get	next message off a stream	getmsg(2)
database subroutines	nextkey	dbm(3N)
NFS, NIS, and REX start/stop/	nfs	nfs(1M)
create	NFS client handles	nfsclnt(1M)
directories to export to	NFS clients	exports(4N)
	NFS daemons	nfsd(1M)
automatically mount	NFS file systems	automount(1M)
	NFS mount request server	mountd(1M)
	NFS remotely mounted file system table	rmtab(4N)
	NFS statistics	nfsstat(1M)
scripts	NFS, NIS, and REX start/stop and configuration	nfs(1M)
NFS, NIS, and REX start/stop/	nfs.config	nfs(1M)
create NFS client handles	nfsclnt	nfsclnt(1M)
NFS daemons	nfsd	nfsd(1M)
NFS statistics	nfsstat	nfsstat(1M)
change priority of a process	nice	nice(2)
run a command at low priority	nice	nice(1)
change login password in	NIS	yppasswd(1N)
password file management for the	NIS	yppassmgmt(1M)
update user password in	NIS	yppasswd(3N)
	NIS client interface	ypclnt(3N)
build and install	NIS database	ypinit(1M)
print values in a	NIS database	ypcat(1N)
rebuild	NIS database	ypmake(1M)
the	NIS database and directory structure	ypfiles(4N)
	NIS database conversion routines	map2ascii(1M)
make an	NIS dbm file	makedbm(1M)
get/set name of current	NIS domain	getdomainname(3N)
set or display name of current	NIS domain	domainname(1N)
force propagation of a changed	NIS map	yppush(1M)
print the value of one or more keys from a	NIS map	ypmatch(1N)
transfer an	NIS map from some NIS server to here	ypxfr(1M)
server for modifying	NIS password file	yppasswdd(1M)
	NIS server and binder processes	ypserv(1M)
which host is the	NIS server or map master?	ypwhich(1N)
transfer an NIS map from some	NIS server to here	ypxfr(1M)
scripts NFS,	NIS, and REX start/stop and configuration	nfs(1M)
line numbering filter	nl	nl(1)
get entries from name list	nlist	nlist(3C)
network listener service/	nlsadmin	nlsadmin(1M)
print name list of common/	nm	nm(1)
null modem daemon pair	nmd	nmd(1M)
SSP interface to the/	nmi	nmi(1M)
mount, unmount multiple file/	nmountall	nmountall(1M)
run a command immune to hangups/	nohup	nohup(1)
	non-local goto	setjmp(3C)
	non-local jumps	sigsetjmp(3P)
Operating System general	non-volatile RAM driver for the REAL/IX	nvram(7)
be	notified if mail arrives and who it is from	biff(1)
generate uniformly distributed/	nrand48	drand48(3C)

remove nroff/troff, tbl, and eqn constructs ..... deroff(1)
close a shared memory name/ nsclose ..... nsclose(3C)
create a shared memory name/ nscreat ..... nscreat(3C)
delete a record from a shared/ nsdelete ..... nsdelete(3C)
get a record from a shared/ nsget ..... nsget(3C)
insert a record into a shared/ nsinsert ..... nsinsert(3C)
map an id to a corresponding/ nsiton ..... nsiton(3C)
map a name to a corresponding/ nsntoi ..... nsntoi(3C)
open a shared memory name/ nsopen ..... nsopen(3C)
remove a shared memory name/ nsremove ..... nsremove(3C)
convert values between host and/ ntohl ..... byteorder(3N)
convert values between host and/ ntohs ..... byteorder(3N)
the null file null ..... null(7)
the null file ..... null(7)
null modem daemon pair ..... nmd(1M)
shell procedures for accounting nulladm ..... acctsh(1M)
convert a virtual to a physical cpu number ..... vcpuno(2)
convert string to double-precision number ..... strtod(3C)
get RPC port number ..... getrpcport(3N)
obtain the prime factors of a number ..... factor(1)
rpc program number data base ..... rpc(4N)
line number entries in a common object file ..... linenum(4)
manipulate line number entries of a common object file function ldlread(3X)
file seek to line number entries of a section of a common object ldlseek(3X)
strip symbol and line number information from a common object file strip(1)
DARPA port to RPC program number mapper ..... portmap(1M)
print the number of days since the Epoch ..... ypday(1M)
report number of free disk blocks and i-nodes ..... df(1M)
return the number of the cpu this process is on ..... cpuno(2)
convert floating-point number to string ..... ecvt(3C)
line numbering filter ..... nl(1)
file produce a C source listing with line numbers from a common object ..... list(1)
find device filename from major and minor numbers ..... getdevname(1M)
generate uniformly distributed pseudo-random numbers ..... drand48(3C)
install random inode generation numbers ..... fsirand(1M)
introduction to system calls and error numbers ..... intro(2)
manipulate parts of floating-point numbers ..... frexp(3C)
track numbers into file system logical block numbers convert ..... fblkgen(1M)
convert track numbers into file system logical block numbers fblkgen(1M)
mount, unmount multiple file/ numountall ..... nmountall(1M)
general non-volatile RAM driver/ nvram ..... nvram(7)
pattern scanning and processing/ oawk ..... oawk(1)
synchronise a memory object ..... msync(2)
files that hold object architecture information ..... ott(4)
object code disassembler ..... dis(1)
an indexed/named section header of a common object file read ..... ldshread(3X)
and line number information from a common object file strip symbol ..... strip(1)
close a common object file ..... ldclose(3X)
compress a common object file ..... cprs(1)
dump selected parts of an object file ..... dump(1)
line number entries in a common object file ..... linenum(4)
print name list of common object file ..... nm(1)
read an indexed symbol table entry of a common object file ..... ldtbread(3X)
read the file header of a common object file ..... ldfhread(3X)
relocation information for a common object file ..... reloc(4)
section header for a common object file ..... scnhdr(4)
seek to an indexed/named section of a common object file ..... ldsseek(3X)

seek to the optional file header of a common object file ..... ldohseek(3X)
seek to the symbol table of a common object file ..... ldtbseek(3X)
the index of a symbol table entry of a common object file compute ..... ldtbindex(3X)
to line number entries of a section of a common object file seek ..... ldlseek(3X)
to relocation entries of a section of a common object file seek ..... ldrseek(3X)
common object file access routines ..... ldfcn(4)
manipulate the object file comment section ..... mcs(1)
common object file converter ..... conv(1)
open a common object file for reading ..... ldopen(3X)
manipulate line number entries of a common object file function ..... ldlread(3X)
line numbers from a common object file produce a C source listing with .... list(1)
retrieve symbol name for common object file symbol table entry ..... ldgetname(3X)
common object file symbol table format ..... syms(4)
display character strings in object files ..... strings(1)
file header for common object files ..... filehdr(4)
link editor for common object files ..... ld(1)
print section sizes in bytes of common object files ..... size(1)
install object files in binary directories ..... cpset(1M)
compare common object format modules ..... coffcmp(1)
dump common object format modules ..... coffdump(1)
find ordering relation for an object library ..... lorder(1)
C Obscure Feature Finder ..... coff(1)
obtain the prime factors of a number ..... factor(1)
octal dump ..... od(1)
octal dump od ..... od(1)
execute a command remotely on ..... on(1N)
converts from a two- to a one-password file scheme ..... pwunconv(1M)
hold serial lines open ..... port_hold(1M)
open for reading or writing open ..... open(2)
open a common object file for reading ..... ldopen(3X)
open a shared memory name server database . nsopen(3C)
open a stream ..... fopen(3S)
open any minor device on a STREAMS driver clone(7)
duplicate an open file descriptor ..... dup(2)
duplicate an open file descriptor ..... dup2(3C)
open for reading or writing ..... open(2)
directory operations opendir ..... directory(3X)
configure and generate the operating system ..... sysgen(1M)
general non-volatile RAM driver for the REAL/IX Operating System ..... nvram(7)
run commands performed to stop the operating system ..... rc0(1M)
operating system profiler ..... prf(7)
configure SCSI devices for realtime operation ..... scsiconf(1M)
control dual console operation ..... dcon(1M)
control block for asynchronous I/O operations ..... aiocb(4)
directory operations ..... directory(3X)
event control operations ..... evctl(2)
memory operations ..... memory(3C)
message control operations ..... msgctl(2)
semaphore control operations ..... semctl(2)
semaphore operations ..... semop(2)
shared memory control operations ..... shmctl(2)
shared memory operations ..... shmop(2)
string operations ..... string(3C)
send and receive messages (message operations) ..... msgop(2)
relational database operator ..... join(1)
copy file systems for optimal access time ..... dcopy(1M)
terminal screen handling and optimization package ..... curses(3X)

get	option letter from argument vector	getopt(3C)
seek to the	optional file header of a common object file	ldohseek(3X)
file control	options	fcntl(5)
parse command	options	getopt(1)
parse command	options	getopts(1)
set the	options for a terminal	stty(1)
manage	options for a transport endpoint	t_optmgmt(3N)
get and set	options on sockets	getsockopt(2N)
convert values between host and network byte	order	byteorder(3N)
find	ordering relation for an object library	lorder(1)
initiate an	orderly release	t_sndrel(3N)
acknowledge receipt of an	orderly release indication	t_rcvrel(3N)
make a directory, or a special or	ordinary file	mknod(2)
files that hold object/	ott	ott(4)
establish an	out-going terminal line connection	dial(3C)
common assembler and link editor	output	a.out(4)
print formatted	output	printf(3S)
get or set the value of the	output and input baud rate	cfgetospeed(3P)
fold long lines for finite width	output device	fold(1)
print formatted	output of a varargs argument list	vprintf(3S)
accounting commands	overview of accounting and miscellaneous	acct(1M)
Quadbus multiprocessor a brief	overview of the software architecture of the	realstar(5)
change	owner	chown(1)
change	owner and group of a file	chown(2)
change	owner or group	chown(1)
compress and expand files	pack	pack(1)
standard buffered input/output	package	stdio(3S)
standard interprocess communication	package	stdipc(3C)
system activity report	package	sar(1M)
terminal screen handling and optimization	package	curses(3X)
software management	package extraction facility	ixf(1M)
software management	package generation facility	igf(1M)
send ICMP ECHO_REQUEST	packets to network hosts	ping(1M)
file perusal filter for crt/	page	more(1)
format of manual	page translation files	man(4)
	paginator for the Tektronix 4014 terminal	4014(1)
null modem daemon	pair	nmd(1M)
get process, process group, and	parent process IDs	getpid(2)
	parse command options	getopt(1)
	parse command options	getopts(1)
GNU project	parser generator (yacc replacement)	bison(1)
run a command on a	particular cpu	target(1R)
return some hardware and software data on a	particular cpu.	getcpudata(2)
poin' **ypbind** at a	particular server	ypset(1M)
mmand to dump system memory image to dump	partition	sysdump(1M)
dump selected	parts of an object file	dump(1)
manipulate	parts of floating-point numbers	frexp(3C)
password files management	passmgmt	passmgmt(1M)
change login password and/	passwd	passwd(1)
password file	passwd	passwd(4)
read a	password	getpass(3C)
	password and file encryption functions	crypt(3X)
change login	password and password attributes	passwd(1)
change login password and	password attributes	passwd(1)
	password file	passwd(4)
server for modifying NIS	password file	yppasswdd(1M)
get shadow	password file entry	getspent(3X)

get	password file entry	getpwent(3C)
write shadow	password file entry	putspent(3X)
write	password file entry	putpwent(3C)
	password file management for the NIS	yppassmgmt(1M)
	password files management	passmgmt(1M)
change login	password in NIS	yppasswd(1N)
update user	password in NIS	yppasswd(3N)
	password/group file checkers	pwck(1M)
merge same lines of several/	paste	paste(1)
examine and change bytes in a/	patch	patch(1M)
dereference bug	patch program with **NULL** pointer	dnp(1)
deadlock	patch XC88100 COFF executables to avoid fdiv	fdivfix(1)
generate	path names from i-numbers	ncheck(1M)
get	path-name of current working directory	getcwd(3C)
alias file for framed access/	pathalias	alias(4)
get configurable pathname/	pathconf	pathconf(2P)
get configurable	pathname variables	pathconf(2P)
deliver portions of	pathnames	basename(1)
search a file for a	pattern	grep(1)
	pattern scanning and processing language	awk(1)
	pattern scanning and processing language	oawk(1)
search a file for a	pattern using full regular expressions	egrep(1)
suspend process until signal	pause	pause(2)
compress and expand files	pcat	pack(1)
initiate pipe to/from a process	pclose	popen(3S)
get processor type truth value	pdp11	machid(1)
get name of connected	peer	getpeername(2N)
gets name of connected	peer	getpeername(3N)
examine	pending signals	sigpending(2P)
	per-process accounting file format	acct(4)
command summary from	per-process accounting records	acctcms(1M)
driver for the MODCOMP 4211	Peregrine FDDI Controller	fddi4211(7)
maintenance bus	perform a simple loopback test on the SSP	mbloop(1M)
maintenance bus	perform one sync transfer on the SSP	mbs(1M)
change	performance tunings of file systems	tunefs(1M)
run commands	performed for multi-user environment	rc2(1M)
run commands	performed for the SSP multiuser environment	rcqwb(1M)
run commands	performed to stop the operating system	rc0(1M)
check the uucp directories and	permissions file	uucheck(1M)
read the	permissions file	getpermsintro(3X2)
echo the real device in the	permissions file for a given alias	real(1)
access utilities	permissions file used by the value-added disk	filesys(4)
	permit or deny messages	mesg(1)
system error messages	perror	perror(3C)
file	perusal filter for crt viewing	more(1)
file	perusal filter for CRTs	pg(1)
control structures for the/	pevbk_t	evt(4)
file perusal filter for CRTs	pg	pg(1)
convert a virtual to a	physical cpu number	vcpuno(2)
split a file into	pieces	split(1)
send ICMP ECHO_REQUEST packets/	ping	ping(1M)
create an interprocess channel	pipe	pipe(2)
	pipe fitting	tee(1)
initiate	pipe to/from a process	popen(3S)
lock process, text, or data in/	plock	plock(2)
	point **ypbind** at a particular server	ypset(1M)
move read/write file	pointer	lseek(2)

patch program with **NULL**	pointer dereference bug	dnp(1)
reposition a file	pointer in a stream	fseek(3S)
STREAMS input/output/	poll	poll(2)
turn SSP maintenance bus	polling on or off	mbpoll(1M)
initiate pipe to/from a process	popen	popen(3S)
bind a socket to a privileged IP	port	bindresvport(3N)
get RPC	port number	getrpcport(3N)
DARPA	port to RPC program number mapper	portmap(1M)
archive and library maintainer for	portable archives	ar(1)
configure or display *tty*/	portconfig	portconfig(1M)
configure or display *tty*/	portdisplay	portconfig(1M)
deliver	portions of pathnames	basename(1)
DARPA port to RPC program/	portmap	portmap(1M)
configure or display *tty*	ports	portconfig(1M)
serial	ports on the MVME181/MVME188	m681(7)
serial	ports on the MVME187	cd2401(7)
hold serial lines open	port_hold	port_hold(1M)
	post an event to a process	evpost(2)
make	posters	banner(1)
exponential, logarithm, power,/	pow	exp(3M)
stop all processes and turn off the	power	powerdown(1M)
exponential, logarithm,	power, square root functions	exp(3M)
stop all processes and turn off/	powerdown	powerdown(1M)
print files	pr	pr(1)
shell procedures for accounting	prctmp	acctsh(1M)
shell procedures for accounting	prdaily	acctsh(1M)
preallocate contiguous file/	prealloc	prealloc(1R)
preallocate contiguous file/	prealloc	prealloc(2)
	preallocate contiguous file space	prealloc(1R)
	preallocate contiguous file space	prealloc(2)
shutdown the system with extreme	prejudice	halt(1M)
	prepare execution profile	monitor(3C)
	preprocess MC88100 assembly language	sifilter(1)
the C language	preprocessor	cpp(1)
allow or	prevent LP requests	accept(1M)
undo a	previous get of an SCCS file	unget(1)
unmap	previously mapped addresses	munmap(2)
operating system profiler	prf	prf(7)
UNIX system profiler	prfdc	profiler(1M)
UNIX system profiler	prfld	profiler(1M)
UNIX system profiler	prfpr	profiler(1M)
UNIX system profiler	prfsnap	profiler(1M)
UNIX system profiler	prfstat	profiler(1M)
obtain the	prime factors of a number	factor(1)
	primitive system data types	types(5)
	print an SCCS file	prs(1)
	print and set the date	date(1)
	print calendar	cal(1)
	print checksum and block count of a file	sum(1)
	print CPU board information	cputype(1)
	print current SCCS file editing activity	sact(1)
	print effective current user id	whoami(1)
	print entries in this manual	man(1)
	print files	pr(1)
concatenate and	print files	cat(1)
	print formatted output	printf(3S)
list	print formatted output of a varargs argument	vprintf(3S)

print service — print information about the status of the LP . lpstat(1)
print name list of common object file ..... nm(1)
set or — print name of current host system ..... hostname(1N)
print name of current system ..... uname(1)
print news items ..... news(1)
messages. — print out incoming SSP maintenance bus async mba(1M)
compare or — print out terminfo descriptions ..... infocmp(1M)
search and — print process accounting file(s) ..... acctcom(1)
files — print section sizes in bytes of common object . size(1)
administer filters used with the LP — print service ..... lpfilter(1M)
administer forms used with the LP — print service ..... lpforms(1M)
configure the LP — print service ..... lpadmin(1M)
print information about the status of the LP — print service ..... lpstat(1)
send/cancel requests to an LP — print service ..... lp(1)
start/stop the LP — print service and move requests ..... lpsched(1M)
print STREAMS trace messages ..... strace(1M)
print system configuration ..... prtconf(1M)
print the number of days since the Epoch .... ypday(1M)
map — print the value of one or more keys from a NIS ypmatch(1N)
print user and group IDs and names ..... id(1M)
print values in a NIS database ..... ypcat(1N)
print various file system information ..... fsinfo(1M)
network — printer daemon ..... netlpd(1N)
line — printer interface ..... lp(7)
enable/disable LP — printers ..... enable(1)
print formatted output — printf ..... printf(3S)
kernel console — printf logger ..... putbuf(1)
set — printing queue priorities ..... lpusers(1M)
send network — printing requests to a remote host ..... netlp(1N)
prints out STREAMS statistics ..... strstat(1M)
set printing queue — priorities ..... lpusers(1M)
get scheduling — priority ..... getpri(2)
run a command at low — priority ..... nice(1)
set scheduling — priority ..... setpri(1R)
set scheduling — priority ..... setpri(2)
change — priority of a process ..... nice(2)
set realtime — privileged groups list ..... setrtgroups(1M)
set realtime — privileged groups list ..... setrtgroups(2)
view the realtime — privileged groups list ..... rtgroups(1)
bind a socket to a — privileged IP port ..... bindresvport(3N)
set realtime — privileged users list ..... setrtusers(1M)
set realtime — privileged users list ..... setrtusers(2)
view the realtime — privileged users list ..... rtusers(1)
set/clear realtime — privileges ..... setrt(2)
Network Interface — Probe ..... ifprobe(1M)
library routines for remote — procedure calls ..... rpc(3N)
system initialization — procedures ..... bcheckrc(1M)
shell — procedures for accounting ..... acctsh(1M)
await completion of — process ..... wait(1)
change priority of a — process ..... nice(2)
create a new — process ..... fork(2)
for asynchronous I/O from the — process free internal resources ..... arwfree(2)
initiate pipe to/from a — process ..... popen(3S)
post an event to a — process ..... evpost(2)
resume a suspended — process ..... resume(1R)
resume a suspended — process ..... resume(2)
script for the init — process ..... inittab(4)

suspend the calling	process	suspend(2)
switch into a	process	swtch(2)
terminate a	process	kill(1)
terminate	process	exit(2)
	process a report of logged errors	errpt(1M)
	process accounting	acctprc(1M)
enable or disable	process accounting	acct(2)
search and print	process accounting file(s)	acctcom(1)
map	process addresses to a file	mmap(2)
set a	process alarm clock	alarm(2)
get	process and child process times	times(2)
	process control initialization	init(1M)
multiprocessor	process control initialization	mpinit(1M)
time a command; report	process data and system activity	timex(1)
set	process group	setpgid(2P)
get distinguished	process group ID	tcgetpgrp(3P)
set distinguished	process group ID	tcsetpgrp(3P)
set	process group ID	setpgrp(2)
set	process group id and execute command	setpgrp(1)
get process,	process group, and parent process IDs	getpid(2)
get a unique identifier for a	process interval timer	gettimerid(2)
get resolution and maximum time value of	process interval timer	resabs(2)
get the current value for a	process interval timer	getinterval(2)
set the expiration time of a	process interval timer	absinterval(2)
release a	process interval timer identifier	reltimerid(2)
format of time values for	process interval timers	itimerstruc(4)
return the number of the cpu this	process is on	cpuno(2)
send a signal to a	process or a group of processes	kill(2)
report	process status	ps(1)
get process and child	process times	times(2)
target the calling	process to some cpu	targetcpu(2)
wait for child	process to stop or terminate	wait(2)
wait for child	process to stop or terminate	waitpid(2)
	process trace	ptrace(2)
suspend	process until signal	pause(2)
get process, process group, and parent	process IDs	getpid(2)
IDs get	process, process group, and parent process	getpid(2)
lock	process, text, or data in memory	plock(2)
list of file systems	processed by fsck and ncheck	checklist(4)
NIS server and binder	processes	ypserv(1M)
kill all active	processes	killall(1M)
send a signal to a process or a group of	processes	kill(2)
stop all	processes and turn off the power	powerdown(1M)
identify	processes using a file or file structure	fuser(1M)
pattern scanning and	processing language	awk(1)
pattern scanning and	processing language	oawk(1)
interactive message	processing system	mailx(1)
macro	processor	m4(1)
decode semaphore	processor instructions	smp(1M)
set/get	Processor Status Register	setpsr(2)
get	processor type truth value	machid(1)
from a common object file	produce a C source listing with line numbers	list(1)
	produce error message	t_error(3N)
display profile data	prof	prof(1)
profile within a function	prof	prof(5)
execution time profile	profil	profil(2)
execution time	profile	profil(2)

prepare execution profile ..... monitor(3C)
setting up an environment at/ profile ..... profile(4)
display call graph profile data ..... gprof(1)
display profile data ..... prof(1)
profile within a function ..... prof(5)
UNIX system profiler ..... profiler(1M)
operating system profiler ..... prf(7)
start, stop, sample, or reset kernel profiling ..... kgmon(1M)
ARPANET file transfer program ..... ftp(1N)
MVME332XT control program ..... m332xctl(1M)
MVME336 control program ..... m336ctl(1M)
MVME350 control program ..... m350ctl(1M)
MVME355 control program ..... m355ctl(1M)
STREAMS error logger cleanup program ..... strclean(1M)
conversion program ..... units(1)
last locations in program ..... end(3C)
maintain a tags file for a C program ..... ctags(1)
network configuration program ..... netc(1M)
side-by-side difference program ..... sdiff(1)
tape device control program ..... tapectl(1M)
the scheduler for the uucp file transport program ..... uusched(1M)
trivial file transfer program ..... tftp(1M)
user information lookup program ..... finger(1)
verify program assertion ..... assert(3X)
C program beautifier ..... cb(1)
a C program checker ..... lint(1)
generate C program cross-reference ..... cxref(1)
C program debugger ..... ctrace(1)
or command forms program for generating/modifying ASSIST menus astgen(1)
file transport program for the uucp system ..... uucico(1M)
generate C program makefile dependencies ..... depend(1)
rpc program number data base ..... rpc(4N)
DARPA port to RPC program number mapper ..... portmap(1M)
dereference bug patch program with **NULL** pointer ..... dnp(1)
shell, the standard/restricted command and programming language ..... sh(1)
shell, the standard/restricted command programming language ..... ksh(1)
the bourne shell command programming language ..... bsh(1)
introduction to commands and application programs ..... intro(1)
introduction to commands and application programs ..... intro(1N)
maintain, update, and regenerate groups of programs ..... make(1)
to maintenance commands and application programs introduction ..... intro(1M)
generate programs for simple lexical tasks ..... lex(1)
GNU project C Compiler ..... gcc(1)
GNU project parser generator (yacc replacement) .. bison(1)
force propagation of a changed NIS map ..... yppush(1M)
Address Resolution Protocol ..... arp(7N)
Internet Transmission Control Protocol ..... tcp(7)
Internet User Datagram Protocol ..... udp(7)
remote execution protocol ..... rex(3N)
user interface to the TELNET protocol ..... telnet(1N)
an RPC protocol compiler ..... rpcgen(1N)
get protocol entry ..... getprotoent(3N)
Internet protocol family ..... inet(7N)
address resolution protocol maintenance ..... arp(1M)
protocol name data base ..... protocols(4N)
DARPA Internet File Transfer Protocol server ..... ftpd(1M)
DARPA TELNET protocol server ..... telnetd(1M)

DARPA Trivial File Transfer	Protocol server	tftpd(1M)
Internet Boot	Protocol server	()
Internet IP	protocol streams multiplexer module	ip(7N)
get	protocol-specific service information	t_getinfo(3N)
protocol name data base	protocols	protocols(4N)
	provide labels for file systems	labelit(1M)
	provide truth values	true(1)
Transport	Provider Interface daemon	tpid(1M)
print an SCCS file	prs	prs(1)
shell procedures for accounting	prtacct	acctsh(1M)
print system configuration	prtconf	prtconf(1M)
report process status	ps	ps(1)
	pseudo shell layer, console manager	conman(1)
	pseudo-device driver	sxt(7)
generate uniformly distributed	pseudo-random numbers	drand48(3C)
	pseudo-terminal driver	pty(7)
pseudo shell layer, console/	pshl	conman(1)
process trace	ptrace	ptrace(2)
pseudo-terminal driver	pty	pty(7)
recondition	pty slave devices	ptyfix(1M)
recondition pty slave devices	ptyfix	ptyfix(1M)
	public UNIX-to-UNIX system file copy	uuto(1C)
	push character back into input stream	ungetc(3S)
	put a string on a stream	puts(3S)
	put character or word on a stream	putc(3S)
read directory entries and	put in a file system independent format	getdents(2)
kernel console printf logger	putbuf	putbuf(1)
put character or word on a/	putc	putc(3S)
put character or word on a/	putchar	putc(3S)
change or add value to/	putenv	putenv(3C)
send a message on a stream	putmsg	putmsg(2)
write password file entry	putpwent	putpwent(3C)
put a string on a stream	puts	puts(3S)
write shadow password file entry	putspent	putspent(3X)
access utmp file entry	pututline	getut(3C)
put character or word on a/	putw	putc(3S)
password/group file checkers	pwck	pwck(1M)
install and update /etc/shadow/	pwconv	pwconv(1M)
working directory name	pwd	pwd(1)
converts from a two- to a/	pwunconv	pwunconv(1M)
quicker sort	qsort	qsort(3C)
	Quadbus console interface	qwbcon(7)
	Quadbus device driver interface to SSP devices	qwbsys(7)
overview of the software architecture of the	Quadbus multiprocessor a brief	realstar(5)
	QuadBus system interface to the SSP monitor	sspmon(1M)
set or	query current locale	setlocale(3C)
initialize a terminal or	query terminfo database	tput(1)
yes/no response from user or check answer to	question get	chkyn(1)
get message	queue	msgget(2)
set printing	queue priorities	lpusers(1M)
remove a message	queue, semaphore set, or shared memory ID	ipcrm(1)
receive any	queued event	evrcv(2)
receive any	queued event from a specified list	evrcvl(2)
receive multiple	queued events	evrcvm(2)
	quicker sort	qsort(3C)
run a command immune to hangups and	quits	nohup(1)
	qwb daemons	qwbd(1M)

clear QWB memory ..... cm(1M)
bootstrap loader for QWB multiprocessor ..... bo(1M)
manager QWB multiprocessor configuration database .. cf(1M)
QWB multiprocessor system initialization .... si(1M)
Quadbus console interface qwbcon ..... qwbcon(7)
qwb daemons qwbd ..... qwbd(1M)
change the qwbd daemon debug trace flags ..... qwbdflags(1M)
initialization file for the qwbd daemons. ..... qwbtab(4)
change the qwbd daemon debug/ qwbdflags ..... qwbdflags(1M)
SSP device driver interface to/ qwbserv ..... qwbserv(7)
Quadbus device driver interface/ qwbsys ..... qwbsys(7)
initialization file for the/ qwbtab ..... qwbtab(4)
general non-volatile RAM driver for the REAL/IX Operating System nvram(7)
general ramdisk driver for the/ ramdisk ..... ramdisk(7)
general ramdisk driver for the REAL/IX System ..... ramdisk(7)
simple random-number generator rand ..... rand(3C)
install random inode generation numbers ..... fsirand(1M)
simple random-number generator ..... rand(3C)
or set the value of the output and input baud rate get ..... cfgetospeed(3P)
summarize cache hit rate for each buffer size ..... bfree(1M)
the bourne shell command/ rbsh ..... bsh(1)
run commands performed to stop/ rc0 ..... rc0(1M)
run commands performed for/ rc2 ..... rc2(1M)
routines for returning a stream/ rcmd ..... rcmd(3N)
remote file copy rcp ..... rcp(1N)
run commands performed for the/ rcqwb ..... rcqwb(1M)
read from file read ..... read(2)
structures before requesting an asynchronous read initialize ..... arinit(2)
read (copy) a MSDOS file to Unix ..... mread(1)
read a password ..... getpass(3C)
object file read an indexed symbol table entry of a common ldtbread(3X)
common object file read an indexed/named section header of a ... ldshread(3X)
independent format read directory entries and put in a file system . getdents(2)
read from file ..... read(2)
read from file in an asynchronous manner .... aread(2)
send mail to users or read mail ..... mail(1)
read one line ..... line(1)
archive file read the archive header of a member of an ... ldahread(3X)
read the file header of a common object file ... ldfhread(3X)
read the permissions file ..... getpermsintro(3X2)
read value of a symbolic link ..... readlink(2)
move read/write file pointer ..... lseek(2)
Transport Interface read/write interface STREAMS module ..... tirdwr(7)
directory operations readdir ..... directory(3X)
open a common object file for reading ..... ldopen(3X)
open for reading or writing ..... open(2)
read value of a symbolic link readlink ..... readlink(2)
do multiple reads from a file ..... readv(2)
do multiple reads from a file readv ..... readv(2)
read from file readv ..... read(2)
echo the real device in the/ real ..... real(1)
alias echo the real device in the permissions file for a given .. real(1)
get real user, effective user, real group, and effective group IDs ..... getuid(2)
effective group IDs get real user, effective user, real group, and ..... getuid(2)
assistance using REAL/IX commands ..... assist(1)
general non-volatile RAM driver for the REAL/IX Operating System ..... nvram(7)
/9-track tape implementation details used by REAL/IX realtime SCSI autoconfiguration drivers scsi9track(7)

general tape implementation details used by	REAL/IX realtime SCSI autoconfiguration drivers	scsitape(7)
various	REAL/IX specific functions	sysrealix(2)
general ramdisk driver for the	REAL/IX System	ramdisk(7)
fast main memory allocator	realloc	malloc(3C)
main memory allocator	realloc	malloc(3X)
a brief overview of the/	RealStar	realstar(5)
introduction to	realtime commands	intro(1R)
configure SCSI devices for	realtime operation	scsiconf(1M)
set	realtime privileged groups list	setrtgroups(1M)
set	realtime privileged groups list	setrtgroups(2)
view the	realtime privileged groups list	rtgroups(1)
set	realtime privileged users list	setrtusers(1M)
set	realtime privileged users list	setrtusers(2)
view the	realtime privileged users list	rtusers(1)
set/clear	realtime privileges	setrt(2)
/tape implementation details used by REAL/IX	realtime SCSI autoconfiguration drivers	scsi9track(7)
/tape implementation details used by REAL/IX	realtime SCSI autoconfiguration drivers	scsitape(7)
and maximum time values for a system-wide	realtime timer get the resolution	restimer(2)
get the current value for a system-wide	realtime timer	gettimer(2)
set the current value for a system-wide	realtime timer	settimer(2)
format of time for system-wide	realtime timers	timestruc(4)
	rebuild NIS database	ypmake(1M)
specify what to do upon	receipt of a signal	signal(2)
acknowledge	receipt of an orderly release indication	t_rcvrel(3N)
	receive a data unit	t_rcvudata(3N)
	receive a unit data error indication	t_rcvuderr(3N)
	receive any queued event	evrcv(2)
	receive any queued event from a specified list	evrcvl(2)
	receive data from a socket	recv(2N)
connection	receive data or expedited data sent over a	t_rcv(3N)
send and	receive messages (message operations)	msgop(2)
	receive multiple queued events	evrcvm(2)
	receive the confirmation from a connect request	t_rcvconnect(3N)
	recondition pty slave devices	ptyfix(1M)
database delete a	record from a shared memory name server	nsdelete(3C)
database get a	record from a shared memory name server	nsget(3C)
database insert a	record into a shared memory name server	nsinsert(3C)
	record locking on files	lockf(3C)
command summary from per-process accounting	records	acctcms(1M)
manipulate connect accounting	records	fwtmp(1M)
extract error	records from dump	errdead(1M)
	recover files from a backup tape	frec(1M)
receive data from a socket	recv	recv(2N)
receive data from a socket	recvfrom	recv(2N)
receive data from a socket	recvmsg	recv(2N)
text editor	red	ed(1)
generate cyclic	redundancy checksums (crc) of files	crc(1)
compile and execute regular/	regcmp	regcmp(3X)
regular expression compile	regcmp	regcmp(1)
maintain, update, and	regenerate groups of programs	make(1)
compile and execute regular/	regex	regcmp(3X)
regular expression compile and/	regexp	regexp(5)
expand the stack	region of the data segment	stkexp(2)
set/get Processor Status	Register	setpsr(2)
compile and execute	regular expression	regcmp(3X)
	regular expression compile	regcmp(1)
	regular expression compile and match routines	regexp(5)

search a file for a pattern using full regular expressions ..... egrep(1)
allow or prevent LP requests reject ..... accept(1M)
select or reject lines common to two sorted files ..... comm(1)
find ordering relation for an object library ..... lorder(1)
relational database operator ..... join(1)
initiate an orderly release ..... t_sndrel(3N)
release a process interval timer identifier ..... reltimerid(2)
release an event identifier ..... evrel(2)
acknowledge receipt of an orderly release indication ..... t_rcvrel(3N)
voluntarily give up CPU relinquish ..... relinquish(2)
relocation information for a/ reloc ..... reloc(4)
object file seek to relocation entries of a section of a common ... ldrseek(3X)
relocation information for a common object file reloc(4)
release a process interval/ reltimerid ..... reltimerid(2)
floor, ceiling, remainder, absolute value functions ..... floor(3M)
source code remake the binary system and commands from :mk(1M)
reminder service ..... calendar(1)
routines for returning a stream to a remote command ..... rcmd(3N)
execute remote command requests ..... uuxqt(1M)
remote execution protocol ..... rex(3N)
RPC-based remote execution server ..... rexd(1M)
remote file copy ..... rcp(1N)
send network printing requests to a remote host ..... netlp(1N)
remote login ..... rlogin(1)
remote login server ..... rlogind(1M)
show all remote mounts ..... showmount(1M)
library routines for remote procedure calls ..... rpc(3N)
mount and unmount file systems and remote resources ..... mount(1M)
remote shell ..... remsh(1)
remote shell server ..... remshd(1M)
try to contact remote system with debugging on ..... uutry(1M)
spawn getty to a remote terminal ..... ct(1C)
remote user information server ..... fingerd(1M)
execute a command remotely ..... on(1N)
NFS remotely mounted file system table ..... rmtab(4N)
keep track of remotely mounted file systems ..... mount(3N)
remove a delta from an SCCS file ..... rmdel(1)
remove a directory ..... rmdir(2)
shared memory ID remove a message queue, semaphore set, or ... ipcrm(1)
remove a MSDOS sub directory ..... mrd(1)
remove a shared memory name server database nsremove(3C)
remove columns from a file ..... colrm(1)
remove directory entry ..... unlink(2)
remove files or directories ..... rm(1)
remove ifdef'ed lines ..... unifdef(1)
remove nroff/troff, tbl, and eqn constructs ... deroff(1)
copy removeable media ..... dcpy(1)
remote shell remsh ..... remsh(1)
remote shell server remshd ..... remshd(1M)
change the name of a file rename ..... rename(2)
rename an existing MSDOS file ..... mren(1)
move or rename files ..... mv(1)
file system check and interactive repair ..... chk(1)
check and repair file systems ..... fsck(1M)
report repeated lines in a file ..... uniq(1)
GNU project parser generator (yacc replacement) ..... bison(1)
report CPU time used ..... clock(3C)

	report file system status	fsstat(1M)
status	report inter-process communication facilities	ipcs(1)
	report number of free disk blocks and i-nodes	df(1M)
process a	report of logged errors	errpt(1M)
system activity	report package	sar(1M)
time a command;	report process data and system activity	timex(1)
	report process status	ps(1)
	report repeated lines in a file	uniq(1)
	report RPC information	rpcinfo(1M)
system activity	reporter	sar(1)
	reposition a file pointer in a stream	fseek(3S)
library routines for external data	representation	xdr(3N)
accept a connect	request	t_accept(3N)
listen for a connect	request	t_listen(3N)
receive the confirmation from a connect	request	t_rcvconnect(3N)
send user-initiated disconnect	request	t_snddis(3N)
NFS mount	request server	mountd(1M)
initialize structures before	requesting an asynchronous read	arinit(2)
initialize structures before	requesting an asynchronous write	awinit(2)
allow or prevent LP	requests	accept(1M)
cancel one or more asynchronous I/O	requests	acancel(2)
execute remote command	requests	uuxqt(1M)
start/stop the LP print service and move	requests	lpsched(1M)
ioctl	requests for sockets	sockio(7N)
send network printing	requests to a remote host	netlp(1N)
send/cancel	requests to an LP print service	lp(1)
get resolution and maximum time/	resabs	resabs(2)
	reserve a terminal	lock(1)
start, stop, sample, or	reset kernel profiling	kgmon(1M)
make locked segments resident/	resident	resident(2)
make locked segments	resident in memory	resident(2)
get resolution and maximum time/	resinc	resabs(2)
interval timer get	resolution and maximum time value of process	resabs(2)
system-wide realtime timer get the	resolution and maximum time values for a	restimer(2)
Address	Resolution Protocol	arp(7N)
address	resolution protocol maintenance	arp(1M)
high	resolution sleep	nanosleep(2)
get	resolution supported by nanosleep(2)	nanosleep_getres(2)
resolver configuration file	resolv.conf	resolv.conf(4N)
	resolver configuration file	resolv.conf(4N)
	resolver routines	resolver(3N)
mount and unmount file systems and remote	resources	mount(1M)
the process free internal	resources for asynchronous I/O from	arwfree(2)
get yes/no	response from user or check answer to question	chkyn(1)
get the resolution and maximum/	restimer	restimer(2)
back up and	restore files and/or file systems	br(1M)
backup and	restore utility	bru(1)
resume a suspended process	resume	resume(1R)
resume a suspended process	resume	resume(2)
	resume a suspended process	resume(1R)
	resume a suspended process	resume(2)
resolver routines	res_init	resolver(3N)
resolver routines	res_mkquery	resolver(3N)
resolver routines	res_send	resolver(3N)
examples	retrieve a command description and usage	usage(1)
	retrieve information from disconnect	t_rcvdis(3N)
symbol table entry	retrieve symbol name for common object file	ldgetname(3X)

	return from a signal	sigret(2)
	return integer absolute value	abs(3C)
	return login name of user	logname(3X)
particular cpu.	return some hardware and software data on a	getcpudata(2)
data	return some systemwide hardware and software	getsysdata(2)
	return the number of the cpu this process is on	cpuno(2)
	return value for environment name	getenv(3C)
data	returned by stat(2) and estat(2) system calls	stat(5)
routines for	returning a stream to a remote command	rcmd(3N)
filter	reverse linefeeds	col(1)
reposition a file pointer in a/	rewind	fseek(3S)
directory operations	rewinddir	directory(3X)
create a new file or	rewrite an existing one	creat(2)
remote execution protocol	rex	rex(3N)
NFS, NIS, and	REX start/stop and configuration scripts	nfs(1M)
RPC-based remote execution/	rexd	rexd(1M)
user-specified file of/	rhosts	rhosts(4)
shell, the standard/restricted/	rksh	ksh(1)
shell, the standard/restricted/	rksh	sh(1)
remote login	rlogin	rlogin(1)
remote login server	rlogind	rlogind(1M)
remove files or directories	rm	rm(1)
send mail to users or read mail	rmail	mail(1)
remove a delta from an SCCS file	rmdel	rmdel(1)
remove a directory	rmdir	rmdir(2)
remove files or directories	rmdir	rm(1)
NFS remotely mounted file/	rmtab	rmtab(4N)
change	root directory	chroot(2)
change	root directory for a command	chroot(1M)
exponential, logarithm, power, square	root functions	exp(3M)
manually manipulate the routing/	route	route(1M)
Internet address manipulation	routines	inet(3N)
NIS database conversion	routines	map2ascii(1M)
common object file access	routines	ldfcn(4)
regular expression compile and match	routines	regexp(5)
resolver	routines	resolver(3N)
library	routines for external data representation	xdr(3N)
library	routines for remote procedure calls	rpc(3N)
command	routines for returning a stream to a remote	rcmd(3N)
multiprocessor interrupt	routing	intrctl(2)
manually manipulate the	routing tables	route(1M)
library routines for remote/	rpc	rpc(3N)
rpc program number data base	rpc	rpc(4N)
create kernel	RPC client handles	kclt_create(3N)
get	RPC entry	getrpcent(3N)
report	RPC information	rpcinfo(1M)
get	RPC port number	getrpcport(3N)
	rpc program number data base	rpc(4N)
DARPA port to	RPC program number mapper	portmap(1M)
an	RPC protocol compiler	rpcgen(1N)
	RPC-based remote execution server	rexd(1M)
an RPC protocol compiler	rpcgen	rpcgen(1N)
report RPC information	rpcinfo	rpcinfo(1M)
routines for returning a stream/	rresvport	rcmd(3N)
shell, the standard/restricted/	rsh	sh(1)
view the realtime privileged/	rtgroups	rtgroups(1)
view the realtime privileged/	rtusers	rtusers(1)

	run a command at low priority	nice(1)
	run a command immune to hangups and quits	nohup(1)
	run a command on a particular cpu	target(1R)
environment	run commands performed for multi-user	rc2(1M)
environment	run commands performed for the SSP multiuser	rcqwb(1M)
system	run commands performed to stop the operating	rc0(1M)
shell procedures for accounting	runacct	acctsh(1M)
write to all users over a network	running the TCP/IP-based software	rwall(1M)
show host status of local/	ruptime	ruptime(1N)
routines for returning a stream/	ruserok	rcmd(3N)
write to all users over a/	rwall	rwall(1M)
network	rwall server	rwalld(1M)
network rwall server	rwalld	rwalld(1M)
who's logged in on local/	rwho	rwho(1N)
system status server	rwhod	rwhod(1M)
devices administered by System/	SA	sa(7)
system activity report package	sa1	sar(1M)
system activity report package	sa2	sar(1M)
print current SCCS file editing/	sact	sact(1)
system activity report package	sadc	sar(1M)
start, stop,	sample, or reset kernel profiling	kgmon(1M)
system activity reporter	sar	sar(1)
	save kernel core dumps	savedump(1M)
	save the core image of a system dump in a file	savecore(1M)
save the core image of a system/	savecore	savecore(1M)
save kernel core dumps	savedump	savedump(1M)
change data segment space/	sbrk	brk(2)
convert formatted input	scanf	scanf(3S)
big file	scanner	bfs(1)
pattern	scanning and processing language	awk(1)
pattern	scanning and processing language	oawk(1)
change the delta commentary of a	SCCS delta	cdc(1)
combine	SCCS deltas	comb(1)
compare two versions of an	SCCS file	sccsdiff(1)
format of	SCCS file	sccsfile(4)
get a version of an	SCCS file	get(1)
make a delta to a	SCCS file	delta(1)
print an	SCCS file	prs(1)
remove a delta from an	SCCS file	rmdel(1)
undo a previous get of an	SCCS file	unget(1)
validate	SCCS file	val(1)
print current	SCCS file editing activity	sact(1)
create and administer	SCCS files	admin(1)
identify	SCCS files	what(1)
compare two versions of an SCCS/	sccsdiff	sccsdiff(1)
format of SCCS file	sccsfile	sccsfile(4)
check file system backup	schedule	ckbupscd(1M)
the	scheduler for the uucp file transport program	uusched(1M)
get	scheduling priority	getpri(2)
set	scheduling priority	setpri(1R)
set	scheduling priority	setpri(2)
converts from a two- to a one-password file	scheme	pwunconv(1M)
show socket configuration	sckconf	sckconf(1M)
section header for a common/	scnhdr	scnhdr(4)
terminal	screen handling and optimization package	curses(3X)
format of curses	screen image file	scr_dump(4)
on ex	screen-oriented (visual) display editor based	vi(1)

assembler driver	script	as(1)
make typescript of terminal/	script	script(1)
	script for the init process	inittab(4)
NFS, NIS, and REX start/stop and configuration	scripts	nfs(1M)
menus from labels stored in command shell	scripts extracts	mkmenus(1M)
format of curses screen image/	scr_dump	scr_dump(4)
implementation details used by REAL/IX realtime	SCSI autoconfiguration drivers /9-track tape	scsi9track(7)
implementation details used by REAL/IX realtime	SCSI autoconfiguration drivers general tape	scsitape(7)
autoconfig MVME187	SCSI controller interface	mvme187(7)
autoconfig MVME327	SCSI controller interface	mvme327(7)
configure	SCSI devices for realtime operation	scsiconf(1M)
general 9-track tape/	scsi9track	scsi9track(7)
configure SCSI devices for/	scsiconf	scsiconf(1M)
merge an a.out file into/	scsiload	scsiload(1M)
general tape implementation/	scsitape	scsitape(7)
symbolic debugger	sdb	sdb(1)
side-by-side difference program	sdiff	sdiff(1)
	search a file for a character string	fgrep(1)
	search a file for a pattern	grep(1)
expressions	search a file for a pattern using full regular	egrep(1)
binary	search a sorted table	bsearch(3C)
	search and print process accounting file(s)	acctcom(1)
linear	search and update	lsearch(3C)
manage hash	search tables	hsearch(3C)
manage binary	search trees	tsearch(3C)
manipulate the object file comment	section	mcs(1)
	section header for a common object file	scnhdr(4)
read an indexed/named	section header of a common object file	ldshread(3X)
seek to an indexed/named	section of a common object file	ldsseek(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)
print	section sizes in bytes of common object files	size(1)
stream editor	sed	sed(1)
generate uniformly distributed/	seed48	drand48(3C)
object file	seek to an indexed/named section of a common	ldsseek(3X)
common object file	seek to line number entries of a section of a	ldlseek(3X)
common object file	seek to relocation entries of a section of a	ldrseek(3X)
object file	seek to the optional file header of a common	ldohseek(3X)
file	seek to the symbol table of a common object	ldtbseek(3X)
directory operations	seekdir	directory(3X)
create shared memory	segment	shmget(1)
expand the stack region of the data	segment	stkexp(2)
get shared memory	segment identifier	shmget(2)
change data	segment space allocation	brk(2)
make locked	segments resident in memory	resident(2)
synchronous I/O multiplexing	select	select(2)
files	select or reject lines common to two sorted	comm(1)
	select terminal filter	greek(1)
cut out	selected fields of each line of a file	cut(1)
dump	selected parts of an object file	dump(1)
control structures for/	sem	sem(4)
conditionally lock a binary	semaphore	bslkc(3C)
free a binary	semaphore	bsfree(2)
get a binary	semaphore	bsget(2)
lock a binary	semaphore	bslk(3C)
unlock a binary	semaphore	bsunlk(3C)
	semaphore control operations	semctl(2)

	semaphore operations	semop(2)
decode	semaphore processor instructions	smp(1M)
remove a message queue,	semaphore set, or shared memory ID	ipcrm(1)
display lock and	semaphore statistics	semstat(1M)
control structures for user-level	semaphores	sem(4)
get set of	semaphores	semget(2)
semaphore control operations	semctl	semctl(2)
get set of semaphores	semget	semget(2)
semaphore operations	semop	semop(2)
display lock and semaphore/	semstat	semstat(1M)
send data over a socket	send	send(2N)
	send a data unit	t_sndudata(3N)
	send a message on a stream	putmsg(2)
processes	send a signal to a process or a group of	kill(2)
	send and receive messages (message operations)	msgop(2)
	send data or expedited data over a connection	t_snd(3N)
	send data over a socket	send(2N)
	send ICMP ECHO_REQUEST packets to network hosts	ping(1M)
	send mail to users or read mail	mail(1)
	send network printing requests to a remote host	netlp(1N)
	send user-initiated disconnect request	t_snddis(3N)
	send/cancel requests to an LP print service	lp(1)
send data over a socket	sendmsg	send(2N)
send data over a socket	sendto	send(2N)
receive data or expedited data	sent over a connection	t_rcv(3N)
hold	serial lines open	port_hold(1M)
	serial ports on the MVME181/MVME188	m681(7)
	serial ports on the MVME187	cd2401(7)
DARPA Internet File Transfer Protocol	server	ftpd(1M)
DARPA TELNET protocol	server	telnetd(1M)
DARPA Trivial File Transfer Protocol	server	tftpd(1M)
Internet Boot Protocol	server	()
NFS mount request	server	mountd(1M)
RPC-based remote execution	server	rexd(1M)
biff	server	comsat(1M)
canonical "server" process for testing	network	sink(1)
canonical "server" process for	testing network	sink(1)
canonical "server"	process for testing network	sink(1)
network rwall	server	rwalld(1M)
point **ypbind** at a particular	server	ypset(1M)
remote login	server	rlogind(1M)
remote shell	server	remshd(1M)
remote user information	server	fingerd(1M)
system status	server	rwhod(1M)
NIS	server and binder processes	ypserv(1M)
close a shared memory name	server database	nsclose(3C)
create a shared memory name	server database	nscreat(3C)
delete a record from a shared memory name	server database	nsdelete(3C)
get a record from a shared memory name	server database	nsget(3C)
insert a record into a shared memory name	server database	nsinsert(3C)
open a shared memory name	server database	nsopen(3C)
remove a shared memory name	server database	nsremove(3C)
to a corresponding id in a shared memory name	server database map a name	nsntoi(3C)
› a corresponding name in a shared memory name	server database map an id	nsiton(3C)
	server for modifying NIS password file	yppasswdd(1M)
what version of a YP map is at a YP	server host	yppoll(1M)
which host is the NIS	server or map master?	ypwhich(1N)

transfer an NIS map from some NIS	server to here	ypxfr(1M)
administer filters used with the LP print	service	lpfilter(1M)
administer forms used with the LP print	service	lpforms(1M)
configure the LP print	service	lpadmin(1M)
information about the status of the LP print	service print	lpstat(1)
reminder	service	calendar(1)
send/cancel requests to an LP print	service	lp(1)
network listener	service administration	nlsadmin(1M)
start/stop the LP print	service and move requests	lpsched(1M)
SSP specific system	service calls	sspsys(2)
get	service entry	getservent(3N)
get protocol-specific	service information	t_getinfo(3N)
	service name data base	services(4N)
service name data base	services	services(4N)
create a new	session	setsid(2)
make typescript of terminal	session	script(1)
map of ASCII character	set	ascii(5)
	set a process alarm clock	alarm(2)
	set and get file creation mask	umask(2)
	set default system time zone	timezone(4)
	set distinguished process group ID	tcsetpgrp(3P)
	set environment for command execution	env(1)
	set file access and modification times	utime(2)
	set file-creation mode mask	umask(1)
	set group access list	setgroups(2)
get	set of semaphores	semget(2)
get and	set options on sockets	getsockopt(2N)
	set or display name of current NIS domain	domainname(1N)
	set or print name of current host system	hostname(1N)
	set or query current locale	setlocale(3C)
	set printing queue priorities	lpusers(1M)
	set process group	setpgid(2P)
	set process group ID	setpgrp(2)
	set process group id and execute command	setpgrp(1)
	set realtime privileged groups list	setrtgroups(1M)
	set realtime privileged groups list	setrtgroups(2)
	set realtime privileged users list	setrtusers(1M)
	set realtime privileged users list	setrtusers(2)
	set scheduling priority	setpri(1R)
	set scheduling priority	setpri(2)
	set tabs on a terminal	tabs(1)
get and	set terminal state	tcgetattr(3P)
discipline	set terminal type, modes, speed, and line	getty(1M)
discipline	set terminal type, modes, speed, and line	uugetty(1M)
realtime timer	set the current value for a system-wide	settimer(2)
print and	set the date	date(1)
timer	set the expiration time of a process interval	absinterval(2)
	set the options for a terminal	stty(1)
get or	set the value of the output and input baud rate	cfgetospeed(3P)
	set time	stime(2)
	set user and group IDs	setuid(2)
get and	set user limits	ulimit(2)
	set value of interval timer	setitimer(2)
	set CPU time slice size	setslice(2)
remove a message queue, semaphore	set, or shared memory ID	ipcrm(1)
	set/clear realtime privileges	setrt(2)
	set/get Processor Status Register	setpsr(2)

assign buffering to a stream	setbuf	setbuf(3S)
character classification macros	setchrcla	ctype(3C)
get/set name of current NIS/	setdomainname	getdomainname(3N)
set user and group IDs	setgid	setuid(2)
get group file entry	setgrent	getgrent(3C)
set group access list	setgroups	setgroups(2)
get network host entry	sethostent	gethostent(3N)
get/set name of current host	sethostname	gethostname(3N)
set value of interval timer	setitimer	setitimer(2)
non-local goto	setjmp	setjmp(3C)
generate hashing encryption	setkey	crypt(3C)
set or query current locale	setlocale	setlocale(3C)
establish mount table	setmnt	setmnt(1M)
get network entry	setnetent	getnetent(3N)
get network group entry	setnetgrent	getnetgrent(3N)
set process group	setpgid	setpgid(2P)
set process group ID	setpgrp	setpgrp(2)
set process group id and/	setpgrp	setpgrp(1)
set scheduling priority	setpri	setpri(1R)
set scheduling priority	setpri	setpri(2)
get protocol entry	setprotoent	getprotoent(3N)
set/get Processor Status/	setpsr	setpsr(2)
get password file entry	setpwent	getpwent(3C)
set/clear realtime privileges	setrt	setrt(2)
set realtime privileged groups/	setrtgroups	setrtgroups(1M)
set realtime privileged groups/	setrtgroups	setrtgroups(2)
set realtime privileged users/	setrtusers	setrtusers(1M)
set realtime privileged users/	setrtusers	setrtusers(2)
manipulate signal	sets	sigsetops(2P)
get service entry	setservent	getservent(3N)
create a new session	setsid	setsid(2)
set CPU time slice size	setslice	setslice(2)
get and set options on sockets	setsockopt	getsockopt(2N)
get shadow password file entry	setspent	getspent(3X)
set the current value for a/	settimer	settimer(2)
	setting up an environment at login time	profile(4)
speed and terminal	settings used by getty	gettydefs(4)
set user and group IDs	setuid	setuid(2)
initialize system for first user	setup	setup(1M)
access utmp file entry	setutent	getut(3C)
assign buffering to a stream	setvbuf	setbuf(3S)
merge same lines of	several files or subsequent lines of one file	paste(1)
access long integer data in a/	sgetl	sputl(3X)
shell, the standard/restricted/	sh	sh(1)
get	shadow password file entry	getspent(3X)
write	shadow password file entry	putspent(3X)
create a	shared library	mkshlib(1)
control structures for	shared memory	shm(4)
	shared memory control operations	shmctl(2)
remove a message queue, semaphore set, or	shared memory ID	ipcrm(1)
close a	shared memory name server database	nsclose(3C)
create a	shared memory name server database	nscreat(3C)
delete a record from a	shared memory name server database	nsdelete(3C)
get a record from a	shared memory name server database	nsget(3C)
insert a record into a	shared memory name server database	nsinsert(3C)
map a name to a corresponding id in a	shared memory name server database	nsntoi(3C)
map an id to a corresponding name in a	shared memory name server database	nsiton(3C)

open a	shared memory name server database	nsopen(3C)
remove a	shared memory name server database	nsremove(3C)
	shared memory operations	shmop(2)
create	shared memory segment	shmget(1)
get	shared memory segment identifier	shmget(2)
remote	shell	remsh(1)
issue a	shell command	system(3S)
the bourne	shell command programming language	bsh(1)
	shell layer manager	shl(1)
pseudo	shell layer, console manager	conman(1)
	shell procedures for accounting	acctsh(1M)
extracts menus from labels stored in command	shell scripts	mkmenus(1M)
remote	shell server	remshd(1M)
programming language	shell, the standard/restricted command	ksh(1)
programming language	shell, the standard/restricted command and	sh(1)
shell layer manager	shl	shl(1)
control structures for shared/	shm	shm(4)
shared memory operations	shmat	shmop(2)
shared memory control operations	shmctl	shmctl(2)
shared memory operations	shmdt	shmop(2)
create shared memory segment	shmget	shmget(1)
get shared memory segment/	shmget	shmget(2)
	show all remote mounts	showmount(1M)
	show host status of local machines	ruptime(1N)
	show network status	netstat(1M)
	show socket configuration	sckconf(1M)
show all remote mounts	showmount	showmount(1M)
	shut down part of a full-duplex connection	shutdown(2N)
	shut down system, change system state	shutdown(1M)
shell procedures for accounting	shutacct	acctsh(1M)
shut down part of a full-duplex/	shutdown	shutdown(2N)
shut down system, change system/	shutdown	shutdown(1M)
	shutdown the system with extreme prejudice	halt(1M)
QWB multiprocessor system/	si	si(1M)
	side-by-side difference program	sdiff(1)
preprocess MC88100 assembly/	sifilter	sifilter(1)
examine or change signal action	sigaction	sigaction(2P)
manipulate signal sets	sigaddset	sigsetops(2P)
manipulate signal sets	sigdelset	sigsetops(2P)
manipulate signal sets	sigemptyset	sigsetops(2P)
manipulate signal sets	sigfillset	sigsetops(2P)
signal management	sighold	sigset(2)
signal management	sigignore	sigset(2)
manipulate signal sets	sigismember	sigsetops(2P)
non-local jumps	siglongjmp	sigsetjmp(3P)
	sign on	login(1)
return from a	signal	sigret(2)
specify what to do upon receipt of a	signal	signal(2)
specify what to do upon receipt/	signal	signal(2)
suspend process until	signal	pause(2)
wait for a	signal	sigsuspend(2P)
examine or change	signal action	sigaction(2P)
	signal management	sigset(2)
manipulate	signal sets	sigsetops(2P)
send a	signal to a process or a group of processes	kill(2)
examine and change blocked	signals	sigprocmask(2P)
examine pending	signals	sigpending(2P)

software	signals	ssignal(3C)
signal management	sigpause	sigset(2)
examine pending signals	sigpending	sigpending(2P)
examine and change blocked/	sigprocmask	sigprocmask(2P)
control structures for the/	sigp_t	evt(4)
signal management	sigrelse	sigset(2)
return from a signal	sigret	sigret(2)
signal management	sigset	sigset(2)
non-local jumps	sigsetjmp	sigsetjmp(3P)
wait for a signal	sigsuspend	sigsuspend(2P)
generate programs for	simple lexical tasks	lex(1)
perform a	simple loopback test on the SSP maintenance bus	mbloop(1M)
	simple random-number generator	rand(3C)
trigonometric functions	sin	trig(3M)
hyperbolic functions	sinh	sinh(3M)
print section sizes in bytes of/	size	size(1)
set CPU time slice	size	setslice(2)
summarize cache hit rate for each buffer	size	bfree(1M)
summarize file	size usage by file system	icount(1M)
print section	sizes in bytes of common object files	size(1)
recondition pty	slave devices	ptyfix(1M)
slice table editor	sledit	sledit(1M)
high resolution	sleep	nanosleep(2)
suspend execution for an/	sleep	sleep(1)
suspend execution for interval	sleep	sleep(3C)
set CPU time	slice size	setslice(2)
	slice table editor	sledit(1M)
find the	slot in the utmp file of the current user	ttyslot(3C)
core memory	smem	mem(7)
decode semaphore processor/	smp	smp(1M)
accept a connection on a	socket	accept(2N)
bind a name to a	socket	bind(2N)
create an endpoint for/	socket	socket(2N)
initiate a connection on a	socket	connect(2N)
listen for connections on a	socket	listen(2N)
receive data from a	socket	recv(2N)
send data over a	socket	send(2N)
show	socket configuration	sckconf(1M)
get	socket name	getsockname(2N)
bind a	socket to a privileged IP port	bindresvport(3N)
get and set options on	sockets	getsockopt(2N)
ioctl requests for	sockets	sockio(7N)
ioctl requests for sockets	sockio	sockio(7N)
users over a network running the TCP/IP-based	software write to all	rwall(1M)
multiprocessor a brief overview of the	software architecture of the Quadbus	realstar(5)
return some systemwide hardware and	software data	getsysdata(2)
return some hardware and	software data on a particular cpu	getcpudata(2)
	software loopback network interface	lo(7)
	software management package extraction facility	ixf(1M)
	software management package generation facility	igf(1M)
	software signals	ssignal(3C)
quicker	sort	qsort(3C)
sort and/or merge files	sort	sort(1)
topological	sort	tsort(1)
	sort and/or merge files	sort(1)
select or reject lines common to two	sorted files	comm(1)
binary search a	sorted table	bsearch(3C)

remake the binary system and commands from source code ..... :mk(1M)
from a common object file produce a C source listing with line numbers ..... list(1)
preallocate contiguous file space ..... prealloc(1R)
preallocate contiguous file space ..... prealloc(2)
truncate file space ..... ftruncate(2)
truncate file space ..... trunc(2)
change data segment space allocation ..... brk(2)
expand tabs to spaces, and vice versa ..... expand(1)
spawn getty to a remote terminal ..... ct(1C)
build special file ..... mknod(1M)
introduction to special files ..... intro(7)
terminals handle special functions of DASI 300 and 300s ..... 300(1)
handle special functions of the DASI 450 terminal ... 450(1)
make a directory, or a special or ordinary file ..... mknod(2)
various REAL/IX specific functions ..... sysrealix(2)
to ASCII language specific strings for converting times and dates . cftime(4)
SSP specific system service calls ..... sspsys(2)
format specification in text files ..... fspec(4)
receive any queued event from a specified list ..... evrcvl(2)
specify what to do upon receipt of a signal ... signal(2)
speed and terminal settings used by getty .... gettydefs(4)
set terminal type, modes, speed, and line discipline ..... getty(1M)
set terminal type, modes, speed, and line discipline ..... uugetty(1M)
find spelling errors spell ..... spell(1)
find spelling errors spellin ..... spell(1)
find spelling errors ..... spell(1)
context split ..... csplit(1)
split a file into pieces split ..... split(1)
split a file into pieces ..... split(1)
uucp spool directory clean-up ..... uucleanup(1M)
print formatted output sprintf ..... printf(3S)
access long integer data in a/ sputl ..... sputl(3X)
exponential, logarithm, power,/ sqrt ..... exp(3M)
exponential, logarithm, power, square root functions ..... exp(3M)
simple random-number generator srand ..... rand(3C)
generate uniformly distributed/ srand48 ..... drand48(3C)
convert formatted input sscanf ..... scanf(3S)
software signals ssignal ..... ssignal(3C)
SSP device driver interface to the MP system . qwbserv(7)
Quadbus device driver interface to SSP devices ..... qwbsys(7)
debugger SSP interface to the multiprocessor system ... nmi(1M)
perform a simple loopback test on the SSP maintenance bus ..... mbloop(1M)
perform one sync transfer on the SSP maintenance bus ..... mbs(1M)
log incoming SSP maintenance bus async messages. ..... mblogger(1M)
print out incoming SSP maintenance bus async messages. ..... mba(1M)
SSP maintenance bus interface ..... sspmb(7)
turn SSP maintenance bus polling on or off ..... mbpoll(1M)
QuadBus system interface to the SSP monitor ..... sspmon(1M)
run commands performed for the SSP multiuser environment ..... rcqwb(1M)
SSP specific system service calls ..... sspsys(2)
SSP maintenance bus interface sspmb ..... sspmb(7)
QuadBus system interface to the/ sspmon ..... sspmon(1M)
SSP specific system service/ sspsys ..... sspsys(2)
expand the stack region of the data segment ..... stkexp(2)
standard buffered input/output package ..... stdio(3S)
standard interprocess communication package stdipc(3C)
language shell, the standard/restricted command and programming sh(1)

language shell, the	standard/restricted command programming ..	ksh(1)
	start, stop, sample, or reset kernel profiling ...	kgmon(1M)
NFS, NIS, and REX	start/stop and configuration scripts	nfs(1M)
requests	start/stop the LP print service and move	lpsched(1M)
information about the system/	starter	starter(1)
shell procedures for accounting	startup	acctsh(1M)
data returned by stat(2) and/	stat	stat(5)
get file status	stat	stat(2)
data returned by	stat(2) and estat(2) system calls	stat(5)
get file system information	statfs	statfs(2)
NFS	statistics	nfsstat(1M)
display lock and semaphore	statistics	semstat(1M)
get file system	statistics	ustat(2)
prints out STREAMS	statistics	strstat(1M)
list file names and	statistics for a file system	ff(1M)
display	statistics for supported IP interfaces	ifstat(1M)
display system	statistics on a crt	systat(1)
get extended file	status	estat(2)
get file	status	stat(2)
report file system	status	fsstat(1M)
report inter-process communication facilities	status	ipcs(1)
report process	status	ps(1)
show network	status	netstat(1M)
stream	status inquiries	ferror(3S)
uucp	status inquiry and job control	uustat(1C)
show host	status of local machines	ruptime(1N)
print information about the	status of the LP print service	lpstat(1)
set/get Processor	Status Register	setpsr(2)
system	status server	rwhod(1M)
standard buffered input/output/	stdio	stdio(3S)
standard interprocess/	stdipc:	stdipc(3C)
set time	stime	stime(2)
expand the stack region of the/	stkexp	stkexp(2)
	stop all processes and turn off the power	powerdown(1M)
wait for child process to	stop or terminate	wait(2)
wait for child process to	stop or terminate	waitpid(2)
run commands performed to	stop the operating system	rc0(1M)
start,	stop, sample, or reset kernel profiling	kgmon(1M)
database subroutines	store	dbm(3N)
extracts menus from labels	stored in command shell scripts	mkmenus(1M)
print STREAMS trace messages	strace	strace(1M)
string operations	strcat	string(3C)
string operations	strchr	string(3C)
STREAMS error logger cleanup/	strclean	strclean(1M)
string operations	strcmp	string(3C)
string operations	strcpy	string(3C)
string operations	strcspn	string(3C)
string operations	strdup	string(3C)
assign buffering to a	stream	setbuf(3S)
close or flush a	stream	fclose(3S)
get a string from a	stream	gets(3S)
get character or word from a	stream	getc(3S)
get next message off a	stream	getmsg(2)
open a	stream	fopen(3S)
push character back into input	stream	ungetc(3S)
put a string on a	stream	puts(3S)
put character or word on a	stream	putc(3S)

reposition a file pointer in a stream ..... fseek(3S)
send a message on a stream ..... putmsg(2)
stream editor ..... sed(1)
stream status inquiries ..... ferror(3S)
routines for returning a stream to a remote command ..... rcmd(3N)
MVME350 Streamer Tape Controller VME Module Interface mvme350(7)
STREAMS ioctl commands streamio ..... streamio(7)
open any minor device on a STREAMS driver ..... clone(7)
STREAMS error logger cleanup program .... strclean(1M)
STREAMS error logger daemon ..... strerr(1M)
interface to STREAMS error logging and event tracing ... log(7)
STREAMS input/output multiplexing ..... poll(2)
STREAMS ioctl commands ..... streamio(7)
Transport Interface cooperating STREAMS module ..... timod(7)
Transport Interface read/write interface STREAMS module ..... tirdwr(7)
Internet IP protocol streams multiplexer module ..... ip(7N)
prints out STREAMS statistics ..... strstat(1M)
print STREAMS trace messages ..... strace(1M)
STREAMS error logger daemon strerr ..... strerr(1M)
convert date and time to string strftime ..... strftime(3C)
convert between long integer and base-64 ASCII string ..... a64l(3C)
convert date and time to string ..... ctime(3C)
convert date and time to string ..... strftime(3C)
convert floating-point number to string ..... ecvt(3C)
search a file for a character string ..... fgrep(1)
get a string from a stream ..... gets(3S)
calculate an integer value from a string of characters ..... getnum(3X)
put a string on a stream ..... puts(3S)
string operations ..... string(3C)
convert string to double-precision number ..... strtod(3C)
convert string to integer ..... strtol(3C)
display character strings in/ strings ..... strings(1)
language specific strings for converting times and dates to ASCII cftime(4)
display character strings in object files ..... strings(1)
strip symbol and line number/ strip ..... strip(1)
common object file strip symbol and line number information from a strip(1)
string operations strlen ..... string(3C)
string operations strncat ..... string(3C)
string operations strncmp ..... string(3C)
string operations strncpy ..... string(3C)
string operations strpbrk ..... string(3C)
string operations strrchr ..... string(3C)
string operations strspn ..... string(3C)
prints out STREAMS statistics strstat ..... strstat(1M)
convert string to/ strtod ..... strtod(3C)
string operations strtok ..... string(3C)
convert string to integer strtol ..... strtol(3C)
allocate a library structure ..... t_alloc(3N)
connected interrupt structure ..... cintrio(4)
free a library structure ..... t_free(3N)
identify processes using a file or file structure ..... fuser(1M)
the NIS database and directory structure ..... ypfiles(4N)
read initialize structures before requesting an asynchronous . arinit(2)
write initialize structures before requesting an asynchronous . awinit(2)
control structures for messages ..... msg(4)
control structures for shared memory ..... shm(4)
control structures for the common event mechanism .. evt(4)

control	structures for user-level semaphores	sem(4)
set the options for a terminal	stty	stty(1)
become super-user or another/	su	su(1M)
make a MSDOS	sub directory	mmd(1)
remove a MSDOS	sub directory	mrd(1)
database	subroutines	dbm(3N)
merge same lines of several files or	subsequent lines of one file	paste(1)
print checksum and block count/	sum	sum(1)
	summarize cache hit rate for each buffer size	bfree(1M)
	summarize disk usage	du(1M)
	summarize file size usage by file system	icount(1M)
command	summary from per-process accounting records	acctcms(1M)
update the	super block	sync(1M)
update	super block	sync(2)
internet	super-server	inetd(1M)
become	super-user or another user	su(1M)
get resolution	supported by nanosleep(2)	nanosleep_getres(2)
display statistics for	supported IP interfaces	ifstat(1M)
suspend the calling process	suspend	suspend(2)
	suspend execution for an interval	sleep(1)
	suspend execution for interval	sleep(3C)
	suspend process until signal	pause(2)
	suspend the calling process	suspend(2)
resume a	suspended process	resume(1R)
resume a	suspended process	resume(2)
host name lookup configuration/	svcorder	svcorder(4N)
swap bytes	swab	swab(3C)
swap administrative interface	swap	swap(1M)
	swap administrative interface	swap(1M)
	swap bytes	swab(3C)
	switch into a process	swtch(2)
switch into a process	swtch	swtch(2)
pseudo-device driver	sxt	sxt(7)
common object file strip	symbol and line number information from a	strip(1)
entry retrieve	symbol name for common object file symbol table	ldgetname(3X)
retrieve symbol 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)
read an indexed	symbol table entry of a common object file	ldtbread(3X)
common object file	symbol table format	syms(4)
kernel debugger	symbol table generator	unixsyms(1)
seek to the	symbol table of a common object file	ldtbseek(3X)
file header for	symbolic constants	unistd(4)
	symbolic debugger	sdb(1)
read value of a	symbolic link	readlink(2)
make	symbolic link to a file	symlink(2)
definitions of terms and	symbols	glossary(1)
make symbolic link to a file	symlink	symlink(2)
common object file symbol table/	syms	syms(4)
update super block	sync	sync(2)
update the super block	sync	sync(1M)
perform one	sync transfer on the SSP maintenance bus	mbs(1M)
	synchronise a memory object	msync(2)
correct the time to allow	synchronization of the system clock	adjtime(2)
disk	synchronize a file's in-core state with that on	fsync(2)
	synchronize transport library	t_sync(3N)
	synchronous I/O multiplexing	select(2)
menu interface to do system/	sysadm	sysadm(1M)

get configurable system/	sysconf	sysconf(2P)
db command to dump system/	sysdump	sysdump(1M)
get file system type information	sysfs	sysfs(2)
configure and generate the/	sysgen	sysgen(1M)
machine-specific functions	sysmot	sysmot(2)
various REAL/IX specific/	sysrealix	sysrealix(2)
display system statistics on a/	systat	systat(1)
RAM driver for the REAL/IX Operating	System general non-volatile	nvram(7)
SSP device driver interface to the MP	system	qwbserv(7)
call another UNIX	system	cu(1C)
configure and generate the operating	system	sysgen(1M)
construct a file	system	fs(1)
construct a file	system	mkfs(1M)
file transport program for the uucp	system	uucico(1M)
general ramdisk driver for the REAL/IX	System	ramdisk(7)
get name of current	system	uname(2)
interactive message processing	system	mailx(1)
issue a shell command	system	system(3S)
list file names and statistics for a file	system	ff(1M)
make literal copy of file	system	volcopy(1M)
mount a file	system	()
mount a file	system	mount(2)
mount and dismount file	system	mnt(1)
mount, unmount a diskette file	system	mountfsys(1M)
print name of current	system	uname(1)
run commands performed to stop the operating	system	rc0(1M)
set or print name of current host	system	hostname(1N)
summarize file size usage by file	system	icount(1M)
unmount a file	system	umount(2)
who is on the	system	who(1)
time a command; report process data and	system activity	timex(1)
	system activity report package	sar(1M)
	system activity reporter	sar(1)
devices administered by	System Administration	sa(7)
menu interface to do	system administration	sysadm(1M)
remake the binary	system and commands from source code	:mk(1M)
check file	system backup schedule	ckbupscd(1M)
file	system block analyzer	fsba(1M)
implementation-defined	system call	sys_local(2)
data returned by stat(2) and estat(2)	system calls	stat(5)
introduction to	system calls and error numbers	intro(2)
file	system check and interactive repair	chk(1)
	system clock	clock(7)
the time to allow synchronization of the	system clock correct	adjtime(2)
UNIX-to-UNIX	system command execution	uux(1C)
print	system configuration	prtconf(1M)
UNIX-to-UNIX	system copy	uucp(1C)
primitive	system data types	types(5)
SSP interface to the multiprocessor	system debugger	nmi(1M)
file	system debugger	fsdb(1M)
save the core image of a	system dump in a file	savecore(1M)
	system error messages	perror(3C)
public UNIX-to-UNIX	system file copy	uuto(1C)
information about the	system for beginning users	starter(1)
initialize	system for first user	setup(1M)
examine	system images	crash(1M)
file	system independent directory entry	dirent(4)

read directory entries and put in a file — system independent format ..... getdents(2)
get file — system information ..... statfs(2)
print various file — system information ..... fsinfo(1M)
QWB multiprocessor — system initialization ..... si(1M)
— system initialization procedures ..... bcheckrc(1M)
QuadBus — system interface to the SSP monitor ..... sspmon(1M)
fetch — system load ..... getsysload(2)
convert track numbers into file — system logical block numbers ..... fblkgen(1M)
db command to dump — system memory image to dump partition ..... sysdump(1M)
display file — system name ..... fstyp(1M)
check a file — system on a diskette ..... checkfsys(1M)
create a file — system on a diskette ..... makefsys(1M)
UNIX — system profiler ..... profiler(1M)
operating — system profiler ..... prf(7)
SSP specific — system service calls ..... sspsys(2)
shut down system, change — system state ..... shutdown(1M)
get file — system statistics ..... ustat(2)
display — system statistics on a crt ..... systat(1)
report file — system status ..... fsstat(1M)
— system status server ..... rwhod(1M)
NFS remotely mounted file — system table ..... rmtab(4N)
mounted file — system table ..... mnttab(4)
set default — system time zone ..... timezone(4)
get file — system type information ..... sysfs(2)
get configurable — system variables ..... sysconf(2P)
format of — system volume ..... fs(4)
try to contact remote — system with debugging on ..... uutry(1M)
shutdown the — system with extreme prejudice ..... halt(1M)
shut down — system, change system state ..... shutdown(1M)
variables — system-wide framed access command environment env(4)
get the current value for a — system-wide realtime timer ..... gettimer(2)
set the current value for a — system-wide realtime timer ..... settimer(2)
the resolution and maximum time values for a — system-wide realtime timer get ..... restimer(2)
format of time for — system-wide realtime timers ..... timestruc(4)
automatically mount NFS file — systems ..... automount(1M)
back up and restore files and/or file — systems ..... br(1M)
change performance tunings of file — systems ..... tunefs(1M)
check and repair file — systems ..... fsck(1M)
keep track of remotely mounted file — systems ..... mount(3N)
mount, unmount multiple file — systems ..... mountall(1M)
mount, unmount multiple file — systems ..... nmountall(1M)
provide labels for file — systems ..... labelit(1M)
mount and unmount file — systems and remote resources ..... mount(1M)
copy file — systems for optimal access time ..... dcopy(1M)
list of file — systems processed by fsck and ncheck ..... checklist(4)
return some — systemwide hardware and software data ..... getsysdata(2)
system error messages — sys_errlist ..... perror(3C)
implementation-defined system/ — sys_local ..... sys_local(2)
system error messages — sys_nerr ..... perror(3C)
NFS remotely mounted file system — table ..... rmtab(4N)
binary search a sorted — table ..... bsearch(3C)
establish mount — table ..... setmnt(1M)
mounted file system — table ..... mnttab(4)
slice — table editor ..... sledit(1M)
symbol name for common object file symbol — table entry retrieve ..... ldgetname(3X)
compute the index of a symbol — table entry of a common object file ..... ldtbindex(3X)
read an indexed symbol — table entry of a common object file ..... ldtbread(3X)

common object file symbol	table format	syms(4)
kernel debugger symbol	table generator	unixsyms(1)
seek to the symbol	table of a common object file	ldtbseek(3X)
translation	table to handle long map names	ypmapxlate(4N)
character classification and conversion	tables generate	chrtbl(1M)
manage hash search	tables	hsearch(3C)
manually manipulate the routing	tables	route(1M)
set tabs on a terminal	tabs	tabs(1)
set	tabs on a terminal	tabs(1)
expand	tabs to spaces, and vice versa	expand(1)
maintain a	tags file for a C program	ctags(1)
deliver the last part of a file	tail	tail(1)
trigonometric functions	tan	trig(3M)
hyperbolic functions	tanh	sinh(3M)
recover files from a backup	tape	frec(1M)
magnetic	tape control	mt(1)
driver for the MVME355 9-Track	Tape Controller	mvme355(7)
MVME350 Streamer	Tape Controller VME Module Interface	mvme350(7)
	tape device control program	tapectl(1M)
	tape file archiver	tar(1)
realtime SCSI autoconfiguration/ general	tape implementation details used by REAL/IX	scsitape(7)
realtime SCSI/ general 9-track	tape implementation details used by REAL/IX	scsi9track(7)
tape device control program	tapectl	tapectl(1M)
tape file archiver	tar	tar(1)
run a command on a particular/	target	target(1R)
	target the calling process to some cpu	targetcpu(2)
target the calling process to/	targetcpu	targetcpu(2)
generate programs for simple lexical	tasks	lex(1)
remove nroff/troff,	tbl, and eqn constructs	deroff(1)
line control functions	tcdrain	tcdrain(3P)
line control functions	tcflow	tcdrain(3P)
line control functions	tcflush	tcdrain(3P)
get and set terminal state	tcgetattr	tcgetattr(3P)
get distinguished process group/	tcgetpgrp	tcgetpgrp(3P)
Internet Transmission Control/	tcp	tcp(7)
write to all users over a network running the	TCP/IP-based software	rwall(1M)
line control functions	tcsendbreak	tcdrain(3P)
get and set terminal state	tcsetattr	tcgetattr(3P)
set distinguished process group/	tcsetpgrp	tcsetpgrp(3P)
manage binary search trees	tdelete	tsearch(3C)
pipe fitting	tee	tee(1)
paginator for the	Tektronix 4014 terminal	4014(1)
process control initialization	telinit	init(1M)
directory operations	telldir	directory(3X)
user interface to the TELNET/	telnet	telnet(1N)
user interface to the	TELNET protocol	telnet(1N)
DARPA	TELNET protocol server	telnetd(1M)
DARPA TELNET protocol server	telnetd	telnetd(1M)
create a name for a temporary/	tempnam	tmpnam(3S)
create a name for a	temporary file	tmpnam(3S)
create a	temporary file	tmpfile(3S)
conventional names for terminals	term	term(5)
format of compiled term file.	term	term(4)
format of compiled	term file.	term(4)
convert a	termcap description into a terminfo description	captoinfo(1M)
find name of a	terminal	ttyname(3C)
generate file name for	terminal	ctermid(3S)

get the name of the terminal ..... tty(1)
handle special functions of the DASI 450 terminal ..... 450(1)
paginator for the Tektronix 4014 terminal ..... 4014(1)
reserve a terminal ..... lock(1)
set tabs on a terminal ..... tabs(1)
set the options for a terminal ..... stty(1)
spawn getty to a remote terminal ..... ct(1C)
terminal capability data base ..... terminfo(4)
select terminal filter ..... greek(1)
controlling terminal interface ..... tty(7)
general terminal interface ..... termio(7)
general terminal interface ..... termios(7)
establish an out-going terminal line connection ..... dial(3C)
initialize a terminal or query terminfo database ..... tput(1)
package terminal screen handling and optimization ..... curses(3X)
make typescript of terminal session ..... script(1)
speed and terminal settings used by getty ..... gettydefs(4)
get and set terminal state ..... tcgetattr(3P)
discipline set terminal type, modes, speed, and line ..... getty(1M)
discipline set terminal type, modes, speed, and line ..... uugetty(1M)
conventional names for terminals ..... term(5)
handle special functions of DASI 300 and 300s terminals ..... 300(1)
wait for child process to stop or terminate ..... wait(2)
wait for child process to stop or terminate ..... waitpid(2)
terminate a process ..... kill(1)
terminate process ..... exit(2)
terminal capability data base terminfo ..... terminfo(4)
terminfo compiler ..... tic(1M)
initialize a terminal or query terminfo database ..... tput(1)
convert a termcap description into a terminfo description ..... captoinfo(1M)
compare or print out terminfo descriptions ..... infocmp(1M)
general terminal interface termio ..... termio(7)
general terminal interface termios ..... termios(7)
definitions of terms and symbols ..... glossary(1)
condition evaluation command test ..... test(1)
perform a simple loopback test on the SSP maintenance bus ..... mbloop(1M)
text editor ..... ed(1)
text editor ..... ex(1)
text editor (variant of ex for casual users) ..... edit(1)
change the format of a text file ..... newform(1)
format specification in text files ..... fspec(4)
lock process, text, or data in memory ..... plock(2)
manage binary search trees tfind ..... tsearch(3C)
trivial file transfer program tftp ..... tftp(1M)
DARPA Trivial File Transfer/ tftpd ..... tftpd(1M)
terminfo compiler tic ..... tic(1M)
get time time ..... time(2)
time a command time ..... time(1)
and maximum time value of process interval timer get resolution ..... resabs(2)
get a unique identifier for a process interval timer ..... gettimerid(2)
get the current value for a process interval timer ..... getinterval(2)
get value of interval timer ..... getitimer(2)
aximum time values for a system-wide realtime timer get the resolution and ..... restimer(2)
set the expiration time of a process interval timer ..... absinterval(2)
set value of interval timer ..... setitimer(2)
the current value for a system-wide realtime timer get ..... gettimer(2)
the current value for a system-wide realtime timer set ..... settimer(2)

release a process interval timer identifier ..... reltimerid(2)
format of time for system-wide realtime timers ..... timestruc(4)
format of time values for process interval timers ..... itimerstruc(4)
get process and child process times ..... times(2)
get process and child process/ times ..... times(2)
set file access and modification times ..... utime(2)
language specific strings for converting times and dates to ASCII ..... cftime(4)
update access and modification times of a file ..... touch(1)
format of time for system-wide/ timestruc ..... timestruc(4)
time a command; report process/ timex ..... timex(1)
set default system time zone timezone ..... timezone(4)
Transport Interface cooperating/ timod ..... timod(7)
Transport Interface read/write/ tirdwr ..... tirdwr(7)
create a temporary file tmpfile ..... tmpfile(3S)
create a name for a temporary/ tmpnam ..... tmpnam(3S)
initiate pipe to/from a process ..... popen(3S)
character classification macros toascii ..... ctype(3C)
translate characters toascii ..... conv(3C)
character classification macros tolower ..... ctype(3C)
translate characters tolower ..... conv(3C)
topological sort ..... tsort(1)
merge or add total accounting files ..... acctmerg(1M)
update access and modification/ touch ..... touch(1)
character classification macros toupper ..... ctype(3C)
translate characters toupper ..... conv(3C)
Transport Provider Interface/ tpid ..... tpid(1M)
initialize a terminal or query/ tput ..... tput(1)
translate characters tr ..... tr(1)
process trace ..... ptrace(2)
change the qwbd daemon debug trace flags ..... qwbdflags(1M)
print STREAMS trace messages ..... strace(1M)
interface to STREAMS error logging and event tracing ..... log(7)
convert bad track list from one format to another ..... xformtrk(1M)
numbers convert track numbers into file system logical block .. fblkgen(1M)
keep track of remotely mounted file systems ..... mount(3N)
here transfer an NIS map from some NIS server to . ypxfr(1M)
perform one sync transfer on the SSP maintenance bus ..... mbs(1M)
ARPANET file transfer program ..... ftp(1N)
trivial file transfer program ..... tftp(1M)
DARPA Internet File Transfer Protocol server ..... ftpd(1M)
DARPA Trivial File Transfer Protocol server ..... tftpd(1M)
translate characters ..... conv(3C)
translate characters ..... tr(1)
format of manual page translation files ..... man(4)
translation table to handle long map names .. ypmapxlate(4N)
Internet Transmission Control Protocol ..... tcp(7)
bind an address to a transport endpoint ..... t_bind(3N)
close a transport endpoint ..... t_close(3N)
disable a transport endpoint ..... t_unbind(3N)
establish a transport endpoint ..... t_open(3N)
look at the current event on a transport endpoint ..... t_look(3N)
manage options for a transport endpoint ..... t_optmgmt(3N)
Transport Interface cooperating STREAMS module timod(7)
STREAMS module Transport Interface read/write interface ..... tirdwr(7)
synchronize transport library ..... t_sync(3N)
the scheduler for the uucp file transport program ..... uusched(1M)
file transport program for the uucp system ..... uucico(1M)

	Transport Provider Interface daemon	tpid(1M)
establish a connection with another	transport user	t_connect(3N)
walk a file	tree	ftw(3C)
Motorola enhanced file	tree walk	motftw(3C)
manage binary search	trees	tsearch(3C)
	trigonometric functions	trig(3M)
	trivial file transfer program	tftp(1M)
DARPA	Trivial File Transfer Protocol server	tftpd(1M)
provide truth values	true	true(1)
truncate file space	trunc	trunc(2)
	truncate file space	ftruncate(2)
	truncate file space	trunc(2)
hosts with equivalent users, i.e., "trusted"	hosts names of	hosts.equiv(4N)
get processor type	truth value	machid(1)
provide	truth values	true(1)
	try to contact remote system with debugging on	uutry(1M)
manage binary search trees	tsearch	tsearch(3C)
topological sort	tsort	tsort(1)
convert and copy a file	tt	tt(1)
controlling terminal interface	tty	tty(7)
get the name of the terminal	tty	tty(1)
find name of a terminal	ttyname	ttyname(3C)
find the slot in the utmp file/	ttyslot	ttyslot(3C)
change performance tunings of/	tunefs	tunefs(1M)
change performance	tunings of file systems	tunefs(1M)
stop all processes and	turn off the power	powerdown(1M)
	turn SSP maintenance bus polling on or off	mbpoll(1M)
shell procedures for accounting	turnacct	acctsh(1M)
manage binary search trees	twalk	tsearch(3C)
converts from a	two- to a one-password file scheme.	pwunconv(1M)
determine file	type	file(1)
get file system	type information	sysfs(2)
get processor	type truth value	machid(1)
set terminal	type, modes, speed, and line discipline	getty(1M)
set terminal	type, modes, speed, and line discipline	uugetty(1M)
primitive system data types	types	types(5)
primitive system data	types	types(5)
make	typescript of terminal session	script(1)
convert date and time to string	tzset	ctime(3C)
accept a connect request	t_accept	t_accept(3N)
allocate a library structure	t_alloc	t_alloc(3N)
bind an address to a transport/	t_bind	t_bind(3N)
close a transport endpoint	t_close	t_close(3N)
establish a connection with/	t_connect	t_connect(3N)
produce error message	t_error	t_error(3N)
free a library structure	t_free	t_free(3N)
get protocol-specific service/	t_getinfo	t_getinfo(3N)
get the current state	t_getstate	t_getstate(3N)
listen for a connect request	t_listen	t_listen(3N)
look at the current event on a/	t_look	t_look(3N)
establish a transport endpoint	t_open	t_open(3N)
manage options for a transport/	t_optmgmt	t_optmgmt(3N)
receive data or expedited data/	t_rcv	t_rcv(3N)
receive the confirmation from a/	t_rcvconnect	t_rcvconnect(3N)
retrieve information from/	t_rcvdis	t_rcvdis(3N)
acknowledge receipt of an/	t_rcvrel	t_rcvrel(3N)
receive a data unit	t_rcvudata	t_rcvudata(3N)

receive a unit data error/	t_rcvuderr	t_rcvuderr(3N)
send data or expedited data/	t_snd	t_snd(3N)
send user-initiated disconnect/	t_snddis	t_snddis(3N)
initiate an orderly release	t_sndrel	t_sndrel(3N)
send a data unit	t_sndudata	t_sndudata(3N)
synchronize transport library	t_sync	t_sync(3N)
disable a transport endpoint	t_unbind	t_unbind(3N)
get processor type truth value	u3b	machid(1)
get processor type truth value	u3b2	machid(1)
get processor type truth value	u3b5	machid(1)
administrative control	uadmin	uadmin(1M)
administrative control	uadmin	uadmin(2)
Internet User Datagram Protocol	udp	udp(7)
get name from	UID	getpw(3C)
get shadow password file entry	ulckpwdf	getspent(3X)
get and set user limits	ulimit	ulimit(2)
set and get file creation mask	umask	umask(2)
set file-creation mode mask	umask	umask(1)
mount and dismount file system	umnt	mnt(1)
mount and unmount file systems/	umount	mount(1M)
unmount a file system	umount	umount(2)
mount, unmount multiple file/	umountall	mountall(1M)
mount, unmount a diskette file/	umountfsys	mountfsys(1M)
get name of current system	uname	uname(2)
print name of current system	uname	uname(1)
	undo a previous get of an SCCS file	unget(1)
expand tabs to spaces, and vice/	unexpand	expand(1)
undo a previous get of an SCCS/	unget	unget(1)
push character back into input/	ungetc	ungetc(3S)
remove ifdef'ed lines	unifdef	unifdef(1)
generate	uniformly distributed pseudo-random numbers	drand48(3C)
report repeated lines in a file	uniq	uniq(1)
make a	unique file name	mktemp(3C)
get a	unique identifier for a process interval timer	gettimerid(2)
get	unique identifier of current host	gethostid(3N)
file header for symbolic/	unistd	unistd(4)
receive a data	unit	t_rcvudata(3N)
send a data	unit	t_sndudata(3N)
receive a	unit data error indication	t_rcvuderr(3N)
conversion program	units	units(1)
read (copy) a MSDOS file to	Unix	mread(1)
write (copy) a	Unix file to MSDOS	mwrite(1)
call another	UNIX system	cu(1C)
	UNIX system profiler	profiler(1M)
	UNIX-to-UNIX system command execution	uux(1C)
	UNIX-to-UNIX system copy	uucp(1C)
public	UNIX-to-UNIX system file copy	uuto(1C)
kernel debugger symbol table/	unixsyms	unixsyms(1)
link and unlink files and/	unlink	link(1M)
remove directory entry	unlink	unlink(2)
link and	unlink files and directories	link(1M)
	unlock a binary semaphore	bsunlk(3C)
	unmap previously mapped addresses	munmap(2)
mount,	unmount a diskette file system	mountfsys(1M)
	unmount a file system	umount(2)
mount and	unmount file systems and remote resources	mount(1M)
mount,	unmount multiple file systems	mountall(1M)

mount, unmount multiple file systems ..... nmountall(1M)
compress and expand files unpack ..... pack(1)
suspend process until signal ..... pause(2)
linear search and update ..... lsearch(3C)
/etc/passwd install and update /etc/shadow with information from ... pwconv(1M)
update access and modification times of a file . touch(1)
update super block ..... sync(2)
update the super block ..... sync(1M)
update user password in NIS ..... yppasswd(3N)
maintain, update, and regenerate groups of programs ... make(1)
specify what to do upon receipt of a signal ..... signal(2)
retrieve a command description/ usage ..... usage(1)
summarize disk usage ..... du(1M)
summarize file size usage by file system ..... icount(1M)
retrieve a command description and usage examples ..... usage(1)
become super-user or another user ..... su(1M)
establish a connection with another transport user ..... t_connect(3N)
find the slot in the utmp file of the current user ..... ttyslot(3C)
get character login name of the user ..... cuserid(3S)
initialize system for first user ..... setup(1M)
return login name of user ..... logname(3X)
write to another user ..... write(1)
set user and group IDs ..... setuid(2)
print user and group IDs and names ..... id(1M)
user crontab file ..... crontab(1)
Internet User Datagram Protocol ..... udp(7)
user environment ..... environ(5)
generate disk accounting data by user ID ..... diskusg(1M)
print effective current user id ..... whoami(1)
user information lookup program ..... finger(1)
remote user information server ..... fingerd(1M)
user interface to the TELNET protocol ..... telnet(1N)
get and set user limits ..... ulimit(2)
get yes/no response from user or check answer to question ..... chkyn(1)
update user password in NIS ..... yppasswd(3N)
group IDs get real user, effective and .. getuid(2)
IDs get real user, effective user, real group, and effective group ..... getuid(2)
send user-initiated disconnect request ..... t_snddis(3N)
control structures for user-level semaphores ..... sem(4)
users user-specified file of equivalent hosts and ..... rhosts(4)
information about the system for beginning users ..... starter(1)
user-specified file of equivalent hosts and users ..... rhosts(4)
write to all users ..... wall(1)
set realtime privileged users list ..... setrtusers(1M)
set realtime privileged users list ..... setrtusers(2)
view the realtime privileged users list ..... rtusers(1)
send mail to users or read mail ..... mail(1)
software write to all users over a network running the TCP/IP-based rwall(1M)
text editor (variant of ex for casual users) ..... edit(1)
names of hosts with equivalent users, i.e., "trusted" hosts ..... hosts.equiv(4N)
identify processes using a file or file structure ..... fuser(1M)
search a file for a pattern using full regular expressions ..... egrep(1)
identify a command using keywords ..... locate(1)
assistance using REAL/IX commands ..... assist(1)
get file system statistics ustat ..... ustat(2)
file used by the value-added disk access utilities permissions ..... filesys(4)
or framed access command environment (FACE) utilities alias file ..... alias(4)

VLAN-E diagnostic utility ..... vlandiag(1M)
backup and restore utility ..... bru(1)
set file access and/ utime ..... utime(2)
utmp and wtmp entry formats utmp ..... utmp(4)
utmp and wtmp entry formats ..... utmp(4)
access utmp file entry ..... getut(3C)
find the slot in the utmp file of the current user ..... ttyslot(3C)
access utmp file entry utmpname ..... getut(3C)
try to contact remote system/ utry ..... uutry(1M)
check the uucp directories and/ uucheck ..... uucheck(1M)
file transport program for the/ uucico ..... uucico(1M)
uucp spool directory clean-up uucleanup ..... uucleanup(1M)
UNIX-to-UNIX system copy uucp ..... uucp(1C)
check the uucp directories and permissions file ..... uucheck(1M)
the scheduler for the uucp file transport program ..... uusched(1M)
uucp spool directory clean-up ..... uucleanup(1M)
uucp status inquiry and job control ..... uustat(1C)
file transport program for the uucp system ..... uucico(1M)
set terminal type, modes,/ uugetty ..... uugetty(1M)
UNIX-to-UNIX system copy uulog ..... uucp(1C)
UNIX-to-UNIX system copy uuname ..... uucp(1C)
public UNIX-to-UNIX system file/ uupick ..... uuto(1C)
the scheduler for the uucp file/ uusched ..... uusched(1M)
uucp status inquiry and job/ uustat ..... uustat(1C)
public UNIX-to-UNIX system file/ uuto ..... uuto(1C)
UNIX-to-UNIX system command/ uux ..... uux(1C)
execute remote command requests uuxqt ..... uuxqt(1M)
validate SCCS file val ..... val(1)
validate SCCS file ..... val(1)
get processor type truth value ..... machid(1)
return integer absolute value ..... abs(3C)
get the current value for a process interval timer ..... getinterval(2)
get the current value for a system-wide realtime timer ..... gettimer(2)
set the current value for a system-wide realtime timer ..... settimer(2)
return value for environment name ..... getenv(3C)
calculate an integer value from a string of characters ..... getnum(3X)
floor, ceiling, remainder, absolute value functions ..... floor(3M)
read value of a symbolic link ..... readlink(2)
get value of interval timer ..... getitimer(2)
set value of interval timer ..... setitimer(2)
print the value of one or more keys from a NIS map ... ypmatch(1N)
get resolution and maximum time value of process interval timer ..... resabs(2)
get or set the value of the output and input baud rate ..... cfgetospeed(3P)
change or add value to environment ..... putenv(3C)
permissions file used by the value-added disk access utilities ..... filesys(4)
machine-dependent values values ..... values(5)
machine-dependent values values ..... values(5)
provide truth values ..... true(1)
convert values between host and network byte order .. byteorder(3N)
get the resolution and maximum time values for a system-wide realtime timer ..... restimer(2)
format of time values for process interval timers ..... itimerstruc(4)
print values in a NIS database ..... ypcat(1N)
handle variable argument list varargs ..... varargs(5)
print formatted output of a varargs argument list ..... vprintf(3S)
handle variable argument list ..... varargs(5)
get configurable pathname variables ..... pathconf(2P)
get configurable system variables ..... sysconf(2P)

/stem-wide framed access command environment variables ..... env(4)
print various file system information ..... fsinfo(1M)
various REAL/IX specific functions ..... sysrealix(2)
get processor type truth value vax ..... machid(1)
version control vc ..... vc(1)
convert a virtual to a physical/ vcpuno ..... vcpuno(2)
get option letter from argument vector ..... getopt(3C)
screen-oriented (visual)/ vedit ..... vi(1)
verify program assertion ..... assert(3X)
expand tabs to spaces, and vice versa ..... expand(1)
version control ..... vc(1)
what version of a YP map is at a YP server host ... yppoll(1M)
get a version of an SCCS file ..... get(1)
compare two versions of an SCCS file ..... sccsdiff(1)
print formatted output of a/ vfprintf ..... vprintf(3S)
screen-oriented (visual)/ vi ..... vi(1)
expand tabs to spaces, and vice versa ..... expand(1)
screen-oriented (visual)/ view ..... vi(1)
view the realtime privileged groups list ..... rtgroups(1)
view the realtime privileged users list ..... rtusers(1)
file perusal filter for crt viewing ..... more(1)
convert a virtual to a physical cpu number ..... vcpuno(2)
driver for the MODCOMP VLAN-E/ vlan ..... vlan-e(7)
VLAN-E diagnostic utility ..... vlandiag(1M)
driver for the MODCOMP VLAN-E Intelligent Ethernet Controller ..... vlan-e(7)
VLAN-E diagnostic utility vlandiag ..... vlandiag(1M)
MVME332XT communication controller VME module interface ..... mvme332xt(7)
MVME350 Streamer Tape Controller VME Module Interface ..... mvme350(7)
make literal copy of file system volcopy ..... volcopy(1M)
format of system volume ..... fs(4)
voluntarily give up CPU ..... relinquish(2)
print formatted output of a/ vprintf ..... vprintf(3S)
print formatted output of a/ vsprintf ..... vprintf(3S)
await completion of process wait ..... wait(1)
wait for child process to stop/ wait ..... wait(2)
wait for a connected interrupt ..... cisema(2)
wait for a signal ..... sigsuspend(2P)
wait for child process to stop or terminate .... wait(2)
wait for child process to stop or terminate .... waitpid(2)
wait for child process to stop/ waitpid ..... waitpid(2)
Motorola enhanced file tree walk ..... motftw(3C)
walk a file tree ..... ftw(3C)
write to all users wall ..... wall(1)
word count wc ..... wc(1)
identify SCCS files what ..... what(1)
who is on the system who ..... who(1)
who's logged in on local machines ..... rwho(1N)
print effective current user id whoami ..... whoami(1)
who is doing what whodo ..... whodo(1M)
fold long lines for finite width output device ..... fold(1)
profile within a function ..... prof(5)
word count ..... wc(1)
get character or word from a stream ..... getc(3S)
put character or word on a stream ..... putc(3S)
change working directory ..... cd(1)
change working directory ..... chdir(2)
get path-name of current working directory ..... getcwd(3C)

	working directory name	pwd(1)
structures before requesting an asynchronous	write initialize	awinit(2)
write to a file	write	write(2)
write to another user	write	write(1)
	write (copy) a Unix file to MSDOS	mwrite(1)
	write password file entry	putpwent(3C)
	write shadow password file entry	putspent(3X)
	write to a file	write(2)
	write to all users	wall(1)
TCP/IP-based software	write to all users over a network running the	rwall(1M)
	write to another user	write(1)
	write to file in an asynchronous manner	awrite(2)
control	write/execute attributes of memory	memctl(2)
do multiple	writes from a file	writev(2)
do multiple writes from a file	writev	writev(2)
write to a file	writev	write(2)
open for reading or	writing	open(2)
utmp and wtmp entry formats	wtmp	utmp(4)
utmp and	wtmp entry formats	utmp(4)
manipulate connect accounting/	wtmpfix	fwtmp(1M)
construct argument list(s) and/	xargs	xargs(1)
patch	XC88100 COFF executables to avoid fdiv deadlock	fdivfix(1)
library routines for external/	xdr	xdr(3N)
convert bad track list from one/	xformtrk	xformtrk(1M)
Bessel functions	y0	bessel(3M)
Bessel functions	y1	bessel(3M)
yet another compiler-compiler	yacc	yacc(1)
question get	yes/no response from user or check answer to	chkyn(1)
	yet another compiler-compiler	yacc(1)
Bessel functions	yn	bessel(3M)
NFS, NIS, and REX start/stop/	yp	nfs(1M)
what version of a	YP map is at a YP server host	yppoll(1M)
what version of a YP map is at a	YP server host	yppoll(1M)
NIS server and binder processes	ypbind	ypserv(1M)
print values in a NIS database	ypcat	ypcat(1N)
NIS client interface	ypclnt	ypclnt(3N)
print the number of days since/	ypday	ypday(1M)
NIS client interface	yperr_string	ypclnt(3N)
the NIS database and directory/	ypfiles	ypfiles(4N)
build and install NIS database	ypinit	ypinit(1M)
rebuild NIS database	ypmake	ypmake(1M)
translation table to handle/	ypmapxlate	ypmapxlate(4N)
print the value of one or more/	ypmatch	ypmatch(1N)
password file management for/	yppassmgmt	yppassmgmt(1M)
change login password in NIS	yppasswd	yppasswd(1N)
update user password in NIS	yppasswd	yppasswd(3N)
server for modifying NIS/	yppasswdd	yppasswdd(1M)
what version of a YP map is at/	yppoll	yppoll(1M)
NIS client interface	ypprot_err	ypclnt(3N)
force propagation of a changed/	yppush	yppush(1M)
NIS server and binder processes	ypserv	ypserv(1M)
point **ypbind** at a/	ypset	ypset(1M)
which host is the NIS server or/	ypwhich	ypwhich(1N)
transfer an NIS map from some/	ypxfr	ypxfr(1M)
NIS client interface	yp_all	ypclnt(3N)
NIS client interface	yp_bind	ypclnt(3N)
NIS client interface	yp_first	ypclnt(3N)

NIS client interface	yp_get_default_domain	ypclnt(3N)
NIS client interface	yp_master	ypclnt(3N)
NIS client interface	yp_match	ypclnt(3N)
NIS client interface	yp_next	ypclnt(3N)
NIS client interface	yp_order	ypclnt(3N)
NIS client interface	yp_unbind	ypclnt(3N)
set default system time	zone	timezone(4)
configure or display	*tty* ports	portconfig(1M)
point	**ypbind** at a particular server	ypset(1M)
set	CPU time slice size	setslice(2)
free internal resources for asynchronous	I/O from the process	arwfree(2)
effective user, real group, and effective group	IDs get real user,	getuid(2)
get process, process group, and parent process	IDs	getpid(2)
print user and group	IDs and names	id(1M)
patch program with	**NULL** pointer dereference bug	dnp(1)
terminate process	_exit	exit(2)
character classification macros	_tolower	ctype(3C)
translate characters	_tolower	conv(3C)
character classification macros	_toupper	ctype(3C)
translate characters	_toupper	conv(3C)

RECEIVED APR 2 8 1993

NAME

intro – introduction to commands and application programs

DESCRIPTION

This section describes, in alphabetical order, commands available with the REAL/IX operating system. Certain distinctions of purpose are made in the headings.

The following Utility packages are part of the system:

Basic Networking Utilities (BNU)
Directory and File Management Utilities
Editing Utilities
Essential Utilities
Help Utilities
Inter-process Communications (IPC)
Line Printer Spooling Utilities
Performance Measurement Utilities
Security Administration Utilities
Spell Utilities
User Environment Utilities
Networking Support Utilities
Remote File Sharing Utilities (RFS)

Manual Page Command Syntax

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

name [*–option...*] [*cmdarg...*]

where:

[ ]	Surround an *option* or *cmdarg* that is not required.
...	Indicates multiple occurrences of the *option* or *cmdarg*.
name	The name of an executable file.
option	(Always preceded by a "–".) *noargletter*... or, *argletter optarg*[,...]
noargletter	A single letter representing an option without an option-argument. Note that more than one *noargletter* option can be grouped after one "–" (see Rule 5).
argletter	A single letter representing an option requiring an option-argument.
optarg	An option-argument (character string) satisfying a preceding *argletter*. Note that groups of *optargs* following an *argletter* must be separated by commas, or separated by white space and quoted (see Rule 8).
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

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

DIAGNOSTICS

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

WARNINGS

Some commands produce unexpected results when processing files containing **NULL** characters. These commands often treat text input lines as strings and therefore become confused upon encountering a **NULL** character (the string terminator) within a line.

NAME

intro – introduction to commands and application programs

DESCRIPTION

This section describes, in alphabetical order, commands available with the SYSTEM V/88 Network Services Extension (NSE). Certain distinctions of purpose are made in the headings.

Manual Page Command Syntax

Unless otherwise noted, commands described in the *SYNOPSIS* section of a manual page accept options and other arguments according to the following syntax and are interpreted as explained below:

name [*−option...*] [*cmdarg...*]

where:

[ ]
Surround an *option* or *cmdarg* that is not required.

...
Indicates multiple occurrences of the *option* or *cmdarg*.

name
The name of an executable file.

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

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

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

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

cmdarg
Path name (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

getopts(1).
exit(2), wait(2), getopt(3C) in the *Programmer's Reference Manual.*

DIAGNOSTICS

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

WARNINGS

Some commands produce unexpected results when processing files containing null characters. These commands often treat text input lines as strings and therefore become confused upon encountering a null character (the string terminator) within a line.

NAME

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

SYNOPSIS

300 [**+12**] [**-n**] [**-d***t,l,c*]

300s [**+12**] [**-n**] [**-d***t,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. In the following discussion of the *300* command, it should be noted that unless your system contains the DOCUMENTER'S WORKBENCH Software, references to certain commands (e.g., *nroff*, *neqn*, *eqn*) will not work. It also attempts to draw Greek letters and other special symbols. It permits convenient use of 12 pitch text and reduces printing time 5 to 70%. The *300* command can be used to print equations neatly, in the sequence:

neqn file ... | nroff | 300

CAUTION. Make sure the **PLOT** switch on your terminal is 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 Per Inch (LPI) text. DASI 300 terminals normally allow only two combinations: 10 pitch, 6 LPI, or 12 pitch, 8 LPI. To obtain the 12 pitch, 6 LPI combination, you 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 linefeed requires 8 increments, while a 12 pitch linefeed needs only 6. The first digit of *n* overrides the default value, which allows 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**. You can also obtain appropriate half-lines for 12 pitch, 8 LPI mode by using the option **-3** alone by setting 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 **NULL**s 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 linefeed 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 **LINEFEED** 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), stty(1), tabs(1).

BUGS

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

If your output contains Greek and/or reverse linefeeds, 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 linefeeds.

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 newline (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:

–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* [*i* or *l*]
Set page length to *L*. *L* accepts the scale factors *i* (inches) and *l* (lines). Default is lines.

It should be noted that unless your system contains the DOCUMENTER'S WORKBENCH Software, references to certain commands (e.g., *troff*, *neqn*, *eqn*) do not pertain.

SEE ALSO

pr(1).

NAME

450 – handle special functions of the DASI 450 terminal

SYNOPSIS

450

DESCRIPTION

The *450* command supports special functions of, 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). It should be noted that, unless your system contains DOCUMENTER'S WORKBENCH Software, certain commands (e.g., *eqn*, *nroff*, *tbl*) will not work. Use *450* to print equations neatly, in the sequence:

neqn file ... | nroff | 450

CAUTION. Make sure 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 LPI, unless dynamically changed to 8 LPI 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 **LINEFEED** 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.

The *neqn* names of, and resulting output for, the Greek and special characters supported by *450* are shown in *greek*(5).

SEE ALSO

300(1), mesg(1), stty(1), tabs(1).

BUGS

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

If your output contains Greek and/or reverse **LINEFEEDS**, 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 **LINEFEEDS**.

NAME

acctcom – search and print process accounting file(s)

SYNOPSIS

acctcom [[*options*][*file*]] . . .

DESCRIPTION

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

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

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

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

–a
Show some average statistics about the processes selected. The statistics will be printed after the output records.

–b
Read backwards, showing latest commands first. This *option* has no effect when the standard input is read.

–f
Print the *fork/exec* flag and system exit status columns in the output. The numeric output for this option will be in octal.

–h
Instead of mean memory size, show the fraction of total available CPU time consumed by the process during its execution. This "hog factor" is computed as:

(total CPU time)/(elapsed time).

–i
Print columns containing the I/O counts in the output.

–k
Instead of memory size, show total kcore-minutes. As with the **–m** option, REAL/IX does not reduce the reported memory usage for the cases where memory is swapped to disk or is shared with other processes.

–m
Show mean core size (the default). This is the average value for the total size of the process, including both main memory usage and any part of the process image that has been swapped to disk. No attempt is made to reduce the reported memory usage for the cases where some part of the process memory is shared with other processes. This style differs from traditional implementations, which do attempt to account for swapping and sharing. The *proc* command of *crash*(1M) can be used if more details of process memory usage are required.

–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 a ? (which designates only those processes associated with unknown user IDs). Remember that the # or the ? character is enclosed within apostrophes or quotation marks or preceded by a backslash.

–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 ending at or after *time*, given in the format *hr*[:*min*[:*sec*]].

–e *time*
Select processes starting 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*; suppress 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

ps(1), su(1), acct(1M), acctcms(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), crash(1M), fwtmp(1M), runacct(1M), 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, *time* is interpreted as occurring on the previous day.

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 file names 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 following describes the keyletter arguments. 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 to create a new SCCS file.

-i*[name]*
The *name* of a file from which the text for a new SCCS file is to be taken. The text constitutes the first delta of the file (see **-r** keyletter for delta numbering scheme). If the **i** keyletter is used but the file name is omitted, the text is obtained by reading the standard input until an EOF 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 also used), the descriptive text file name must also be supplied. In the case of existing SCCS files: (1) a **-t** keyletter without a file name causes removal of descriptive text (if any) currently in the SCCS file, and (2) a **-t** keyletter with a file name causes text (if any) in the named file to replace the descriptive text (if any) currently in the SCCS file.

-f*flag*
This keyletter specifies a *flag*, and, possibly, a value for the *flag*, to be placed in the SCCS file. Several **f** keyletters may be supplied on a single *admin* command line. The allowable *flags* and their values are:

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

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 (SIDs+1) 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 exactly match the given string, however the string must contain a keyword, and no embedded newlines.

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

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> ::= *RELEASE NUMBER* | **a**

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

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

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 *logins*, or numerical group IDs, as desired may be on the list simultaneously. If the list of users is empty, then anyone may add deltas. If *login* or group ID is preceded by a **!**, they are to be denied permission to make deltas.

–e*login*

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

–m*[mrlist]*

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

–y*[comment]*

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

date and time created *YY/MM/DD HH:MM:SS* by *login*

The –**y** keyletter is valid only if the –**i** and/or –**n** keyletters are specified (i.e., a new SCCS file is being created).

–**h**

Causes *admin* to check the structure of the SCCS file (see *sccsfile*(5)), and to compare a newly computed checksum (the sum of all the 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. 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 file names must be of the form **s.***file-name*. New SCCS files are given mode 444 (see *chmod*(1)). Write permission in the pertinent directory is, of course, required to create a file. All writing done by *admin* is to a temporary x-file, called **x.***file-name*, (see *get*(1)), created with mode 444 if the *admin* command is creating a new SCCS file, or with the same mode as the SCCS file if it exists. After successful execution of *admin*, the SCCS file is removed (if it exists), and the x-file is renamed with the name of the SCCS file. This ensures that changes are made to the SCCS file only if no errors occurred.

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

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

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

FILES

g-file	Existed before the execution of *delta*; removed after completion of *delta*.
p-file	Existed before the execution of *delta*; may exist after completion of *delta*.
q-file	Created during the execution of *delta*; removed after completion of *delta*.
x-file	Created during the execution of *delta*; renamed to SCCS file after completion of *delta*.
z-file	Created during the execution of *delta*; removed during the execution of *delta*.
d-file	Created during the execution of *delta*; removed after completion of *delta*.
/usr/bin/bdiff	Program to compute differences between the "gotten" file and the *g-file*.

SEE ALSO

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

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

ar – archive and library maintainer for portable archives

SYNOPSIS

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

DESCRIPTION

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

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

Unlike command options, the command key is a required part of *ar*'s command line. The key (which may begin with a –) is formed with one of the following letters: **drqtpmx**. Arguments to the *key*, alternatively, are made with one of more of the following set: **vuaibcls**. *posname* is an archive member name used as a reference point in positioning other files in the archive. *afile* is the archive file. The *names* are constituent files in the archive file. The meanings of the *key* characters are:

d

Delete the named files from the archive file.

r

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

q

Quickly append the named files to the end of the archive file. Optional positioning characters are invalid. The command does not check if the added members are already in the archive. This option is useful to avoid quadratic behavior when creating a large archive piece-by-piece. Unchecked, the file may grow exponentially up to the second degree.

t

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

p

Print the named files in the archive.

m

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

x

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

The meanings of the key arguments are:

v

Give a verbose file-by-file description of the making of a new archive file from the old archive and the constituent files. When used with **t**, give a long listing of all information about the files. When used with **x**, precede each file with a name.

c

Suppress the message that is produced by default when *afile* is created.

l

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

s

Force the regeneration of the archive symbol table even if *ar*(1) is not invoked with a command which will modify the archive contents. This command is useful to restore the archive symbol table after the *strip*(1) command has been used on the archive.

FILES

$TMPDIR/* temporary files

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

SEE ALSO

ld(1), lorder(1), strip(1), tmpnam(3S), a.out(4), ar(4).

NOTES

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

NAME

/bin/as – assembler driver script

SYNOPSIS

/bin/as [*options*] *filename*

DESCRIPTION

The */bin/as* command both silicon filters and assembles the named file. This driver script allows the silicon filtering to be performed in a transparent fashion which avoids having to make special changes to makefiles or scripts to accommodate the silicon filter pass. The following flags may be specified in any order:

–o *objfile*
Put the output of the assembly in *objfile*. By default, the output file name is formed by removing the **.s** suffix, if there is one, from the input file name and appending a **.o** suffix.

–F*options*
Pass these options on to the silicon-filter. This option is used to specify the *sifilter* command options that will be used by the silicon filter when it is automatically run by */bin/as*. See *sifilter*(1) for a complete list of filter options.

/bin/as is implemented as a script that calls both the silicon filter and the assembler in two passes.

FILES

/usr/tmp/as[A-Z]XXXXX	temporary files
/lib/as	assembler
/bin/sifilter	silicon filter

SEE ALSO

sifilter(1).

NAME

as – assembler

SYNOPSIS

/lib/as [*options*] *filename*

DESCRIPTION

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

–o *objfile*
Put the output of the assembly in *objfile*. By default, the output file name is formed by removing the **.s** suffix, if there is one, from the input file name and appending a **.o** suffix.

–m
Run the **m4** macro processor on the input to the assembler.

–R
Remove (unlink) the input file after assembly is completed.

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

–Y *[md],dir*
Find the **m4** preprocessor (**m**) and/or the file of predefined macros (**d**) in directory *dir* instead of in the customary place.

FILES

/usr/tmp/as[A-Z]XXXXX temporary files

NAME

asa – interpret ASA carriage control characters

SYNOPSIS

asa [*file*...]

DESCRIPTION

asa interprets the output of FORTRAN programs that use 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; their meanings are:

(blank)	single newline before printing
0	double newline 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 a space. 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.

EXAMPLE

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

a.out | asa | lp

The output, properly formatted and paginated, would be directed to the line printer. FORTRAN output sent to a file could be viewed by:

asa file

FILES

/bin/asa

NAME

assist – assistance using REAL/IX commands

SYNOPSIS

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

DESCRIPTION

The *assist* command invokes the ASSIST menu interface software. The ASSIST menus categorize user commands according to function in a hierarchy. 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 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:

name
invoke an ASSIST-supported 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 and check or modify your terminal variable; or 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 **^A (CTRL–a)** or **f8 (function-key** 8) when you are in *assist*. When you do this, a list of the commands is printed on the terminal screen.

EXAMPLE

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* (e.g., about the terminal you are using)
/usr/lib/assist	default directory containing assist command forms, walkthrus, and executable programs *assist* and *astgen*

NOTES

The first time you invoke *assist* it ignores any options you give and asks for information about the terminal you are using. Once it has saved this information in a file named *.assistrc* in *assist*, it shows you a list of basic *assist* commands and offers you an introduction to ASSIST.

SEE ALSO

astgen(1).

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*** automatically will 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*]

batch

DESCRIPTION

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

–r
Removes jobs previously scheduled with *at*.

–l
Reports all jobs scheduled for the invoking user.

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

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

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

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

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

Thus legitimate commands include:

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

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

batch submits a batch job. It is almost equivalent to "at ˜now", but not quite. For one, it goes into a different queue. For another, "at ˜now" will respond with the error message **too ˜late**.

at **-r** removes jobs previously scheduled by *at* or *batch*. The job number is the number given to you previously by the *at* or *batch* command. You can also get job numbers by typing *at* **-l**. You can only remove your own jobs unless you are the superuser.

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')
```

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

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

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

```
echo "sh shellfile" | at 1900 thursday next week
```

FILES

/usr/lib/cron	main cron directory
/usr/lib/cron/at.allow	list of allowed users
/usr/lib/cron/at.deny	list of denied users
/usr/lib/cron/queue	scheduling information
/usr/spool/cron/atjobs	spool area

SEE ALSO

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 [**–F** *re*] [*parameter...*] ['*prog*'] [**–f** *progfile*] [*file...*]

DESCRIPTION

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

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

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

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

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

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

A pattern-action statement has the form:

pattern { action }

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

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

expression relop expression
expression matchop regular expression

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

var in array,

or a Boolean combination of these.

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

Regular expressions are as in *egrep* (see *grep*(1)). In patterns, they must be surrounded by slashes. Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions. A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines between an occurrence of the first pattern and the next occurrence of the second pattern.

A regular expression may be used to separate fields by using the **–F** *re* option or by assigning the expression to the built-in variable FS. The default is to ignore leading blanks and to separate fields by blanks and/or tab characters. However, if FS is assigned a value, leading blanks are no longer ignored.

Other built-in variables include:

ARGC	command line argument count
ARGV	command line argument array
FILENAME	name of the current input file
FNR	ordinal number of the current record in the current file
FS	input field separator regular expression (default blank)
NF	number of fields in the current record
NR	ordinal number of the current record
OFMT	output format for numbers (default %.6g)
OFS	output field separator (default blank)
ORS	output record separator (default newline)
RS	input record separator (default newline)

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

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

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

The **print** statement prints its arguments on the standard output, or on a file if *>expression* is present, or on a pipe if | *cmd* is present. The arguments are separated by the current output field separator and terminated by the output record separator. The **printf** statement formats its expression list according to the format (see *printf*(3S)).

awk has a variety of built-in functions: arithmetic, string, input/output, and general.

The arithmetic functions are: *atan2*, *cos*, *exp*, *int*, *log*, *rand*, *sin*, *sqrt*, and *srand*. *int* truncates its argument to an integer. *rand* returns a random number between 0 and 1. *srand* (*expr*) sets the seed value for *rand* to *expr* or uses the time of day if *expr* is omitted.

The string functions are:

gsub(for, repl, in)
behaves like *sub* (see below), except that it replaces successive occurrences of the regular expression (like the *ed* global substitute command).

index(s, t)
returns the position in string *s* where string *t* first occurs, or 0 if it does not occur at all.

length(s)
returns the length of its argument taken as a string, or of the whole line if there is no argument.

match(s, re)
returns the position in string *s* where the regular expression *re* occurs, or 0 if it does not occur at all. RSTART is set to the starting position (which is the same as the returned value), and RLENGTH is set to the length of the matched string.

split (s,a,s)
splits the string *s* into array elements *a [1], a [2], a [n],* and returns *n*. The separation is done with the regular expression *fs* or with the field separator FS if *fs* is not given.

sprintf(fmt,expr,expr,...)
formats the expressions according to the *printf*(3S) format given by *fmt* and returns the resulting string.

sub(for,repl,in)
substitutes the string *repl* in place of the first instance of the regular expression *for* in string *in* and returns the number of substitutions. If *in* is omitted, *awk* substitutes in the current record ($0).

substr(s, m, n)
returns the *n -character* substring of *s* that begins at position *m* .

The input/output and general functions are:

close(filename)
closes the file or pipe named *filename*.

cmd |getline
pipes the output of *cmd* into *getline*; each successive call to *getline* returns the next line of output from *cmd*.

getline
sets $0 to the next input record from the current input file.

getline <file
sets $0 to the next record from *file*.

getline var
sets variable *var* instead.

getline var <file
sets *var* from the next record of *file*.

system(cmd)
executes *cmd* and returns its exit status.

All forms of *getline* return 1 for successful input, 0 for end of file, and −1 for an error.

awk also provides user-defined functions. Such functions may be defined (in the pattern position of a pattern-action statement) as:

function name(args,...) { stmts }
func name(args,...) { stmts }

Function arguments are passed by value if scalar and by reference if array name. Argument names are local to the function; all other variable names are global. Function calls may be nested and functions may be recursive. The **return** statement may be used to return a value.

EXAMPLES

Print lines longer than 72 characters:

```
length > 72
```

Print first two fields in opposite order:

```
{ print $2, $1 }
```

Same, with input fields separated by comma and/or blanks and tabs:

```
BEGIN { FS = ",[ \t]*|[ \t]+" }
      { print $2, $1 }
```

Add up first column, print sum and average:

```
      { s += $1 }
END   { print "sum is", s, " average is", s/NR }
```

Print fields in reverse order:

```
{ for (i = NF; i > 0; —i) print $i }
```

Print all lines between start/stop pairs:

```
/start/, /stop/
```

Print all lines whose first field is different from previous one:

```
$1 != prev { print; prev = $1 }
```

Simulate *echo*(1):

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

Print file, filling in page numbers starting at 5:

```
/Page/ { $2 = n++; }
       { print }
```

command line: **awk –f program n=5 input**

SEE ALSO

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

BUGS

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

NAME

banner – make posters

SYNOPSIS

banner strings

DESCRIPTION

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

SEE ALSO

echo(1).

NAME

basename, dirname – deliver portions of 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 following example will set the shell variable **NAME** to **/usr/src/cmd**:

```
NAME=`dirname /usr/src/cmd/cat.c`
```

SEE ALSO

sh(1).

NAME

bc – arbitrary-precision arithmetic language

SYNOPSIS

bc [–c] [–l] [*file* ...]

DESCRIPTION

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

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

The same letter may be used as an array, a function, and a simple variable simultaneously. All variables are global to the program. "Auto" variables are pushed down during function calls. When using arrays as function arguments or defining them as automatic variables, empty square brackets must follow the array name.

EXAMPLE

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

defines a function to compute an approximate value of the exponential function and

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

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

FILES

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

SEE ALSO

dc(1).

BUGS

The *bc* command does not yet recognize the logical operators, && and | |.
for statement must have all three expressions (E's).
quit is interpreted when read, not when executed.

NAME

bdiff – big diff

SYNOPSIS

bdiff *file1 file2* [n] [–s]

DESCRIPTION

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

The parameters to *bdiff* are:

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

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

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

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

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

FILES

/tmp/bd?????

SEE ALSO

diff(1), help(1).

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

bfs – big file scanner

SYNOPSIS

bfs [–] *name*

DESCRIPTION

The *bfs* command is (almost) like *ed*(1) except that it is read-only and processes much larger files. Files can be up to 1024Kb and 32K lines, with up to 4096 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 more manageable pieces for editing.

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

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

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

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

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

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

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

(. , .)**xb**/*regular expression*/*label*
A jump (either upward or downward) is made to *label* if the command succeeds. It fails under any of the following conditions:

1. Either address is not between **1** and **$**.
2. The second address is less than the first.
3. The regular expression does not match at least one line in the specified range, including the first and last lines.

On success, **.** is set to the line matched and a jump is made to *label*. This command is the only one that does not issue an error message on bad addresses, so it may be used to test whether addresses are bad before other commands are executed. 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**:

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

will all print the first 100 lines.

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

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

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

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

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

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

would put the current line into variable **5**, print it, and increment the variable **6** by one. To escape the special meaning of **!** as the first character of *value*, precede it with a \.

```
xv7\!date
```

stores the value **!date** into variable **7**.

xbz *label*

xbn *label*

These two commands will test the last saved *return code* from the execution of a command (!*command*) or nonzero value, respectively, to the specified label. The following two examples 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** it is not. Without an argument, **xc** reverses *switch*. Initially, *switch* is set for no crunching. Crunched output has strings of tabs and blanks reduced to one blank and blank lines suppressed.

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

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

SYNOPSIS

biff [yn]

DESCRIPTION

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

biff y

enables notification; the command

biff n

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

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

SEE ALSO

sh(1), mail(1), comsat(1M).

NAME

bison – GNU Project parser generator (yacc replacement)

SYNOPSIS

bison [**-dlvty**] file

DESCRIPTION

Bison is a parser generator in the style of *yacc*(1). It should be upwardly compatible with input files designed for *yacc*.

Input files should follow the *yacc* convention of ending in ".y". Unlike *yacc*, the generated files do not have fixed names, but instead use the prefix of the input file. For instance, a grammar description file named **parse.y** would produce the generated parser in a file named **parse.tab.c**, instead of *yacc*'s **y.tab.c**.

Bison takes five optional flags.

-d Produce a **.tab.h** file, similar to *yacc*'s **y.tab.h** file.

-l Omit **#line** lines in the parser output file. Error messages from the C compiler will then be associated with lines in the parser output file, instead of lines in the original grammar file.

-t Turn on debugging. This option causes the *bison* output to have debugging code made available via the C pre-processor. The external variable **yydebug** should be made non-zero to have the debugging code actually produce output.

-v Be verbose. Analogous to the same flag for *yacc*.

-y Use fixed output file names. I.e., force the output to be in files **y.tab.c**, **y.tab.h**, and so on. This is for full *yacc* compatibility.

FILES

/usr/local/lib/bison.simple	simple parser
/usr/local/lib/bison.hairy	complicated parser

SEE ALSO

yacc(1)

The *Bison Reference Manual*, included as the file **bison.texinfo** in the *bison* source distribution.

DIAGNOSTICS

"Self explanatory."

NAME

bpatch – displays or alters byte content of files

SYNOPSIS

bpatch [<*filename*>]

DESCRIPTION

bpatch allows a user to look at specific areas of a file and to change selected bytes. *bpatch* takes no information from the command line except for the optional name of a file to open initially. *bpatch* displays a prompt (>) and waits for user input. The following commands are recognized:

o *filename*
Open the specified file. If another file is already open, close it.

s *xxxx yyyy*
Display the contents of the file from *xxxx* to *yyyy*, where *xxxx* and *yyyy* are hex values. The starting address is rounded down modulo hex 10, and the ending address is rounded up modulo hex 10. The contents are shown both as hex and printed ASCII. Nonprinting characters are shown as periods in the ASCII display.

c *xxxx zz ...*
Change contents of location *xxxx* to *zz*, where *xxxx* and *zz* are hex numbers. Several values can be changed by entering additional values. Each additional value changes the next sequential byte.

d
Print information about the file, e.g., its type, permissions, size.

q
Exit.

The hex numbers *xxxx* and *yyyy* can represent a full 32-bit address (i.e., eight hex characters). The hex number *zz* represents a byte and should be only two hex characters.

bpatch is insensitive to case of alphabetic characters. The above commands, as well as the characters **A-F** within hex numbers, may be uppercase or lowercase.

NOTE

The prompt has a trailing **NULL** character (> NUL) not visible on a normal terminal.

EXAMPLE

s 0 ff	display bytes 0 thru ff
S 10000 100FF	display bytes 10000 thru 100ff
c 400 7F 31 21	change contents of locations 400-402 to 7F 31 21
o newfile	close the present file and open "newfile"
d	display file information
q	exit the *bpatch* program

DIAGNOSTICS

bpatch gives error messages if it is unable to open a file, or if any commands are entered that would read or write when no file is open.

If an illegal command or no input is entered, *bpatch* displays a brief help message.

If the user has read permission but not write permission on a file, *bpatch* will display a message to this effect at open time. The user will be unable to use the c command.

NAME

bru – backup and restore utility

SYNOPSIS

bru *modes* [*control options*] [*selection options*] [*files*]

DESCRIPTION

bru is a UNIX file system backup utility with significant enhancements over other more common utilities, e.g., *tar*, *cpio*, *volcopy*, and *dd*. Some of capabilities of *bru* include:

- Full or incremental backup with quick and easy restoration of files.
- Multiple physical volumes per archive.
- Data integrity assurance via checksum computation on every archive block.
- Ability to properly save and restore directories, symbolic links, block special files, and character special files.
- Comparison of archives with current directory hierarchy.
- Ability to recover files from corrupted archives or damaged media with minimal data loss.
- No inherent maximum archive buffer size.
- Improved performance through random access archive I/O when available.
- Automatic byte or half word swapping as necessary when reading archives produced on other machines.
- Recognition of file name generation patterns in the same form as the shell for files read from an archive.
- Intelligent handling of large, sparse files.

When *files* are specified on the command line, the actions to be performed are limited to those *files*. If a named file is a directory, it and all its descendants are used. If no *files* are specified, the default for writing archives is all files in and below the current directory. The default for reading archives is selection of all files in the archive.

If – is given instead of *files*, then the standard input is read to obtain the file list. This is useful with the *find* command to provide finer control over files selected for backup. Obviously, this mode is only valid when *bru* is not also reading its archive from the standard input.

DEFAULTS

Various default parameters (e.g., archive device name and size, archive buffer size, and controlling terminal name) are system dependent. These defaults, along with version, variant, and other miscellaneous internal information may be discovered via the **–h** mode.

MODES

One or more of the following modes must be specified. The order of execution, from highest priority to lowest, is **ecitxdgh**.

–c

create a new archive. Forces a new archive to be created regardless of whether one currently exists. Writing starts at the first block.

–d

differences between archived *files* and current *files* are detected and reported. It may be specified more than once, as **–dd** **–ddd** or **–dddd** to control the level of difference checking.

When specified as **–d**, *bru* reports when it discovers that a regular file's size (st_size) or contents (when compared as byte streams) has changed since the archive was made.

When specified as **-dd**, *bru* reports additional differences in modification date (st_mtime), access mode (st_mode), number of links (st_nlink) for nondirectory files, differences in the contents of symbolic links, owner id (st_uid), and group id (st_gid).

When specified as **-ddd**, *bru* reports additional differences in host device (st_dev), major/minor device (st_rdev) for special files, and time of last access (st_atime) for regular files.

When specified as **-dddd**, *bru* reports all differences except time of last status change (st_ctime is not resettable), major/minor device numbers for nonspecial files (meaningless), and size differences for directory files (may have empty entries). The **-dddd** mode is generally only meaningful during a verification pass with full backups of quiescent file systems.

-e

estimate media requirements for archive creation with same arguments. Prints estimated number of volumes, number of files to be archived, total number of archive blocks, and total size of archive in kilobytes. If the media size is unknown or unspecified, it is assumed to be infinite.

-g

dump archive info block in a form more easily parsed by programs implementing a complete file system management package. Performs no other archive actions.

-h

print help summary of options. Also prints some internal information, e.g., version number and default values for archive pathname, media size, archive buffer size.

-i

inspect archive for internal consistency and data integrity. When **-vv** option is also given, prints information from archive header block.

-t

list table of contents of archive. When used with the **-v** option, gives a verbose table of contents in the same format as the *ls -l* command. When used with the **-vv** option, it also indicates what files are linked to other files and where symbolic links point to.

-x

extract named *files* from archive. If an archived file is extracted (see **-u** option), then the access mode, device id (special files only), owner uid, group uid, access time, and modification time are also restored. If the **-C** flag is given, the owner uid and group uid are changed to that of the current user.

Nonexistent directories are recreated from archived directories if possible, otherwise, they are created with appropriate defaults for the current user. Extracted or created directories are initially empty.

CONTROL OPTIONS

Many of the control options are similar in function to their *tar* or *cpio* equivalents.

Sizes are specified in bytes. The scale factors **M**, **k**, or **b** can be used to indicate multiplication by 2**20 (1Mb), 1024, or 512 respectively. Thus, "10k", "20b", and "10240" all specify the same number of bytes.

-# *str*

use string *str* as a control string for the built-in debugging system. This option provides information about the internal workings of *bru* for the software maintainer or the merely curious. Some examples are given later.

–a

do not reset the access times of disk files that have been read while performing other actions. *bru* previously restored the access and modification times of disk files after they had been read. Resetting the times prevents defeat of the mechanism used to track down and remove *dead* files that have not been accessed in any meaningful way recently. For various reasons, this flag is now permanently enabled, thus defeating the preservation of the access time field in the stat structure.

–A *flags*

special machine dependent options for the Commodore Amiga. Current *flags* are:

c

clear file archived bit after processing

i

ignore file archived bit for selecting files

r

reject files that have archived bit set

s

set file archived bit after processing

–b *bsize*

use *bsize* as the archive input/output buffer size. The minimum is the size of an archive block (2K or 2048 bytes) and the maximum is determined by available memory and I/O device limitations. If *bsize* is not an even multiple of 2048 bytes, it is rounded up. Normally, this option is only required with the –c mode because *bru* writes this information in the archive header block. If specified, *bsize* overrides any existing default value (generally 20K), whether built in or read from the archive header.

–B

useful in shell scripts where *bru* is run in the background with no operator present. Under these conditions, *bru* terminates with appropriate error messages and status, rather than attempting interaction with the terminal.

–C

change the owner (*chown*) and group of each extracted file to the owner uid and group gid of the current user. Normally, *bru* restores the owner and group to those recorded in the archive. This flag causes *bru* to follow the system default, with extracted files having the same owner and group as the user running *bru*, including root. On systems that support a user being in multiple groups, the default group is that of the directory in which the file is created.

The –C option is useful with archives imported from other systems. In general, it should not be used by the operator or System Administrator when restoring saved files. Use the **–tv** option to see the owner and group of files stored in the archive.

–D

causes *bru* to use double buffering to the archive device on systems that have REAL/IX style shared memory. Depending upon hardware constraints, double buffering may dramatically increase the archive device I/O rate, but may adversely affect the error recovery algorithms.

–f *path*

use *path* as the archive file instead of the default. If the *path* is –, *bru* uses the standard input for archive reading or standard output for archive writing, as appropriate.

If multiple **–f** options are given, each *path* is added to a list of files to cycle through each time a volume change is required. When the end of the list is reached, *bru* automatically cycles back to the first *path* and waits for confirmation to continue the cycle again. Any input other

than a carriage return causes *bru* to use the newly entered *path* and to abort the cycling for the remainder of the current run.

–F

fast mode. In fast mode, checksum computations and comparisons are disabled. This mode is useful when the output of one *bru* is piped to the input of another *bru*, or when the data integrity of the archive transmission medium is essentially perfect. Archives recorded with fast mode enabled must also be read with fast mode. Also, note that some of the automatic features of *bru* (e.g., automatic byte swapping) are not functional in fast mode.

–I *iopt*

set interaction option from *iopt*. Currently recognized values for *iopt* are:

q,*fifo*
write queries to *fifo*

r,*fifo*
read responses from *fifo*

l,*file*
write log info to *file*

The **q** and **r** options are useful for interacting with *bru* when it has been run without a controlling terminal. See the discussion under *Running From Cron*.

–L *str*

label the archive with the specified string *str*. *str* is limited to 63 characters and is usually some meaningful reminder pertaining to the archive contents.

–l

ignore unresolved links. Normally, *bru* reports problems with unresolved links (both regular and symbolic links). This option suppresses all such complaints.

–m

Do not cross mounted file system boundaries during expansion of explicitly named directories. This option applies only to directories named in *files*. It limits selection of directory descendants to those located on the same file system as the explicitly named directory. This option currently applies only to the **–c** and **–e** modes.

–N *nbits*

use *nbits* as the number of bits for LZW compression if **–Z** is also specified. The default is to use 12 bits, which is also the minimum allowable. The maximum allowable value is 16 bits. Archives created with more than 12 bits of compression may be unreadable on smaller systems due to memory or processor word length constraints.

–p

pass over files in archive by reading instead of seeking. Normally, *bru* uses random access capabilities if available. This option forces reading instead of seeks.

–P *flags*

pathname options that provide explicit control of expansion of directories or automatic archiving of parent directories. Possible characters for *flags* are:

e
disable automatic expansion of explicitly named directories.

E
enable automatic expansion of explicitly named directories.

f

disable filter mode. Builds internal tree of pathnames before doing anything with the pathnames from the input list.

F

enable filter mode. Each pathname from the input list is treated individually, and processing is done with that pathname before the next pathname is examined.

p

disable automatic archiving of parent directories of explicitly named files or directories.

P

enable automatic archiving of parent directories of explicitly named files or directories.

See the discussion under *Directories*.

–R

remote files are to be *excluded* from the archive. If the system does not support remote file systems, this option is ignored.

–s *msize*

use *msize* as the media size. The effective media size is computed from *msize* because it must be an integral multiple of the input/output buffer size (see the **–b** option). Normally, this option is only required with the **–c** mode since *bru* writes this information in the archive header block. If specified, *msize* overrides any existing default value, whether built in or read from the archive header.

–S *size*

enable options to deal more intelligently with *sparse* files (files with many NULL bytes). When used in conjunction with the **–c** mode, it turns on automatic file compression for files that are larger than the specified *size*. When used in conjunction with the **–x** mode, seeks are used to create blocks of NULL bytes in the output file, instead of actually writing NULL bytes. See the discussion under *Sparse Files*.

–v

enable verbose mode. May be specified more than once, as **–vv**, **–vvv**, or **–vvvv** to get even more verbosity.

–w

wait for confirmation. *bru* prints the filename, the action to be taken, and waits for confirmation. Any response beginning with **y** or **Y** causes the action to finish. Any response beginning with **g** or **G** causes the action to finish and resets the **–w** option so that no further confirmations are requested. Any other response aborts the action.

–Z

use LZW file compression. This is not the default because not all versions of *bru* can handle compressed files. When the **–v** option is also selected, the compression ratio for each file is printed as a percentage. When this flag is used in conjunction with the **–t** option on an archive that contains a compressed file, the actual archive file sizes and names are printed, instead of the original values before archiving.

A limited amount of backwards compatibility to noncompression versions of *bru* is provided. Archives read by older versions appear to contain files that were precompressed prior to archiving. The public domain *compress* utility can be used to decompress such files after dearchiving.

Also see the **–N** option.

FILE SELECTION OPTIONS

The file selection options control which files are selected for processing. Note that some options are only valid with specific modes.

−n *date*

select only files *newer* than *date*. The *date* is given in one of the forms:

DD−MMM−YY[,HH:MM:SS]	12−Mar−90,12:45:00
MM/DD/YY[,HH:MM:SS]	3/12/90
MMDDHHMM[YY]	0312124590
pathname	**/etc/lastfullbackup**

The time of day is optional in the first two forms. If present, it is separated from the date with a comma.

If *date* is really the pathname of a file, then the modification date of that file is used instead. This is useful in automated backups when a dummy file is *touched* to save the date of last backup.

−o *user*

select only files owned by *user*. *user* may be specified in one of three ways:

- As an ASCII string corresponding to a user name in the password file.
- As the pathname of a file in which case the owner of that file is used.
- As a numeric value (decimal).

−u *flags*

when used with **−x** mode, causes files of type specified by *flags* to be unconditionally selected regardless of modification times. Normally, *bru* does not overwrite (supersede) an existing file with an older archive file of the same name. Files that are not superseded will give warnings if **verbose** mode level 2 (**−vv**) or higher is enabled. Possible characters for *flags* are:

a
select all files (same as specifying all flags)

b
select block special files

c
select character special files

d
select directories

l
select symbolic links

p
select fifos (named pipes)

f
select regular files (same as **r**)

r
select regular files (same as **f**)

Selection of directories only implies that their attributes may be modified. Existing directories are never overwritten, this option merely allows their attributes to be set back to some previously existing state. Selection of symbolic links only implies that the contents of the link are modified. It is currently impossible to change access time, modification time, or the file mode of a symbolic link.

EXAMPLES

Create (**-c**) a new archive of all files under **/usr/src**, writing archive to file (**-f**) **/dev/rst0** using multiple tapes with a maximum size (**-s**) of 30Mb per tape:

```
bru -c -f /dev/rst0 -s 30M /usr/src
```

Create (**-c**) a new archive on the default device in the first pass, archiving all files in and below the current directory that have been created or modified (**-n**) since 3 P.M. on 14-Jan-90. Then, do a second pass to verify that there are no differences (**-d**) between the archive and current files. Each file is listed (**-v**) as it is processed:

```
bru -cvd -n 14-Jan-90,15:00:00
```

Archive all files owned (-o) by user **user1** using the default archive device:

```
find / -user user1 -print | bru -c -
bru -c -o user1 /
```

Copy a directory hierarchy from **/usr/u1** to **/usr/u2**:

```
(cd /usr/u1; bru -cf - ) | (cd /usr/u2; bru -xf -)
```

Extract (**-x**) the regular file **/usr/guest/myfile** unconditionally (**-ur**) from an archive on file (**-f**) **/dev/rf0**. Since the device size was recorded in the header block, it need not be specified. Note that option arguments do not need to be separated from their corresponding option flag by white space:

```
bru -x -ur -f/dev/rf0 ./usr/guest/myfile
```

Extract (**-x**) all C source files in **/usr/src/cmd** that have names beginning with characters **a** through **m**. Wait (**-w**) for confirmation before extracting each file:

```
bru -xw '/usr/src/cmd/[a-m]*.c'
```

Inspect (**-i**) a previously created archive on the default device, dumping the contents of the header block for inspection (**-vvv**) and verifying internal consistency and data integrity of the archive:

```
bru -ivvv
```

Perform the same function as the last example except enable various features of the built-in debugger (when linked in). The debug control string is a string of the form **-#<opt1>:<opt2>:...**, where each option is either a single flag character or a flag character followed by a comma separated list. Available flag characters are: **d** enable debugging for list of keywords, **f** limit debugging to list of function names, **F** print source file name, **L** print source file line numbers, **n** print nesting depth, **o** redirect output to listed file, **p** print process name, **t** enable tracing.

```
bru -ivvv -#t
bru -ivvv -#d:t
bru -ivvv -#d,ar_io,verify:F:L
bru -ivvv -#d:f,ar_seek
bru -ivvv -#d:o,trace.out:t:p
```

Back up the entire **root** file system without crossing mounted (**-m**) file system boundaries. The archive is written to file (**-f**) **/dev/rst0** using an I/O buffer size (**-b**) of 10Kb. A record of all files processed will be written to file **brulogfile** for future reference.

```
cd /
bru -cvm -f /dev/rst0 -b 10k > brulogfile
```

DIAGNOSTICS

Most diagnostics are reasonably informative. The most common have to do with meaningless combinations of options, incompatible options, hitting memory or device limits, unresolved file links, trying to archive or restore something to which access is normally denied, or problems with media errors and/or archive corruption.

DEVICE TABLE

bru contains an internal table of known devices and their characteristics. *bru* first looks for an environment variable BRUTAB, that contains the name of a dynamically loaded file if it begins with a / character, or contains device descriptions if the first character is not /. If there is no BRUTAB environment variable, the file **/etc/brutab** is loaded. If neither of the preceding is found, an internal default description is loaded.

SIGNAL HANDLING

bru normally catches both interrupt (SIGINT) and quit (SIGQUIT). When interrupt is caught during archive creation or extraction, *bru* completes its work on the current file before cleaning up and exiting. This is the normal way of aborting *bru*. When a quit signal is caught, an immediate exit is taken.

Note that during file extraction, a quit signal may leave the last file only partially extracted. Similarly, a quit signal during archive writing may leave the archive truncated. When either interrupt or quit is caught at any other time, an immediate exit is taken.

ERROR RECOVERY

When properly configured for a given software/hardware environment, *bru* can recover from most common errors. For example, attempts to use unformatted media are detected, allowing substitution of formatted media. Random blocks in an archive can be deliberately overwritten (corrupted) without affecting *bru*'s ability to recover data from the rest of the archive. When I/O errors are detected, retries are performed automatically. Out of order sequencing on multi-volume archive reads is detected, allowing replacement with the correct volume.

DIRECTORIES

There are two actions *bru* takes with respect to directories that make creation and extraction of entire hierarchies of files more convenient and complete: automatic archiving of parent directories and automatic expansion of explicitly named directories.

Automatic archiving of parent directories means that when *bru* is given the complete pathname of a file to archive, it attempts to automatically archive all parent directory nodes necessary to fully restore the specified file. During extraction, any required directories that do not already exist are restored from the archive if possible, otherwise, they are created with appropriate defaults for the current user. When *bru* reads its list of files from the standard input, or when the **-Pp** option is given, this automatic archiving of parent directory nodes is suppressed. Note also that when creating archives with additional constraints on the selected files (e.g., using the **-n** option), these parent directories may be excluded.

Automatic expansion of explicitly named directories means that when *bru* is given an explicit filename that names a directory node, not only is that directory node archived, but all files and subdirectories in that directory are archived, i.e., the entire file hierarchy rooted in the explicitly named directory is archived. When *bru* reads its list of files from the standard input, or when the **-Pe** option is given, this automatic expansion of directories is suppressed.

Note that incremental archives, archives created with the **-Pp** option, or archives created from a list of files supplied on the standard input stream, may not contain all the necessary parent directories to replicate the original hierarchy, thus, may result in creation of directories with the default attributes when files are extracted from the archive.

When *bru* reads the list of *files* from the standard input stream, the default values for the **-P** options are **-PeFp**, which turns off expansion of directories, turns on filter mode, and turns off automatic archiving of parent directories. This allows *bru* to be conveniently used to archive only file system nodes that are explicitly named on the input list.

When *files* are explicitly named on the command line (or default to .), the default values for the **-P** options are **-PEfP**, which turns on expansion of directories, turns off filter mode, and turns on automatic archiving of parent directories. This is typically the most convenient behavior for arguments given on the command line.

WILDCARDS

When **reading** archives, *bru* recognizes filename generation patterns in the same format as the shell. This allows greater flexibility in specifying files to be extracted, compared, or listed. As a special extension to shell type expansion, the sense of the match is reversed for patterns that begin with an exclamation (**!**).

Note that the patterns may have to be quoted to prevent expansion by the shell. Also note that patterns are processed independently, without regard for any other patterns that may or may not be present. In particular, **/bin/a* /bin/b*** is equivalent to **/bin/[ab]***, but **/bin/!a* /bin/!b*** is equivalent to **/bin/***, *not* **/bin/![ab]***.

BYTE/WORD SWAPPING

While reading archives produced on other machines, *bru* automatically attempts to perform byte and/or word swapping as necessary.

REMOTE TAPE DRIVES

Remote tape drives are not supported on the REAL/IX Operating System.

RUNNING FROM CRON

Sometimes it is convenient to run *bru* under conditions where there is no controlling terminal. This can be a problem if interaction is needed, e.g., when switching to a new volume. As an example, consider the case of running *bru* from *cron*, where the operator mounts a tape before leaving in the evening, and *bru* writes the first volume in the middle of the night. When returning in the morning, the operator wants to be able to mount a second tape if necessary, and instruct *bru* to continue.

If no interaction with the user is required, running from *cron* is no different than running directly from a terminal. However, when interaction is necessary, there are basically two options: terminate, or find some way to communicate with the operator (or another program masquerading as the operator). The **-B** option provides for simple termination. The **-I** option provides for communication with an operator.

On systems that support FIFOs, a pair of FIFOs are used to send requests and receive replies. Before running *bru*, create a pair of FIFOs with the commands:

```
mknod /dev/bru.q p
mknod /dev/bru.r p
```

Then, add the arguments **-Iq,/dev/bru.q -Ir,/dev/bru.r** to the desired *bru* command line which ultimately gets executed under *cron*. The first time *bru* needs to communicate with an operator, it opens the two FIFOs, writes a query to the **bru.q** FIFO, and waits for a response from the **bru.r** FIFO. A simple program provided with *bru*, called *brutalk*, can be used to read the query and send a reply:

brutalk < /dev/bru.q > /dev/bru.r

The *brutalk* program continues to read queries and sends replies until either *bru* exits, or a **< Ctrl> d** (EOF) is typed at the terminal.

EXIT CODES

bru always returns meaningful status:

0 Normal exit, no errors or warnings.
1 Warnings (or interrupted).
2 Errors (or quit signal).

SPARSE FILES

With the UNIX file system, it is possible to create files that take up much less space in the file system than their size would indicate. These files are generally known as *sparse* files, and are common in database or scientific applications.

For example, it is acceptable to create an empty file, seek to an offset of 1Mb from the start of the file, and write a single byte of data at that point, thus creating a file 1 byte more than 1Mb in length, with the first 1Mb being NULL bytes that are not actually stored in separately allocated blocks in the file system.

Without using the **–S** option, *bru* would archive this file using 1Mb of archive space, and on extraction, would create a file that actually takes up 1Mb of space in the file system.

With the **–S** option, *bru* automatically compresses this file down to about 4Kb in the archive, and on extraction, uses seeks to recreate the sparse file, thus saving about 1Mb of space in the file system.

The **–S** option is not the default because not all systems on which *bru* runs can support sparse files, and because the overhead of compressing all files larger than the autocompression threshold can be significant.

Note that this option is also useful even when sparse files are not expected, since it can be used during archive creation to selectively compress files larger than a specified size. This is generally faster than compressing every file, while still saving considerable space in the archive, because large files generally tend to compress better than small ones.

SEE ALSO

cpio(1), tar(1), ff(1M), filesave(1M), finc(1M), frec(1M), volcopy(1M).

CAVEATS

Pathnames are limited to 127 characters.
Implementation differences complicate the algorithms for automatic detection of end-of-file on devices. The algorithms can be fooled, hence, the **–s** option.

Special files moved to a machine other than their original host are generally useless and possibly even dangerous. This is also true for symbolic links.

When extracting files from archives, patterns used to match directories may result in some unnecessary directories being extracted. For example, if the pattern is **a/*/c**, and the directory **a/b** is encountered in the archive, the directory file **a/b** is extracted because it is needed when (and if) the file **a/b/c** is encountered. When in doubt, use the **–w** option.

In order to be able to efficiently archive needed directories, *bru* builds an image of the directory tree for *files* using dynamically allocated memory. Since there may be at most 5120 characters passed on the command line, it is very unlikely that *bru* runs out of memory while building the tree from command line arguments. This is not true of file lists read from the standard input, particularly on machines with limited address space.

Information about file linkages is also kept in memory. Some linkages may be lost if memory is exhausted.

Since *bru* is owned by **root** and runs with **set user id** to allow it to create directories and special files, it makes every attempt to prevent normal users from archiving or extracting files to which they would normally not have access. There may be loopholes. Also note that anyone with physical or electronic access to an archive, and knowledge of the archive structure, can recover any of its contents by writing their own file extraction program.

Directories that have file systems mounted on them are not properly archived until the file system is unmounted. This is not generally a problem.

Explicitly naming both a directory and one of its descendants causes the descendant to be archived twice, unless they are on separate file systems and the **-m** flag is used.

Explicitly naming a file more than once is ineffective.

When reading from the raw magnetic tape file (**rmtxxx**), *bru* automatically attempts to adjust the I/O buffer size to match that used to record the archive. Under certain circumstances, it may fail and require help via the **-b** option.

The remote magnetic tape protocol, *rmt*(8), allows writing and reading of no more than 10Kb, so **-b10k** must be specified when reading from or writing to a remote tape drive. Also, using remote tape drives can be slow.

NAME

bsh, rbsh – the bourne shell command programming language

NOTE

The default shell on the REAL/IX Operating System is the KornShell *sh*(1) **/bin/sh** = **/bin/ksh**. The Bourne shell *bsh*(1), is still available as **/bin/bsh**. Wherever you see *sh*(1) it means *ksh*(1).

SYNOPSIS

bsh [**–acefhiknrstuvx**] [args]
rbsh [**–acefhiknrstuvx**] [args]

DESCRIPTION

bsh is a command programming language that executes commands read from a terminal or a file. *rbsh* is a restricted version of the standard command interpreter *bsh*; 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.

This man page describes the Bourne shell. Other shells; such as the Korne shell *ksh*(1), exist, and many system administrators prefer to point the names *sh* and *rbsh* to one of these. Users of these systems can still gain access to the Bourne shell via the alternate names *bsh* and *rbsh*.

Definitions

A *blank* is a tab or a space. A *name* is a sequence of letters, digits, or underscores beginning with a letter or underscore. A *parameter* is a name, a digit, or any of the characters *, @, #, ?, –, $, and !.

Commands

A *simple-command* is a sequence of nonblank *words* separated by *blanks*. The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see *exec*(2)). The *value* of a *simple-command* is its exit status if it terminates normally, or (octal) 200+*status* if it terminates abnormally (see *signal*(2) for a list of status values).

A *pipeline* is a sequence of one or more *commands* separated by **|**. The standard output of each command but the last is connected by a *pipe*(2) to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate. The exit status of a pipeline is the exit status of the last command.

A *list* is a sequence of one or more pipelines separated by ;, &, &&, or **| |**, and optionally terminated by ; or &. Of these four symbols, ; and & have equal precedence, which is lower than that of && and **| |**. The symbols && and **| |** also have equal precedence. A semicolon (;) causes sequential execution of the preceding pipeline; an ampersand (&) causes asynchronous execution of the preceding pipeline (i.e., the shell does *not* wait for that pipeline to finish). The symbol && (**| |**) causes the *list* following it to be executed only if the preceding pipeline returns a zero (non-zero) exit status. An arbitrary number of newlines may appear in a *list*, instead of semicolons, to delimit commands.

A *command* is either a *simple-command* or one of the following. Unless otherwise stated, the value returned by a command is that of the last *simple-command* executed in the command.

for *name* [**in** *word* ...] **do** *list* **done**

Each time a **for** command is executed, *name* is set to the next *word* taken from the **in** *word* list. If **in** *word* ... is omitted, then the **for** command executes the **do** *list* once for each positional parameter that is set (see *Parameter Substitution* below). Execution ends when there are no more words in the list.

case *word* **in** [*pattern* [| *pattern*] ...) *list* ;;] ... **esac**

A **case** command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for file name generation (see "File Name Generation") except that a slash, a leading dot, or a dot immediately following a slash need not be matched explicitly.

if *list* **then** *list* [**elif** *list* **then** *list*] ... [**else** *list*] **fi**

The *list* following **if** is executed and, if it returns a zero exit status, the *list* following the first **then** is executed. Otherwise, the *list* following **elif** is executed and, if its value is zero, the *list* following the next **then** is executed. Failing that, the **else** *list* is executed. If no **else** *list* or **then** *list* is executed, then the **if** command returns a zero exit status.

while *list* **do** *list* **done**

A **while** command repeatedly executes the **while** *list* and, if the exit status of the last command in the list is zero, executes the **do** *list*; otherwise the loop terminates. If no commands in the **do** *list* are executed, then the **while** command returns a zero exit status; **until** may be used in place of **while** to negate the loop termination test.

(*list*)

Execute *list* in a subshell.

{*list*;}

list is executed in the current (that is, parent) shell.

name () {*list*;}

Define a function which is referenced by *name*. The body of the function is the *list* of commands between { and }. Execution of functions is described below (see *Execution*).

The following words are only recognized as the first word of a command and when not quoted:

if then else elif fi case esac for while until do done { }

Comments

A word beginning with # causes that word and all the following characters up to a newline to be ignored.

Command Substitution

The shell reads commands from the string between two grave accents (`` ` ` ``) and the standard output from these commands may be used as all or part of a word. Trailing newlines from the standard output are removed.

No interpretation is done on the string before the string is read, except to remove backslashes (\) used to escape other characters. Backslashes may be used to escape a grave accent (`` ` ``) or another backslash (\) and are removed before the command string is read. Escaping grave accents allows nested command substitution. If the command substitution lies within a pair of double quotes (" ...` ...` ... "), a backslash used to escape a double quote (\") will be removed; otherwise, it will be left intact.

If a backslash is used to escape a newline character (**newline**), both the backslash and the newline are removed (see the later section on "Quoting"). In addition, backslashes used to escape dollar signs (\$) are removed. Since no interpretation is done on the command string before it is read, inserting a backslash to escape a dollar sign has no effect. Backslashes that precede characters other than \, `` ` ``, ", **newline**, and **$** are left intact when the command string is read.

Parameter Substitution

The character **$** is used to introduce substitutable *parameters*. There are two types of parameters, positional and keyword. If *parameter* is a digit, it is a positional parameter. Positional parameters may be assigned values by **set**. Keyword parameters (also known as variables) may be assigned values by writing:

name=*value* [*name*=*value*] ...

Pattern matching is not performed on *value*. There cannot be a function and a variable with the same *name*.

${*parameter***}**
The value, if any, of the parameter is substituted. The braces are required only when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. If *parameter* is * or @, all the positional parameters, starting with **$1**, are substituted (separated by spaces). Parameter **$0** is set from argument zero when the shell is invoked.

${*parameter***:-***word***}**
If *parameter* is set and is non-null, substitute its value; otherwise substitute *word*.

${*parameter***:=***word***}**
If *parameter* is not set or is null set it to *word*; the value of the parameter is substituted. Positional parameters may not be assigned to in this way.

${*parameter***:?***word***}**
If *parameter* is set and is non-null, substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, the message "parameter null or not set" is printed.

${*parameter***:+***word***}**
If *parameter* is set and is non-null, substitute *word*; otherwise substitute nothing.

In the above, *word* is not evaluated unless it is to be used as the substituted string, so that, in the following example, **pwd** is executed only if **d** is not set or is null:

echo ${d:-`pwd`}

If the colon (:) is omitted from the above expressions, the shell only checks whether *parameter* is set or not.

The following parameters are automatically set by the shell:

The number of positional parameters in decimal.

- Flags supplied to the shell on invocation or by the **set** command.

? The decimal value returned by the last synchronously executed command.

$ The process number of this shell.

! The process number of the last background command invoked.

The following parameters are used by the shell:

HOME The default argument (home directory) for the *cd* command.

PATH The search path for commands (see *Execution* below). The user may not change **PATH** if executing under *rbsh*.

CDPATH
The search path for the *cd* command.

MAIL If this parameter is set to the name of a mail file *and* the **MAILPATH** parameter is not set, the shell informs the user of the arrival of mail in the specified file.

MAILCHECK
This parameter specifies how often (in seconds) the shell will check for the arrival of mail in the files specified by the **MAILPATH** or **MAIL** parameters. The default value is 600 seconds (10 minutes). If set to 0, the shell will check before each prompt.

MAILPATH
A colon (:) separated list of file names. If this parameter is set, the shell informs the user of the arrival of mail in any of the specified files. Each file name can be

followed by % and a message that will be printed when the modification time changes. The default message is *you have mail*.

PS1 Primary prompt string, by default "$ ".

PS2 Secondary prompt string, by default "> ".

IFS Internal field separators, normally **space**, **tab**, and **newline**.

SHACCT
If this parameter is set to the name of a file writable by the user, the shell will write an accounting record in the file for each shell procedure executed.

SHELL When the shell is invoked, it scans the environment (see "Environment" below) for this name. If it is found and 'rbsh' is the file name part of its value, the shell becomes a restricted shell.

The shell gives default values to **PATH**, **PS1**, **PS2**, **MAILCHECK** and **IFS**. **HOME** and **MAIL** are set by *login*(1).

Blank Interpretation

After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in **IFS**) and split into distinct arguments where such characters are found. Explicit null arguments ("" or '') are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

Input/Output

A command's input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a *simple-command* or may precede or follow a *command* and are *not* passed on as arguments to the invoked command. Note that parameter and command substitution occurs before *word* or *digit* is used.

<word Use file *word* as standard input (file descriptor 0).

>word Use file *word* as standard output (file descriptor 1). If the file does not exist it is created; otherwise, it is truncated to zero length.

>>word Use file *word* as standard output. If the file exists output is appended to it (by first seeking to the end-of-file); otherwise, the file is created.

<<[-]word After parameter and command substitution is done on *word*, the shell input is read up to the first line that literally matches the resulting *word*, or to an end-of-file. If, however, - is appended to <<:

1) leading tabs are stripped from *word* before the shell input is read (but after parameter and command substitution is done on *word*),
2) leading tabs are stripped from the shell input as it is read and before each line is compared with *word*, and
3) shell input is read up to the first line that literally matches the resulting *word*, or to an end-of-file.

If any character of *word* is quoted (see "Quoting," later), no additional processing is done to the shell input. If no characters of *word* are quoted:

1) parameter and command substitution occurs,
2) (escaped) **newline** is ignored, and
3) \ must be used to quote the characters \, **$**, and `.

The resulting document becomes the standard input.

<&digit Use the file associated with file descriptor *digit* as standard input. Similarly for the standard output using **>&digit**.

<&- The standard input is closed. Similarly for the standard output using >&-.

If any of the above is preceded by a digit, the file descriptor which will be associated with the file is that specified by the digit (instead of the default 0 or 1). For example:

... 2>&1

associates file descriptor 2 with the file currently associated with file descriptor 1.

The order in which redirections are specified is significant. The shell evaluates redirections left-to-right. For example:

... 1>*xxx* 2>&1

first associates file descriptor 1 with file *xxx*. It associates file descriptor 2 with the file associated with file descriptor 1 (i.e., *xxx*). If the order of redirections were reversed, file descriptor 2 would be associated with the terminal (assuming file descriptor 1 had been) and file descriptor 1 would be associated with file *xxx*.

Using the terminology introduced on the first page, under "Commands," if a *command* is composed of several *simple commands*, redirection will be evaluated for the entire *command* before it is evaluated for each *simple command*. That is, the shell evaluates redirection for the entire *list*, then each *pipeline* within the *list*, then each *command* within each *pipeline*, then each *list* within each *command*.

If a command is followed by & the default standard input for the command is the empty file **/dev/null**. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

Redirection of output is not allowed in the restricted shell.

File Name Generation

Before a command is executed, each command *word* is scanned for the characters *, ?, and [. If one of these characters appears the word is regarded as a *pattern*. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches the pattern, the word is left unchanged. The character . at the start of a file name or immediately following a /, as well as the character / itself, must be matched explicitly.

* Matches any string, including the null string.

? Matches any single character.

[...] Matches any one of the enclosed characters. A pair of characters separated by – matches any character lexically between the pair, inclusive. If the first character following the opening ``['' is a "!" any character not enclosed is matched.

Quoting

The following characters have a special meaning to the shell and cause termination of a word unless quoted:

; & ( ) | ^ < > newline space tab

A character may be *quoted* (i.e., made to stand for itself) by preceding it with a backslash (\) or inserting it between a pair of quote marks (' ' or ""). During processing, the shell may quote certain characters to prevent them from taking on a special meaning. Backslashes used to quote a single character are removed from the word before the command is executed. The pair **\newline** is removed from a word before command and parameter substitution.

All characters enclosed between a pair of single quote marks (' '), except a single quote, are quoted by the shell. Backslash has no special meaning inside a pair of single quotes. A single quote may be quoted inside a pair of double quote marks (for example, " ' ").

Inside a pair of double quote marks (""), parameter and command substitution occurs and the shell quotes the results to avoid blank interpretation and file name generation. If **$*** is within a pair of double quotes, the positional parameters are substituted and quoted, separated by quoted spaces ("**$1 $2** ..."); however, if **$@** is within a pair of double quotes, the positional parameters are substituted and quoted, separated by unquoted spaces ("**$1**" "**$2**" ...). \ quotes the characters \, `, ", and **$**. The pair **\newline** is removed before parameter and command substitution. If a backslash precedes characters other than \, `, ", **$**, and newline, then the backslash itself is quoted by the shell.

Prompting

When used interactively, the shell prompts with the value of **PS1** before reading a command. If at any time a newline is typed and further input is needed to complete a command, the secondary prompt (i.e., the value of **PS2**) is issued.

Environment

The *environment* (see *environ*(5)) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. If the user modifies the value of any of these parameters or creates new parameters, none of these affects the environment unless the **export** command is used to bind the shell's parameter to the environment (see also **set -a**). A parameter may be removed from the environment with the **unset** command. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, minus any pairs removed by **unset**, plus any modifications or additions, all of which must be noted in **export** commands.

The environment for any *simple-command* may be augmented by prefixing it with one or more assignments to parameters. Thus the following are equivalent, as far as the execution of *cmd* is concerned.

```
TERM=450 cmd
(export TERM; TERM=450; cmd)
```

If the **-k** flag is set, *all* keyword arguments are placed in the environment, even if they occur after the command name. The following first prints **a=b c** and **c**:

```
echo a=b c
set -k
echo a=b c
```

Signals

The INTERRUPT and QUIT signals for an invoked command are ignored if the command is followed by &; otherwise signals have the values inherited by the shell from its parent, with the exception of signal 11 (but see also the **trap** command below).

Execution

Each time a command is executed, the above substitutions are carried out. If the command name matches one of the *Special Commands* listed below, it is executed in the shell process. If the command name does not match a *Special Command*, but matches the name of a defined function, the function is executed in the shell process (note how this differs from the execution of shell procedures). The positional parameters **$1**, **$2**, ... are set to the arguments of the function. If the command name matches neither a *Special Command* nor the name of a defined function, a new process is created and an attempt is made to execute the command via *exec*(2).

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a colon (:). The default path is **:/bin:/usr/bin** (specifying the current directory, **/bin**, and **/usr/bin**, in that order). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign, between

two colon delimiters anywhere in the path list, or at the end of the path list. If the command name contains a / the search path is not used; such commands will not be executed by the restricted shell. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an **a.out** file, it is assumed to be a file containing shell commands. A subshell is spawned to read it. A parenthesized command is also executed in a subshell.

The location in the search path where a command was found is remembered by the shell (to help avoid unnecessary *execs* later). If the command was found in a relative directory, its location must be redetermined whenever the current directory changes. The shell forgets all remembered locations whenever the **PATH** variable is changed or the **hash -r** command is executed (see below).

Special Commands

Input/output redirection is now permitted for these commands. File descriptor 1 is the default output location.

: No effect; the command does nothing. A zero exit code is returned.

. *file* Read and execute commands from *file* and return. The search path specified by **PATH** is used to find the directory containing *file*.

break [*n*]
Exit from the enclosing **for** or **while** loop, if any. If *n* is specified break *n* levels.

continue [*n*]
Resume the next iteration of the enclosing **for** or **while** loop. If *n* is specified resume at the *n*-th enclosing loop.

cd [*arg*]
Change the current directory to *arg*. The shell parameter **HOME** is the default *arg*. The shell parameter **CDPATH** defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (:). The default path is **<null>** (specifying the current directory). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a / the search path is not used. Otherwise, each directory in the path is searched for *arg*. The *cd* command may not be executed by *rbsh*.

echo [*arg* ...]
Echo arguments. See *echo*(1) for usage and description.

eval [*arg* ...]
The arguments are read as input to the shell and the resulting command(s) executed.

exec [*arg* ...]
The command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, cause the shell input/output to be modified.

exit [*n*]
Causes a shell to exit with the exit status specified by *n*. If *n* is omitted the exit status is that of the last command executed (an end-of-file will also cause the shell to exit.)

export [*name* ...]
The given *names* are marked for automatic export to the *environment* of subsequently-executed commands. If no arguments are given, variable names that have been marked for export during the current shell's execution are listed. (Variable names exported from a parent shell are listed only if they have been exported again during the current shell's execution.) Function names are *not* exported.

fexpr Built in version of *expr*(1). The same syntax applies. If the System V Interface Definition is ever extended to allow *expr* itself to be a special (built in) command, *fexpr* may go away. To ensure upward compatibility, shells should check for the existence of *fexpr* before using it.

getopts
Use in shell scripts to support command syntax standards (see *intro*(1)); it parses positional parameters and checks for legal options. See *getopts*(1) for usage and description.

hash [**-r**] [*name* ...]
For each *name*, the location in the search path of the command specified by *name* is determined and remembered by the shell. The **-r** option causes the shell to forget all remembered locations. If no arguments are given, information about remembered commands is presented. *Hits* is the number of times a command has been invoked by the shell process. *Cost* is a measure of the work required to locate a command in the search path. If a command is found in a "relative" directory in the search path, after changing to that directory, the stored location of that command is recalculated. Commands for which this will be done are indicated by an asterisk (*) adjacent to the *hits* information. *Cost* will be incremented when the recalculation is done.

newgrp [*arg* ...]
Equivalent to **exec newgrp** *arg* .... See *newgrp*(1) for usage and description.

pwd Print the current working directory. See *pwd*(1) for usage and description.

read [*name* ...]
One line is read from the standard input and, using the internal field separator, **IFS** (normally space or tab), to delimit word boundaries, the first word is assigned to the first *name*, the second word to the second *name*, etc., with leftover words assigned to the last *name*. Lines can be continued using **\newline**. Characters other than **newline** can be quoted by preceding them with a backslash. These backslashes are removed before words are assigned to *names*, and no interpretation is done on the character that follows the backslash. The return code is 0 unless an end-of-file is encountered.

readonly [*name* ...]
The given *names* are marked *readonly* and the values of the these *names* may not be changed by subsequent assignment. If no arguments are given, a list of all *readonly* names is printed.

return [*n*]
Causes a function to exit with the return value specified by *n*. If *n* is omitted, the return status is that of the last command executed.

set [—**aefhkntuvx** [*arg* ...]]

- **-a** Mark variables which are modified or created for export.
- **-e** Exit immediately if a command exits with a non-zero exit status.
- **-f** Disable file name generation
- **-h** Locate and remember function commands as functions are defined (function commands are normally located when the function is executed).
- **-k** All keyword arguments are placed in the environment for a command, not just those that precede the command name.
- **-n** Read commands but do not execute them.
- **-t** Exit after reading and executing one command.
- **-u** Treat unset variables as an error when substituting.

BSH(1)

-v Print shell input lines as they are read.

-x Print commands and their arguments as they are executed.

— Do not change any of the flags; useful in setting **$1** to **-**.

Using **+** rather than **-** causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in **$-**. The remaining arguments are positional parameters and are assigned, in order, to **$1**, **$2**, ... If no arguments are given the values of all names are printed.

shift [*n*]
The positional parameters from **$n+1** ... are renamed **$1** .... If *n* is not given, it is assumed to be 1.

test Evaluate conditional expressions. See *test*(1) for usage and description.

times Print the accumulated user and system times for processes run from the shell.

trap [*arg*] [*n*] ...
The command *arg* is to be read and executed when the shell receives signal(s) *n*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. An attempt to trap on signal 11 (memory fault) produces an error. If *arg* is absent all trap(s) *n* are reset to their original values. If *arg* is the null string this signal is ignored by the shell and by the commands it invokes. If *n* is 0 the command *arg* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

type [*name* ...]
For each *name*, indicate how it would be interpreted if used as a command name.

ulimit [*n*]
Impose a size limit of *n* blocks on files written by the shell and its child processes (files of any size may be read). If *n* is omitted, the current limit is printed. You may lower your own ulimit, but only a superuser (see *su*(1M)) can raise a ulimit.

umask [*nnn*]
The user file-creation mask is set to *nnn* (see *umask*(1)). If *nnn* is omitted, the current value of the mask is printed.

unset [*name* ...]
For each *name*, remove the corresponding variable or function. The variables **PATH**, **PS1**, **PS2**, **MAILCHECK** and **IFS** cannot be unset.

wait [*n*]
Wait for your background process whose process ID is *n* and report its termination status. If *n* is omitted, all your shell's currently active background processes are waited for and the return code will be zero.

Invocation

If the shell is invoked through *exec*(2) and the first character of argument zero is -, commands are initially read from */etc/profile* and from *$HOME/.profile*, if such files exist. Thereafter, commands are read as described below, which is also the case when the shell is invoked as */bin/sh*. The flags below are interpreted by the shell on invocation only; Note that unless the **-c** or **-s** flag is specified, the first argument is assumed to be the name of a file containing commands, and the remaining arguments are passed as positional parameters to that command file:

-c *string* If the **-c** flag is present commands are read from *string*.

-s If the **-s** flag is present or if no arguments remain commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output (except for *Special Commands*) is written to file descriptor 2.

-i If the **-i** flag is present or if the shell input and output are attached to a terminal, this shell is *interactive*. In this case TERMINATE is ignored (so that **kill 0** does not kill an interactive shell) and INTERRUPT is caught and ignored (so that **wait** is interruptible). In all cases, QUIT is ignored by the shell.

-r If the **-r** flag is present the shell is a restricted shell.

The remaining flags and arguments are described under the **set** command above.

rbsh Only

rbsh is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of *rbsh* are identical to those of *bsh*, except that the following are disallowed:

changing directory (see *cd*(1)),
setting the value of **$PATH**,
specifying path or command names containing **/**,
redirecting output (> and >>).

The restrictions above are enforced after **.profile** is interpreted.

A restricted shell can be invoked in one of the following ways: (1) *rbsh* is the file name part of the last entry in the */etc/passwd* file (see *passwd*(4)); (2) the environment variable **SHELL** exists and *rbsh* is the file name part of its value; (3) the shell is invoked and *rbsh* is the file name part of argument 0; (4) the shell is invoke with the **-r** option.

When a command to be executed is found to be a shell procedure, *rbsh* invokes *sh* to execute it. Thus, it is possible to provide to the end user shell procedures that have access to the full power of the standard shell, while imposing a limited menu of commands; this scheme assumes that the end user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the **.profile** (see *profile*(4)) has complete control over user actions by performing guaranteed setup actions and leaving the user in an appropriate directory (probably *not* the login directory).

The system administrator often sets up a directory of commands (e.g., **/usr/rbin**) that can be safely invoked by a restricted shell. Some systems also provide a restricted editor, *red*.

EXIT STATUS

Errors detected by the shell, such as syntax errors, cause the shell to return a non-zero exit status. If the shell is being used non-interactively execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also the **exit** command above).

FILES

/etc/profile
$HOME/.profile
/tmp/sh*
/dev/null

SEE ALSO

cd(1), echo(1), env(1), getopts(1), intro(1), login(1), newgrp(1), pwd(1), test(1), umask(1), wait(1), dup(2), exec(2), fork(2), pipe(2), signal(2), ulimit(2), profile(4).

WARNING

Words used for filenames in input/output redirection are not interpreted for filename generation (see "File Name Generation," above). For example, **cat file1 >a*** will create a file named **a***.

Because commands in pipelines are run as separate processes, variables set in a pipeline have no effect on the parent shell.

BSH(1)

If you get the error message *cannot fork, too many processes*, try using the *wait*(1) command to clean up your background processes. If this doesn't help, the system process table is probably full or you have too many active foreground processes. (There is a limit to the number of process IDs associated with your login, and to the number the system can keep track of.)

BUGS

If a command is executed, and a command with the same name is installed in a directory in the search path before the directory where the original command was found, the shell will continue to *exec* the original command. Use the **hash** command to correct this situation.

If you move the current directory or one above it, **pwd** may not give the correct response. Use the **cd** command with a full path name to correct this situation.

Not all the processes of a 3- or more-stage pipeline are children of the shell, and thus cannot be waited for.

For *wait n*, if *n* is not an active process ID, all your shell's currently active background processes are waited for and the return code will be zero.

NAME

cal – print calendar

SYNOPSIS

cal [[*month*] *year*]

DESCRIPTION

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

EXAMPLES

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

BUGS

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

Beware that "cal 83" refers to the early Christian era, not the 20th century.

NAME

calendar – reminder service

SYNOPSIS

calendar [-]

DESCRIPTION

calendar consults the file **calendar** in the current directory and prints out lines that contain today's or tomorrow's date anywhere in the line. Most reasonable month-day dates, e.g., "Aug. 24," "august 24," "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 operating system facilities.

FILES

/usr/lib/calprog	to figure out today's and tomorrow's dates
/etc/passwd	
/usr/lib/caldir/*	

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

cat – concatenate and print files

SYNOPSIS

cat [–**u**] [–**s**] [–**v** [–**t**] [–**e**]] *file* ...

DESCRIPTION

cat reads each *file* in sequence and writes it on the standard output. Thus:

cat file

prints **file** on your terminal, and:

cat file1 file2 >file3

concatenates **file1** and **file2**, and writes the results in **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, newlines 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 used with the **–v** option, the following options may 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 (before the newline).

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

WARNING

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:

cat file1 file2 >file1

will cause the original data in **file1** to be lost.

SEE ALSO

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

NAME

cb – C program beautifier

SYNOPSIS

cb [**–s**] [**–j**] [**–l** *leng*] [*file* ...]

DESCRIPTION

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

cb accepts the following options.

–s
Canonicalizes the code to the style of Kernighan and Ritchie in *The C Programming Language*.

–j
Causes split lines to be put back together.

–l *leng*
Causes *cb* to split lines that are longer than *leng*.

SEE ALSO

cc(1).
The C Programming Language. Prentice-Hall, 1978.

BUGS

Punctuation that is hidden in preprocessor statements causes indentation errors.

NAME

cc – GLS C compiler

SYNOPSIS

cc [*options*] *files*

DESCRIPTION

The *cc* command is the interface to the GLS C Compilation System. The compilation tools consist of a preprocessor, compiler, assembler, and linker. By default, the GLS C compiler produces assembly code and, unless certain compile options are specified, the compilation process will invoke the assembler step. The *cc* command processes the supplied options, then executes the various tools with the proper arguments. The *cc* command accepts several types of arguments:

Arguments whose names end with *.c* are C source programs; they are compiled and left in a *.o* file in the working directory.

Arguments whose names end with *.s* are assembly source programs; they are assembled and left in a *.o* file in the working directory.

Other arguments are taken to be link editor option arguments, C-compatible object programs, or object program libraries.

The *.o* file is deleted if a single source file is compiled and linked.

The *.s* files created for each module usually are deleted.

The following options are interpreted by *cc*:

–c Do not produce executable files; produce only object files. For each source file specified, compile the source language file into object code output and place the output in a file whose name is the same as the input source file but which ends with *.o* rather than the suffix of the input source file.

–g Generate source level symbolic debugger information in the object file, for use with a debugger such as *sdb*(1).

–ga Generate a frame for stack traces. The default compiler setting is to optimize the program to the point that stack traces become impossible on some machines. This makes program debugging difficult. Specify this option when debugging a program. (This option does not imply option **–g**.)

–o *filename*

Place the executable (or, if **–c** is specified, object file) output into *filename*. If this option is not specified, *cc* names the executable file *a.out*. If **–c** is specified and **–o** is not used, the object filename will be the same as the source filename with the suffix changed to *.o*. *cc* displays an error message to *stderr* if *filename* is suffixed with one of the following: *.F*, *.f*, *.c*, *.i*, *.p*, or *.s*. This protects source files with those suffixes from being overwritten. This option is ignored if you specify **–S**, or if you specify **–c** with more than one source file.

–p, **–qp**, or **–pg**

Produce code that counts the number of times each routine is called. If link editing takes place, link profiled versions of *libc.a* and *libm.a* (with **–lm** option) and call *monitor*(3C). Produce a *mon.out* file at normal termination of execution of the object program. An execution profile can then be generated with *prof*(1). **–pg** produces a monitor output file for the Berkley profiler *gprof*(1).

–v Display the command line arguments on *stderr* during each step of the compilation process.

-w Suppress warning messages.

-ansi Compile programs substantially according to the ANSI C specification. This adds another directory to the search path for include files, as well as other libraries. Use the **-X316** switch if you merely want the language changes.

-ANSI This is similar to **-ansi** except full conformance is supported. In particular, floating-point arithmetic may be slower. Use the **-X153** switch if you merely want the language changes.

-noansi Compiles programs according to traditional UNIX C syntax and semantics.

-C By default, the preprocessing compile step strips C-style comments. Specifying **-C** will cause comments to be left in the preprocessor output. This option will be passed to the GLS C compiler because it has its own internal preprocessor. If **-P** has been specified, then this option will be passed to *cpp*(1).

-D*name*[=*string*]
If *string* is not specified, define *name* to *cpp*(1) with the value 1. If *string* is specified, *name* is defined to *cpp*(1) with the value of *string*. This is equivalent to placing the directive `#define` *name string* at the beginning of the source file.

-E Do not compile the program; instead, place the output of the preprocessor on the standard output file. This is useful for debugging preprocessor macros.

-H Display the names of the files on *stderr* as they are opened. This is useful for finding out why the compiler cannot find an include file. This option invokes **-X32** and passes it to the compiler.

-I*dir* Search for include file names that are not absolute (do not start with /) in directory *dir* before a standard list of directories. The standard list of directories searched is your current directory, then the */usr/include* directory. To search more than one directory, specify **-I***dir* for each directory.

-O Activate the following optimizers: static address elimination, loop invariant analysis, strength reduction, and peephole. These optimizations are safe for programs that are not loop bound. Routines with more than a thousand lines of code may require large amounts of memory to compile. If memory constraints are a concern, do not use the **-O** compile option, and break large routines into several smaller routines.

-OA Apply algorithmic changes to the program. Currently, this will change the behavior during full ANSI C compiles to maintain floating-point results in floating-point registers.

-OL Optimize the program to be as fast as possible, even if this requires making the program larger. Most of the available resources are allocated to optimizations of the innermost loops. **-OL** causes scalar multiplies to be performed in-line and performs larger block moves sequentially. It is counterproductive to specify **-OL** on code that contains no loops or that is rarely executed, because it will make the whole program larger but not faster.

-OM Allow the optimizer to assume that memory locations do not change except by explicit stores. No memory locations referenced by the compiled program are I/O device registers. These memory locations cannot be changed by external hardware or concurrent processes. **-OM** must be used with caution with device drivers, operating systems, shared memory environments, and interrupts or signals.

-OMLA This option is equivalent to **-OM** and **-OL** and **-OA**. The *M*, the *L*, and the *A* may be specified in any order, or one or more of them may be omitted to disable the corresponding optimization option.

-P Run only *cpp*(1) on the named C programs and store the result in the corresponding files that end with *.i*.

-S Compile the named programs and leave the assembly language output on corresponding files suffixed with *.s*.

-T Truncate symbols/defines to 8 characters. This is also a preprocessor *cpp*(1) option.

-U*name* Undefine the predefined preprocessor symbol *name* for this compilation. This option is equivalent to putting **#undef** *name* at the beginning of the source file.

-V Display the version of the link editor and/or assembler (if it is invoked).

-W*c*,*arg1*[,*arg2*]...
Pass argument[s] *argi* to pass *c* where *c* is one of [p0al] interpreted as for the **-Y** option. For example: **-Wa,-m** passes **-m** to the assembler.

-X*nnn* Set option *nnn* (where *nnn* is an unsigned integer constant). To reset an option set with **-X** *nnn*, use **-Z** *nnn*. Not all **-X** options are listed in the man pages. A complete list can be found in the *GLS Programming Guide for 97xx Systems*.

-X18 Only declared *register* variables are allocated in registers.

-X168 Do not move invariant floating-point expressions out of loops.

-X153 Support the full ANSI language in the compiler, without using the ANSI libraries or include files.

-Y[*p0alSILU*],*dirname*
Specify a new pathname, *dirname*, for the locations of the tools and directories designated in the first argument. [*p0alSILU*] represents:

- *p* preprocessor *cpp*(1); applicable only when using the **-P** option
- *0* GLS C compiler
- *a* assembler
- *l* link editor
- *S* directory containing the startup routines
- *I* default include directory searched for by *cpp*(1)
- *L* first default library directory searched for by *ld*(1)
- *U* second default library directory searched for by *ld*(1)

-Z*nnn* Turn off option number *nnn*. This is the reverse of the **-X** option. This option is useful for turning off options that are on by default.

FILES

file.c	C source file
file.o	object file
file.s	assembly language file
a.out	link edited output (executable)
mon.out	profile output for use by *prof*(1)
/bin /cc	driver for compiler
/usr/lib/ccom88	compiler
/lib/crt0.o	startup routine
/lib/mcrt0.o	profiling startup routine
/lib/libm.a	standard math library
/lib/libc.a	standard C library
/usr/include	standard directory for #include files

SEE ALSO

B. W. Kernighan and D. M. Ritchie, *The C Programming Language,* Prentice-Hall, 1978
B. W. Kernighan, *Programming in C—A tutorial*
D. M. Ritchie, *C Reference Manual*
GLS Programming Guide for 97xx Systems
as(1), prof(1), sdb(1), ld(1).

DIAGNOSTICS

The diagnostics produced by the GLS CC compiler are intended to be self-explanatory and are similar to those produced by other C compilers. Occasional messages may be produced by the assembler or link editor.

BUGS

Routines with more than 1,000 lines of code may require large amounts of memory to compile. If memory constraints are a problem, do not use the **-O** option.

NAME

cd – change working directory

SYNOPSIS

cd [*directory*]

DESCRIPTION

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

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

SEE ALSO

pwd(1), sh(1).

NAME

cdc – change the delta commentary of an SCCS delta

SYNOPSIS

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

DESCRIPTION

cdc changes the *delta commentary*, for the SID (**S**CCS **ID**entification string) specified by the **–r** keyletter, of each named SCCS file.

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

If a directory is named, *cdc* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the 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 file names.

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

–r*SID*

Used to specify the *S*CCS *ID*entification (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). 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 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 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

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

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

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

does the same thing.

WARNINGS

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

FILES

x-file (see *delta*(1))
z-file (see *delta*(1))

SEE ALSO

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

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

cflow – generate C flowgraph

SYNOPSIS

cflow [–**r**] [–**ix**] [–**i_**] [–d *num*] *files*

DESCRIPTION

The *cflow* command analyzes a collection of C, yacc, lex, assembler, and object files and attempts to build a graph charting the external references. Files suffixed with **.y**, **.l**, and **.c** are yacced, lexed, and C-preprocessed as appropriate. The results of the preprocessed files, and files suffixed with **.i**, are then run through the first pass of *lint*(1). Files suffixed with **.s** are assembled. Assembled files, and files suffixed with **.o**, have information extracted from their symbol tables. The results are collected and turned into a graph of external references which is displayed upon the standard output.

Each line of output begins with a reference number, followed by a suitable number of tabs indicating the level, then the name of the global symbol followed by a colon and its definition. Normally only function names that do not begin with an underscore are listed (see the –**i** options below). For information extracted from C source, the definition consists of an abstract type declaration (e.g., **char** *), and, delimited by angle brackets, the name of the source file and the line number where the definition was found. Definitions extracted from object files indicate the file name and location counter under which the symbol appeared (e.g., *text*). Leading underscores in C-style external names are deleted.

Once a definition of a name has been printed, subsequent references to that name contain only the reference number of the line where the definition may be found. For undefined references, only < > is printed.

As an example, given the following in *file.c*:

```
int     i;

main()
{
        f();
        g();
        f();
}
f()
{
        i = h();
}
```

The command:

cflow –ix file.c

Produces the output:

```
1   main: int(), <file.c 4>
2       f: int(), <file.c 11>
3               h: <>
4               i: int, <file.c 1>
5       g: <>
```

When the nesting level becomes too deep, the output of *cflow* can be piped to *pr*(1), using the –e option, to compress the tab expansion to something less than every eight spaces.

In addition to the –**D**, –**I**, and –**U** options (which are interpreted just as they are by *cc*(1) and *cpp*(1)), the following options are interpreted by *cflow*:

-r
Reverse the "caller:callee" relationship producing an inverted listing showing the callers of each function. The listing is also sorted in lexicographical order by callee.

-ix
Include external and static data symbols. The default is to include only functions in the flowgraph.

-i_
Include names that begin with an underscore. The default is to exclude these functions (and data if *-ix* is used).

-d*num*
The *num* decimal integer indicates the depth at which the flowgraph is cut off. By default, this is a very large number. Attempts to set the cutoff depth to a nonpositive integer will be ignored.

DIAGNOSTICS

Complains about bad options. Complains about multiple definitions and only believes the first. Other messages may come from the various programs used (e.g., the C-preprocessor).

SEE ALSO

as(1), cc(1), cpp(1), lex(1), lint(1), nm(1), pr(1), yacc(1).

BUGS

Files produced by *lex*(1) and *yacc*(1) cause the reordering of line number declarations which can confuse *cflow*. To get proper results, feed *cflow* the *yacc* or *lex* input.

NAME

chgrp – change group

SYNOPSIS

chgrp [-f -R] group file ...

DESCRIPTION

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

The user invoking *chgrp* must belong to the specified group and be the owner of the file, or be the super-user. If *chgrp* is invoked by other than the superuser, the set-user-ID and set-group-ID bits of the file mode, 04000 and 02000 respectively, will be cleared.

No errors are reported when the **–f** (force) option is given.

When the **–R** option is given, *chgrp* recursively descends its directory arguments setting the specified group-ID.

FILES

/etc/group

SEE ALSO

chgrp(1), chown(1B), passwd(4), group(4).

NAME

chk – file system check and interactive repair

SYNOPSIS

chk [*options*] [*alias*]

DESCRIPTION

chk checks to see that *alias* is in the *permissions* file and that file system check permission is given. Read-only check permission is indicated by an **R** only in the "perms" field of the *permissions* file; read/write permission is indicated by a **W**. *chk* takes the same options as *fsck*(1M).

If neither options nor *alias* is given, *chk* assumes the default alias, **floppy**.

FILES

/etc/fsck
/etc/filesys permissions file

SEE ALSO

fsck(1M), filesys(4).

NAME

chkyn – get yes/no response from user or check answer to question

SYNOPSIS

chkyn [**–ynfeid!**] [**–D** *defaultvalue*] [**–k** *pid*] [**–q** *inputvalue*]
[**–Q** *aliasname*] [**–h** *helpstring*] [**–H** *helpmessage*]
[**–m** *number*] *Prompt String*

chkyn [**–c**] *Prompt String answer ...*

chkyn [**–r**] *Prompt String answer message ...*

chkyn [**–R**] *Prompt String answer message ...*

DESCRIPTION

chkyn is a program that is provided primarily for use by the *sysadm* System Administration package. However, as this utility is also very useful in a variety of shell script applications, it is documented here.

chkyn prints out the *Prompt String*, followed by the possible answers, and asks the user for input. The handling of user input is controlled by the command line options.

chkyn exits with an exit value of **0** for yes or **1** for no.

The following command line options are supported by *chkyn:*

–y
default answer to prompt is **yes**.

–n
default answer to prompt is **no**.

–f
force an answer. If the answer supplied by the user is not valid, ask the question again, and continue asking until a valid answer is obtained.

–e
echo the answer. This allows a shell script to capture the response which the user typed in.

–c *Prompt String answer ...*
check a list. This is often used in conjunction with the **–f** option, which forces an answer. *chkyn* prompts the user with the prompt string (which does not automatically include a question mark), then expects a reply that matches one of the *answer* fields. The exit code of *chkyn* indicates which item from the list of answers was chosen; an exit code that is higher than the number of items in the list indicates that the user's entry did not match any of the list items.

–r *Prompt answer message ...*
accept matching regular expressions. This is most useful with the **–f** option, which forces an answer. The user supplies a regular expression in *answer*; *chkyn* prints *message* if a non-matching answer is given.

–R
reject matching regular expressions. This works the same as **–r**, except that if an *answer* from the list is matched, *chkyn* prints the corresponding *message* and considers this an invalid answer.

–i
match an integer expression. This requires the *bs* utility, which is not supported on REAL/IX systems.

–d
match a decimal expression. This requires the *bs* utility, which is not supported on REAL/IX systems.

–D *value*
the default value for an empty input. Must be accompanied by a **–c**, **–r**, **–i** or **–d** option.

–k *pid*
quit sends SIGTERM to this process id. Used with the **–q** or **–Q** options, which define *quit*.

–q *input*
input that causes quit.

–Q *alias*
alias name for quit.

–h *helpstring*
string to request help. **–h** and **–H** must both be specified.

–H *helpmessage*
help message to display. **–h** and **–H** must both be specified.

–m *number*
maximum number of times to ask a question.

–!
permit shell escapes from prompt.

DIAGNOSTICS

Exit value from *chkyn* is basically **0** for true or **1** for false. When a list of acceptable answers is provided (as with the **–c** option), the exit code points to the matched item in the list; a higher exit code indicates no item was matched.

NOTES

Since *chkyn* is part of the *sysadm* package, the options and command syntax may change in future releases to support the requirements of *sysadm*. User scripts that depend on certain functions of *chkyn* would need to be rewritten at that time.

The meaning of multiple answer/message pairs with the **–r** option is not at all clear.

NAME

chmod – change mode

SYNOPSIS

chmod *mode file ...*

chmod *mode directory ...*

DESCRIPTION

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

chmod *nnn file(s)*

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

chmod *a operator b file(s)*

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

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

4000	set user ID on execution
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 UID or GID. Permissions are described in three sequences each having three characters:

User	Group	Other
rwx	rwx	rwx

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

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

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

A command line using the symbolic method would appear as:

chmod g+rw *file*

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

The *who* part can be stated as one or more of the following letters:

u	user's permissions
g	group's permissions
o	others permissions

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

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

permission is any compatible combination of the following letters:

r	reading permission
w	writing permission
x	execution permission
s	user or group set-ID is turned on
t	sticky bit is turned on
l	mandatory locking will occur during access

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

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

chmod g+x,+l *file*

chmod g+s,+l *file*

As long as the user has the appropriate privilege, if the file is a symbolic link, the file pointed to by the symbolic link (not the symbolic-link file itself) is affected.

Only the owner of a file or directory (or the superuser) may change a file's mode. Only the superuser may 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.

EXAMPLES

chmod a−x *file*

chmod 444 *file*

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

chmod go+rw *file*

chmod 066 *file*

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

chmod +l *file*

This causes a file to be locked during access.

chmod =rwx,g+s *file*

chmod 2777 *file*

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

NOTES

In a RFS environment, you may not have the permissions that the output of the **ls** –l command leads you to believe.

SEE ALSO

ls(1), chmod(2).

NAME

chmod – change mode

SYNOPSIS

chmod [**–Rf**] mode file ...

DESCRIPTION

The mode of each named file is changed according to *mode*, which may be absolute or symbolic. An absolute *mode* is an octal number constructed from the OR of the following modes:

4000	set user ID on execution
2000	set group ID on execution
1000	sticky bit, 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

A symbolic *mode* has the form:

[*who*] *op permission* [*op permission*] ...

The *who* part is a combination of the letters **u** (for user's permissions), **g** (group) and **o** (other). The letter **a** stands for all, or **ugo**. If *who* is omitted, the default is *a* but the setting of the file creation mask (see umask(2)) is taken into account.

Op can be **+** to add *permission* to the file's mode, **–** to take away *permission* and **=** to assign *permission* absolutely (all other bits will be reset).

Permission is any combination of the letters **r** (read), **w** (write), **x** (execute), **X** (set execute only if file is a directory or some other execute bit is set), **s** (set owner or group id) and **t** (save text – sticky). Letters **u**, **g**, or **o** indicate that *permission* is to be taken from the current mode. Omitting *permission* is only useful with **=** to take away all permissions.

When the **–R** option is given, *chmod* recursively descends its directory arguments setting the mode for each file as described above.

If the **–f** option is given, *chmod* will not complain if it fails to change the mode on a file.

EXAMPLES

The first example denies write permission to others, the second makes a file executable by all if it is executable by anyone:

```
chmod o–w file
chmod +X file
```

Multiple symbolic modes separated by commas may be given. Operations are performed in the order specified. The letter **s** is only useful with **u** or **g**.

Only the owner of a file (or the super-user) may change its mode.

SEE ALSO

ls(1), chmod(1), chmod(2), stat(2), umask(2).

NAME

chown, chgrp – change owner or group

SYNOPSIS

chown [**–h**] *owner file ...*

chown [**–h**] *owner directory ...*

chgrp [**–h**] *group file ...*

chgrp [**–h**] *group directory ...*

DESCRIPTION

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

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

If either command is invoked by other than the superuser, the set-user-ID and set-group-ID bits of the file mode (04000 and 02000, respectively) are cleared.

Only the owner of a file (or the superuser) may change the owner or group of that file. If _POSIX_CHOWN_RESTRICTED is in effect (system configuration option, off by default), only the superuser may change the owner of a file. A user invoking *chgrp* must belong to the specified group and be the owner of the file or be the superuser.

If *file* is a symbolic link and the **–h** flag is not given, the file to which the link points is changed. If the **–h** flag is given, the link is changed and not the file to which it points.

FILES

/etc/passwd
/etc/group

NOTES

In an RFS environment, you may not have the permissions that the output of the **ls –l** command leads you to believe.

DIAGNOSTICS

chgrp(1) outputs one of the following messages if it finds an incorrect entry in the **/etc/group** file:

fgetgrent: too many users in group < *group*>
fgetgrent: line exceeds system limit (*limit*)

where:

group
is the invalid entry

limit
is the system limit for the length of a line.

The output of one of these messages does not result in the failure of the command.

SEE ALSO

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

NAME

chown – change owner

SYNOPSIS

/etc/chown [**–f –R**] owner[.group] file ...

DESCRIPTION

Chown changes the owner of the *files* to *owner*. The owner may be either a decimal UID or a login name found in the password file. An optional group may also be specified. The group may be either a decimal GID or a group name found in the group-ID file.

If *chown* is invoked by other than the superuser, the set-user-ID and set-group-ID bits of the file mode, 04000 and 02000 respectively, will be cleared. Only the owner of a file (or the superuser) may change the owner or group of that file. No errors are reported when the **–f** (force) option is given.

When the **–R** option is given, *chown* recursively descends its directory arguments setting the specified owner.

FILES

/etc/passwd

SEE ALSO

chgrp(1B), chown(1), chown(2), passwd(4), group(4).

NAME

cmp – compare two files

SYNOPSIS

cmp [**–l**] [**–s**] *file1 file2*

DESCRIPTION

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

Options:

–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

coffcmp – compare common object format modules

SYNOPSIS

coffcmp [**–h**] [**–s**] *file1 file2*

DESCRIPTION

coffcmp is a utility used to compare common object module format files. It is intended primarily as an aid for the compiler, assembler, and/or loader maintenance person.

coffcmp compares two **COFF** files, ignoring the **timdat** field in the header. As a result, if two **COFF** files are identical except for the date and time they were compiled, *coffcmp* indicates that they are identical.

The following are the *coffcmp* options:

–h
print this help information.

–s
silent mode, compare and return code only.

DIAGNOSTICS

filename can't open
indicates that *coffcmp* was unable to open one or both of the files specificed on the command line.

*filename*seek*error*
indicates that *coffcmp* was unable to seek to a specified location in the file. In general, you should never see this error message, but it could occur if you are attempting to compare two large **COFF** files.

filename1 filename2 different at offset1 offset2
indicates that the two files are different. The offset value given is the first byte found that is different in the two specified files.

internal error; bad error *number*
indicates that *coffcmp* has encountered a situation that it is not prepared to deal with. If you see this error message, please report the situation that caused it to appear to your Customer Service Representative.

NAME

coffdump – dump common object format modules

SYNOPSIS

coffdump [*options*] *files...*

DESCRIPTION

coffdump is a utility used to dump common object module format files in a human readable form. It is intended primarily as an aid for the compiler, assembler, and/or loader maintenance person.

The following are the *coffdump* options:

–D
output is suitable for diff

–H
print this help information

–a
equivalent to **–cfhlorst**

–c
dump the string table

–f
dump each file header

–h
dump section headers

–l
dump line number information

–o
dump each optional header

–r
dump relocation information

–s
dump section contents

–t
dump symbol table entries

The default options are **–fho**.

DIAGNOSTICS

filename can't open
error message that indicates *coffdump* was unable to open one or more of the files specified on the command line.

can't read *number* section header
error message that indicates *coffdump* was unable to read one of the section headers in the object file. It probably means the object file being dumped is in an old format and the headers in the file are not formatted correctly for *coffdump*.

can't read relocation entry *number*
Same as section header message.

can't read line number entry *number*
Same as section header message.

can't read symbol table entry *number*
Same as section header message.

internal error; bad error *number*
error message that indicates there is an internal problem in *coffdump* that a user does not normally see. If this error occurs, report it to your Customer Service Representataive as a bug.

NAME

col – filter reverse linefeeds

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-linefeeds (**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-linefeeds (**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, newline, **SI**, **SO**, **VT** (\013), and **ESC** followed by **7**, **8**, or **9**. The **VT** character is an alternate form of full reverse linefeed, included for compatibility with some earlier programs of this type. All other nonprinting 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.

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, otherwise **–Tlp**.

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

colrm – remove columns from a file

SYNOPSIS

colrm [startcol [endcol]]

DESCRIPTION

Colrm removes selected columns from a file. Input is taken from standard input. Output is sent to standard output.

If called with one parameter the columns of each line will be removed starting with the specified column. If called with two parameters the columns from the first column to the last column will be removed.

Column numbering starts with column 1.

SEE ALSO

expand(1B), cut(1).

NAME

comb – combine SCCS deltas

SYNOPSIS

comb *files*

DESCRIPTION

comb generates a shell procedure (see *sh*(1)) which, when run, will reconstruct the given SCCS files. The reconstructed files will, hopefully, be smaller than the original files. The arguments may be specified in any order, but all keyletter arguments apply to all named SCCS files. If a directory is named, *comb* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the 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 following describes keyletter arguments; 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:

-p *SID*
The SID of the oldest delta to be preserved. All older deltas are discarded in the reconstructed file.

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

-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 file name, size (in blocks) after combining, original size (also in blocks), and percentage change computed by:

100 * (original – combined) / original

It is recommended that before any SCCS files are actually combined, determine exactly how much space is saved by the combining process.

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

EXAMPLE

comb s.file1 > tmp1

produces a shell script saved in **tmp1** that will remove from the SCCS-format file **s.file1** all deltas previous to the last set of changes, i.e., removes the capability of returning to earlier versions.

FILES

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

SEE ALSO

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

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

comb may rearrange the shape of the tree of deltas. It may not save any space; in fact, it is possible for the reconstructed file to actually be larger than the original.

NAME

comm – select or reject lines common to two sorted files

SYNOPSIS

comm [– [**123**]] *file1 file2*

DESCRIPTION

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

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

SEE ALSO

cmp(1), diff(1), sort(1), uniq(1).

NAME

pshl, conman – pseudo shell layer, console manager

SYNOPSIS

pshl [-f file] [-h] [-H home] [-P path] [-p priority] [-S shell] [-T term] [-v]
conman [-f file] [-h] [-H home] [-P path] [-p priority] [-S shell] [-T term] [-v]

DESCRIPTION

pshl (conman) allows a user to interact with more than one shell or terminal session from a single terminal. The user controls these shells or sessions, known as *layers*, using the commands described below. Note that *pshl (conman)* is not the job control program, *shl*, but is an alternative when efficiency is less critical than other functions made available. These functions, including screen refresh and logging layers, are available as options. The difference between invoking the layer manager as *pshl* or *conman* is primarily the function of the **lock** command, which permits increased security in some applications.

The *current (foreground) layer* is the layer which can receive input from the keyboard. Other (*background*) layers attempting to read from the keyboard are blocked. *Background* layers wishing to write to the screen are blocked when the **block** option is set, like *shl*. Unlike *shl*, when **noblock** is set, output from multiple layers is *not* simultaneously multiplexed onto the terminal screen; rather, the layer's output is either spooled and appears on the screen when the layer becomes current, or the background screen is silently updated; this is determined by an *argument* to **noblock**. When the output of a non-current (background) layer is important and should be viewed immediately, the **alarm** option can be set; if the current foreground layer is not so enabled, the alarm layer will automatically become the current layer when it has pending output for the screen.

The *swtch* character (see *termio*(7)) is ^Z (control Z); in the current implementation, only this character can be used as the *swtch* character.

The *swtch* character can be used in various ways:

When in a layer, the *swtch* character must be used to return to the manager.

When in the manager, the *swtch* character may be used to return to a layer, by terminating certain command lines with this character. Such command lines either contain a layer-switch command (one of **resume**, **toggle** or **next**), or no command at all. In the former case, the specified command is both executed and used to redefine the default layer-switch command, and thus shares some of the semantics of the **switch** command; in the latter case, the default layer-switch command, as currently defined, is invoked. (In either case, a layer may be optionally specified.) Note that explicit layer-switch commands need not be terminated by the *swtch* character, unless implicit redefinition of the default is also desired.

The manager prompts with its invocation name *(pshl, conman)*.

A *layer* is bound to a pseudo tty device (i.e., a *pty*, rather than an *sxt*, as in *shl*). The pseudo device can be manipulated like a real tty device using *stty*(1) and *ioctl*(2). Each layer has its own process group id.

Prior to accepting commands from the keyboard, the "rc" file is executed, which allows the user to initialize defaults and establish layers. If invoked as **pshl**, then *.pshlrc* is executed if present in the user's home directory; if invoked as **conman**, then *.conmanrc* is executed if present in the user's home directory.

Options

The *-f* option specifies a file containing commands to be executed prior to soliciting commands from the keyboard. The contents of such a file is similar to *.pshlrc* or *.conmanrc*.

The *-h* option requests usage to be printed, and then exits.

The *-H* option permits a home directory to be specified, rather than defaulting to $HOME.

The *-i* option requests some interactive initialization.

The *-p* option permits the scheduling priority for the layer manager to be specified. Setting priority requires privilege.

The *-P* option permits a path to be specified, in lieu of $PATH.

The *-S* option permits specifying the shell, rather than defaulting to $SHELL. If no shell is defined by this option and none is defined in the environment, /bin/ksh is used as the default.

The *-T* option permits specifying the actual terminal device, rather than defaulting to $TERM.

The *-v* option requests additional initialization messages (verbose).

Definitions

A *layer name* is a sequence of characters delimited by a blank, tab or new-line. It may be used to reference the layer in all commands which permit or require one or more layers to be specified. The *names* **(0)** through **(7)** cannot be used when creating a layer. They are default names that are assigned by the manager when the layer is created, and permit each layer to be uniquely specified in appropriate commands. They may be abbreviated to just the digit, as when using the *swtch* character to invoke the default layer-switch command, or when using an explicit layer-switch command. Each layer also has a non-numeric name which is the same as the name of the program created in the layer, unless explicitly specified via the **name** command or option when the layer is created. Note that non-numeric names may not be unique by default, and thus may not allow layers sharing the name to be uniquely specified.

Commands

The following commands may be issued from the layer manager's prompt level. Abbreviations are not permitted. Layers are referenced by their name or number, as described above. Note that many commands may also be used as options to **create** or **set**, as indicated below.

create [*options*] [*name*]

Create a layer executing the program *name*, and make it the current layer. The default program is $SHELL, or the *login* program (see **lock** command). The layer's alphabetic name defaults to the name of the program; it may be overridden by specifying the **name** option. Many of the other commands can be specified on the create command as options; when this is done, each option must precede the program name if specified. Options permitted are: **[no]alarm**, **block**, **noblock**[=**[no]spool**], **log**[=*string*], **nolog**, **[no]refresh**, **[no]resume**, **pri**[=*value*], **nopri**, **name**=*string*, **[no]delete**, **[no]respawn**.

set [*options*]

Set default creation options for newly created layers. If no options are given on the command line, the current default options are printed. Options permitted are as for the **create** command above, excluding the **name** option.

resume [*layer*]

This command is one of the layer-switch commands. **Resume** makes the specified layer the current (foreground) layer. If no layer is specified, the last foreground layer is implied. If multiple layers are specified, only the first one will be affected.

noresume [*layers*]

This is an *option* to either **create** or **set**, or a command that affects layers as they respawn (see **respawn** below). This option ensures that the layer is specified, ready for creation, but the fork/exec is deferred until the layer is explicitly resumed; the option is then also reset.

toggle [*layer*]

This command is one of the layer-switch commands. **Toggle** resumes the layer specified, or the second most recent foreground layer. If multiple layers are specified, only the first one will be affected.

next [*layer*]
This command is one of the layer-switch commands. **Next** resumes the layer specified, or the one that follows the most recent foreground layer, in round-robin order. If multiple layers are specified, only the first one will be affected.

switch [**resume**|**toggle**|**next**]
This defines the default layer-switch command. The default layer-switch command is invoked whenever the *swtch* character is used to terminate a command line to the manager and no other command is present on that input line (note that a layer can be specified). The **switch** command is implied whenever one of the layer-switch commands is terminated by the *swtch* character, i.e., the default is redefined to the specified layer-switch command, which is then executed.

alarm [*layers*]
Enable alarm for the named layers. When such a layer is in the background and has output pending, the layer can preempt a non-alarm enabled foreground layer. If the alarm occurs while no layer is in the foreground (i.e., while in the user interface), an alarm message is issued. If no name is specified, the last foreground layer referenced is implied.

noalarm [*layers*]
Disable alarm for the named layers.

block [*layers*]
Block the output of the named layers when they are in the background. Such a layer will block when its pending output reaches a highwatermark. If no name is specified, the last foreground layer referenced is implied.

noblock[=[**no**]**spool**] [*layers*]
Don't block the output of any of the specified layers when in the background. Such a layer's output is either spooled to a temporary file, until the layer is again the current foreground layer, or its screen is immediately updated in the background. In either case, the layer will not block due to pending screen output when it is in the background. If no name is specified, the last foreground layer referenced or created is implied.

delete [*layers*]
Delete the named layers. If no name is specified, the last foreground layer referenced is implied. All processes in the process group of the layer are sent the SIGHUP signal (see *signal*(2)). Note that as an option, this indicates that a layer should automatically be deleted when it dies; its last screen of output is discarded. If **nodelete** is specified as an option, the layer is not automatically deleted when it dies; its last screen of output is retained to permit later examination, and an explicit delete command is required to remove the layer.

help (or ?) [commands]
Print the syntax of the layer manager's commands. If a command is specified, detailed information is provided.

[**no**]**lock**
The lock command "locks" the name of the program that can be created in new layers to the shell *(pshl)* or the login program *(conman)*. Additionally, the *root password is required* for the **quit** and **nolock** commands, once **lock** is issued from *conman*. The **nolock** command turns off these restrictions. Note that the login program requires user privilege.

log[=*filename*] [*layers*]
Copy the layers' output to a log file; the log's filename, if not specified, is defaulted. If a default name has been established by the set command, it is used; otherwise, one is created from the name of the layer and the layer's number as: *log_name#*, e.g., log_ksh4. If desired, the name of the log file can be specified by the following syntax: **log**=*filename*. Note that when in a particular layer, it is not desirable to 'tail' or 'cat' the layer's logfile.

nolog [*layers*]
If a log file is associated with the layers, break the association. Note that if it is desired *to change the log file*, **nolog** *should precede the next* **log** *command to bind the new log file.*

pri[=*priority*]
This is an *option* only. Give the layer the specified scheduling priority, or default value. When it is desired to specify the layer's priority, this must be provided on the create command, or established as a creation default on the set command prior to the create. *An existing layer's priority cannot be altered by this command.* Note that setting priority requires privilege. The priority to be given can be specified by the following syntax: **pri**=*priority*

setpri=*priority*
Establish the layer manager's own priority. The priority value must be specified. Setting priority requires privilege.

refresh [*layers*]
Give each of the specified layers automatic screen refresh whenever it is resumed. If no layers are specified, the last foreground layer is implied. Note that the layer's TERM variable should always be set to *vt100* when the refresh option is used. This is set *automatically* when the layer is *created* with **refresh** as an option.

norefresh [*layers*]
Disable screen refresh for the specified layers. If no layers are specified, the last foreground layer referenced is implied.

status [*layers*]
List layer name and its status. If one or more layers are specified, then only the status of these layers is reported; otherwise, the status of all defined layers is reported. If an alarm enabled layer has pending output, the display of this layer is highlighted if the physical device supports this display attribute.

[**no**]**respawn** [*layers*]
If set, the layers are recreated with their current options when they die. This permits layers that should always be available to reappear if desired.

name=string [*layer*]
Give the provided name to the layer specified, or to the most recent foreground layer. The name can be used on all commands which can take such an argument. This permits multiple shell layers, for instance, to each have a unique name. This is also a creation option.

quit Exit the layer manager. All layers are sent the SIGHUP signal. If the **lock** command has been issued from *conman*, then the root password is required.

FILES

/dev/pty[a-z][a-z][0-9a-f]	
/dev/pty[a-z][0-9a-f]	Virtual tty devices
$SHELL	Variable containing path name of the shell to use (default is **/bin/ksh**).

SEE ALSO

ksh(1), stty(1), shl(1), ioctl(2), signal(2), pty(7).

NAME

conv – common object file converter

SYNOPSIS

conv [–a] [–o] [–p] –t *target* [– | *files*]

DESCRIPTION

The *conv* command converts object files in the common object file format from their current byte ordering to the byte ordering of the *target* machine. The converted file is written to **file.v**. The *conv* command can be used on either the source (sending) or target (receiving) machine.

Command line options are:

–
indicates that the names of *files* should be read from the standard input.

–a
If the input file is an archive, produce the output file in the UNIX System V Release 2.0 portable archive format.

–o
If the input file is an archive, produce the output file in the old (pre-UNIX System V) archive format.

–p
If the input file is an archive, produce the output file in the UNIX System V Release 1.0 random access archive format.

–t *target*
Convert the object file to the byte ordering of the machine (*target*) to which the object file is being shipped. This may be another host or a target machine. Legal values for *target* are: pdp, vax, ibm, x86, b16, n3b, mc68 and m32.

The *conv* command is meant to ease the problems created by a multi-host cross-compilation development environment. The *conv* command is best used within a procedure for shipping object files from one machine to another.

The *conv* command will recognize and produce archive files in three formats: the pre-UNIX System V format, the UNIX System V Release 1.0 random access format, and the UNIX System V Release 2.0 portable ASCII format. By default, *conv* will create the output archive file in the same format as the input file. To produce an output file in a different format than the input file, use the **–a**, **–o**, or **–p** option. If the output archive format is the same as the input format, the archive symbol table will be converted, otherwise the symbol table will be stripped from the archive. The *ar*(1) command with its **–t** and **–s** options must be used on the target machine to recreate the archive symbol table.

DIAGNOSTICS

The diagnostics are self-explanatory. Fatal diagnostics on the command lines cause termination. Fatal diagnostics on an input file cause the program to continue to the next input file.

CAVEATS

The *conv* command will not convert archives from one format to another if both the source and target machines have the same byte ordering. You should use *convert*(1) for this purpose.

SEE ALSO

ar(1), convert(1), ar(4), a.out(4).

NAME

convert – convert archive files to common formats

SYNOPSIS

convert *infile outfile*

DESCRIPTION

The *convert* command transforms input *infile* to output *outfile*. *infile* must be a UNIX System V Release 1.0 archive file and *outfile* will be the equivalent UNIX System V Release 2.0 archive file. All other types of input to the *convert* command will be passed unmodified from the input file to the output file (along with appropriate warning messages).

infile must be different from *outfile*.

FILES

TMPDIR/conv* temporary files

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

SEE ALSO

ar(1), tmpnam(3S), a.out(4), ar(4)

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 permissible; if not, the command exits. For *mv*, when the parent directory of **file1** is writable 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 writable 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.

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 that 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, nor group. The last modification time of *target* (and last access time, if *target* did not exist) and the last access time of *file1* are set to the time the copy was made. If *target* is a link to a file, all links remain and the file is changed.
There are two kinds of links: hard links and symbolic links. By default, *ln* makes hard links. A hard link to a file is indistinguishable from the original directory entry; any changes to a file are effective independent of the name used to reference the file. Hard links may not span file systems and may not refer to directories.

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

SEE ALSO

chmod(1), cpio(1), lstat(2), readlink(2), rm(1), symlink(2).

WARNINGS

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

BUGS

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

NAME

cp – copy

SYNOPSIS

cp [**–ip**] file1 file2

cp [**–ipr**] file ... directory

DESCRIPTION

File1 is copied onto *file2*. By default, the mode and owner of *file2* are preserved if it already existed; otherwise the mode of the source file modified by the current *umask*(2) is used. The **–p** option causes *cp* to attempt to preserve (duplicate) in its copies the modification times and modes of the source files, ignoring the present *umask*.

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

Cp refuses to copy a file onto itself.

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

By default, directories will be copied as plain files. If the **–r** option is specified and any of the source files are directories, *cp* copies each subtree rooted at that name; in this case the destination must be a directory.

SEE ALSO

cat(1), cp(1), mv(1B), rcp(1).

NAME

cpio – copy file archives in and out

SYNOPSIS

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

cpio –i[BcdmrtuvVfsSb6k] [—C *bufsize*] [[—**I** *file*] [—**M** *message]] [patterns*]

cpio –p[adlLmuvV] *directory*

DESCRIPTION

cpio –o (copy out) reads the standard input to obtain a list of path names and copies those files onto the standard output together with path name and status information. Output is padded to a 512-byte boundary 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 name-generating notation of *sh*(1). In *patterns*, meta-characters **?**, *****, and [...] match the slash / 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* should be surrounded by double quotes, otherwise the name of a file in the current directory is used. The extracted files are conditionally created and copied into the current directory tree based upon the options described below.

The permissions of the files will be those of the previous *cpio –o*. The owner and group of the files will be that of the current user unless the user is superuser, which causes *cpio* to retain the owner and group of the files of the previous *cpio –o*. Note that 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 overwrite the existing file unconditionally.)

cpio –p (pass) reads the standard input to obtain a list of path names of files that are conditionally created and copied into the destination *directory* tree based upon the options described below.

The meanings of the available options are:

a Reset *access* times of input files after they have been copied. Access times are not reset for linked files when *cpio -pla* is specified.

b Reverses 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 (does not apply to the *pass* option; meaningful only with data directed to or from a character special device, e.g., **/dev/TAPE.CART**.

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 and **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, e.g., **/dev/TAPE.CART**.

d *Directories* are to be created as needed.

f Copy in all *files* except those in *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 Recover from an "out of sync" error. This option searches for a valid header which follows the bad spot. Note that file(s) may be lost on recovery.

l Whenever possible, *link* files rather than copying them. Usable only with the **–p** option.

L Follow symbolic links; the default is not the follow links. If an archive is made from a tree containing symbolic links, it will record the path associated with each link. When it is restored, the symbolic links will be remade. If **-L** is specified, the actual file pointed to by the link is archived instead of the symbolic link contents.

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 enter the device name to continue to the next medium. Use only with the **–o** option.

P Pad files on output when read error occurs. When a read error occurs on a file write zeros instead of skipping the write. This prevents an "out of sync" error. Note that the file contents is only as good as the successful reads.

r Interactively *rename* files. If the user types a null line, the file is skipped. (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). Also forces the target file to be unlinked (removed) before the copy is brought in, such that links to the target are not updated (unless they appear later in the archive and are selected for retrieval).

v *Verbose:* causes a list of file names to be printed. When used with the **t** option, the table of contents looks like the output of an *ls –l* command (see *ls*(1)).

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

6 Process an old (i.e., *Sixth* Edition format) file. Useful only with -i (copy in).

NOTE: *cpio* assumes four-byte words.

If *cpio* reaches the end-of-medium when writing to (-o) or reading from (-i) a character special device, *cpio* will print the message:

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

To continue, you must replace the medium and type the character special device name (**/dev/r02s0** for example) and carriage return. You may want to continue by directing *cpio* to use a different device.

EXAMPLES

The following examples show some uses of *cpio*.

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

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

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

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

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

find . -depth -print | cpio -pdlmv *newdir*

To write multiple archive files to one tape it is necessary to write to a "no-rewind" tape device and use the shell redirection append characters.

```
echo file1a file1b | cpio -o > /dev/r41tn
echo file2a file2b | cpio -o >> /dev/r41tn
echo file3a file3b | cpio -o >> /dev/r41t
```

The above example writes three archive files to tape (each containing 2 files).

To read multiple archive files from one tape it is also necessary to use a "no-rewind" tape device.

```
cpio -i < /dev/r41tn
cpio -i < /dev/r41tn
cpio -i < /dev/r41t
```

SEE ALSO

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

NOTES

1) Path names 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 will not be saved or restored.
5) When restoring files to a non REAL/IX system, ignore errors relating to the file named "/tmp/__PREALLOC__."

NAME

cpp - the C language preprocessor

SYNOPSIS

LIBDIR/*cpp* [*option* ...] [*ifile* [*ofile*]]

DESCRIPTION

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

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

The following *options* to *cpp* are recognized:

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

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

-U*name*
Remove any initial definition of *name*, where *name* is a reserved symbol that is predefined by the particular preprocessor. Following is the current list of these possibly reserved symbols. On the REAL/IX system, *m88k, m88000, sysV88, realix, BCS, _POSIX* and *unix* are defined.

-D*name*

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

-T
The **-T** option forces *cpp* to use only the first eight characters to distinguish preprocessor symbols and is included for backward compatibility.

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

-Y*dir*
Use directory *dir* in place of the standard list of directories when searching for **#include** files.

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

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

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

#define *name token-string*
Replace subsequent instances of *name* with *token-string*.

#define *name(arg, ..., arg) token-string*
Notice that there can be no space between *name* and the (. Replace subsequent instances of *name* followed by a (, a list of comma-separated sets of tokens, and a) followed by *token-string*, where each occurrence of an *arg* in the *token-string* is replaced by the corresponding set of tokens in the comma-separated list. When a macro with arguments is expanded, the arguments are placed into the expanded *token-string* unchanged. After the entire *token-string* has been expanded, *cpp* re-starts its scan for names to expand at the beginning of the newly created *token-string*.

#undef *name*
Cause the definition of *name* (if any) to be forgotten from now on. No additional tokens are permitted on the directive line after *name*.

#ident *"string"*
Put *string* into the .comment section of an object file.

#include *"filename"*

#include <*filename*>
Include at this point the contents of *filename* (which will then be run through *cpp*). When the <*filename*> notation is used, *filename* is only searched for in the standard places. See the **-I** and **-Y** options above for more detail. No additional tokens are permitted on the directive line after the final " or >.

#line *integer-constant "filename"*
Causes *cpp* to generate line control information for the next pass of the C compiler. *Integer-constant* is the line number of the next line and *filename* is the file from which it comes. If *"filename"* is not given, the current file name is unchanged. No additional tokens are permitted on the directive line after the optional *filename*.

#endif
Ends a section of lines begun by a test directive (**#if**, **#ifdef**, or **#ifndef**). Each test directive must have a matching **#endif**. No additional tokens are permitted on the directive line.

#ifdef *name*
The lines following will appear in the output if and only if *name* has been the subject of a previous **#define** without being the subject of an intervening **#undef**. No additional tokens are permitted on the directive line after *name*.

#ifndef *name*
The lines following will appear in the output if and only if *name* has not been the subject of a previous **#define**. No additional tokens are permitted on the directive line after *name*.

#if *constant-expression*
Lines following will appear in the output if and only if the *constant-expression* evaluates to non-zero. All binary non-assignment C operators, the ?: operator, the unary -, !, and ~ operators are all legal in *constant-expression*. The precedence of the operators is the same as defined by the C language. There is also a unary operator **defined**, which can be used in *constant-expression* in these two forms: **defined** (*name*) or **defined** *name*. This allows the

utility of **#ifdef** and **#ifndef** in a **#if** directive. Only these operators, integer constants, and names which are known by *cpp* should be used in *constant-expression*. In particular, the **sizeof** operator is not available.

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

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

#elif *constant-expression*

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

#else

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

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

FILES

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

SEE ALSO

cc(1), lint(1), m4(1).

DIAGNOSTICS

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

NOTES

The unsupported **−W** option enables the **#class** directive. If it encounters a **#class** directive, *cpp* will exit with code 27 after finishing all other processing. This option provides support for "C with classes".

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

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

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

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

cpp warns about the use of the absolute pathname.

NAME

cprs – compress a common object file

SYNOPSIS

cprs [–p] *file1 file2*

DESCRIPTION

The *cprs* command reduces the size of a common object file, **file1**, by removing duplicate **structure** and union descriptors. The reduced file, **file2**, is produced as output.

The sole option to *cprs* is:

–p
Print statistical messages including: total number of tags, total duplicate tags, and total reduction of **file1**.

SEE ALSO

strip(1), a.out(4), syms(4).

NAME

cputype – print CPU board information

SYNOPSIS

/bin/cputype [**–ps**] [**–n** namelist]

DESCRIPTION

cputype prints information about the CPU the system is running. Without options, it prints the CPU board name and defaults to **/realix** as the currently running system.

OPTIONS

–p print the processor name instead of the board name.

–s print CPU speed (in MHZ).

–n *namelist*
Take *namelist* in place of **/realix** as the currently running system.

FILES

/dev/mem	system image of currently running system.
/realix	namelist of currently running system.

NAME

crc - generate cyclic redundancy checksums (crc) of files

SYNOPSIS

crc [-frcld] - | *file_list*

DESCRIPTION

The *crc* shell command utility is a versatile tool for use in generating 16-bit *crc* values of an input stream. The input stream can consist either of data or of names of files to be checked. There are four different display options available.

If the file to be checked is an object file, *crc* will ignore the compiler-generated time stamps embedded in the file.

The various options are defined as:

-f

Selects *file mode* operation. The input stream is interpreted as a list of the names of the files to be processed instead of the data itself.

-r

Selects a *raw mode* of operation. This option is used mainly to determine if two versions of an executable file are *exactly* the same. This switch causes *crc* to include the compiler-generated time stamps in the *coff* file image when computing the *crc*.

-c

Changes the output to include the byte count of each file processed.

-d

Adds the time of the file's modification to the output.

-l

Computes the *crc* in decimal for *each line of the input file* instead of the whole file itself. *Use of this option overrides all others.*

Note that the first four options can be used in any combination. There are three general forms of output. The first form is produced without the **-c** option:

$nnnn for *filename (time stamp)*

where *$nnnn* is the 16-bit checksum in hexadecimal representation; and *time stamp* is the time of the file's modification (displayed if the **-d** option is selected). The fields are separated by space (20h) characters.

The second output form is generated when the **-c** option is selected:

$nnnn length time stamp filename

where *length* is the true size of the file, regardless of whether *raw mode* (**-r**) is selected; and *time stamp* is the time of the file's modification (displayed if the **-d** option is selected). All fields of this second form are delimited by tab (\t) characters.

The third form of output is produced by the *line mode* option (**-l**). It replaces each line of input with its corresponding *crc* in the form *nnnn*.

DIAGNOSTICS

crc: bad option letter. an invalid option letter was specified.

crc: argument count. at least one file name (or '-') must be provided.

crc: can't open file for reading. file cannot be opened for some reason.

crc: can't read file. input file cannot be read for some reason.

EXAMPLES

Suppose a **touch *; ls -log** command produces the following directory listing:

```
-rwxrwxrwx 1      23 Apr  8 12:39   apple
-rwxrwxrwx 1    8307 Apr  8 12:39   peaches
-rwxrwxrwx 1    1280 Apr  8 12:39   pears
-rwxrwxrwx 1     771 Apr  8 12:39   plums
```

Note that **ls | crc -fdc -** *is equivalent to* **crc -dc ***, and both would produce output similar to:

```
$8AC3      23    Apr  8 12:39:51 1986   apple
$FD06    8307    Apr  8 12:39:51 1986   peaches
$C3B0    1280    Apr  8 12:39:51 1986   pears
$02D2     771    Apr  8 12:39:51 1986   plums
```

A means of generating checksums for an entire directory hierarchy is:

find *root_path* **-type f -print | crc -fcd -**

Use of the '-type f' option on *find* is recommended because *crc* will generate the *crc* for the directory files themselves if presented with their names.
You can extract just the *crc* and length from a stream of *crc's* by using the *cut* command. When appended to the above command,

find *args* **-type f -print | crc -fcd - | cut -f1,2**

produces an output of two columns: the *crc* and the file's length.

FILES

/usr/bin/crc

NAME

crontab – user crontab file

SYNOPSIS

crontab *[file]*
crontab **–r**
crontab **–l**

DESCRIPTION

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

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

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

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

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

sh(1). cron(1M).

WARNINGS

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

NAME

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 –, 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 ... *prefixn*. 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 (e.g., **/Page/–5**).

%rexp%
This argument is the same as */rexp/*, except that no file is created for the section.

lnno
A file is to be created from the current line up to (but not including) *lnno*. The current line becomes *lnno*.

{*num*}
Repeat argument. This argument may follow any of the above 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 new-lines. *csplit* does not affect the original file; it is the users responsibility to remove it.

EXAMPLES

csplit –f cobol file '/procedure division/' /par5./ /par16./

This example creates four files, **cobol00 ... cobol03**. After editing the "split" files, they can be recombined as follows:

cat cobol0[0–3] > file

Note that this example overwrites the original file.

csplit –k file 100 {99}

This example would split the file at every 100 lines, up to 10,000 lines. The **–k** option causes the created files to be retained if there are less than 10,000 lines; however, an error message would still be printed.

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

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

SEE ALSO

ed(1), sh(1), split(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 [**–w***n*] [**–x***n*] [**–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. The *telno* argument 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 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, for how many 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; **–x9** is the most useful value.

Normally, *ct* will hang up the current line so the line can answer the incoming call. The **–h** option prevents this action. The **–h** option also waits for the termination of the specified *ct* process before returning control to the user's terminal. If the **–v** option is used, *ct* sends a running narrative to the standard error output stream.

The data rate may be set with the **–s** option, where *speed* is expressed in baud. The default rate is 1200.

After the user on the destination terminal logs out, there are two things that could occur depending on what type of *getty* is on the line (*getty* or *uugetty*). For the first case, *ct* prompts **Reconnect?** . If the response begins with the letter **n**, the line will be dropped; otherwise, *getty* will be started again and the **login:** prompt will be printed. In the second case, there is already a *getty* (*uugetty*) on the line, so the **login:** message will appear.

To log out properly, the user must type **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
/usr/spool/locks/LCK*

SEE ALSO

login(1), cu(1C), 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 – maintain a tags file for a C program

SYNOPSIS

ctags [**–a**] [**–u**] [**–w**] [**–x**] *name* ...

DESCRIPTION

ctags makes a tags file for *ex* (1) and *vi* (1) from the specified 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 function name, the file in which it is defined, and a scanning pattern used to find the function definition. These are given in separate fields on the line, separated by blanks or tabs. Using the tags file, *ex* can quickly find these function definitions.

Options

–a
appends the output to the tags file instead of rewriting it.

–u
causes the specified files to be *updated* in tags, that is, all references to them are replaced by new values. (Beware: this option is implemented in a way which is rather slow; it is usually faster to simply rebuild the *tags* file.)

–w
suppresses warning diagnostics.

–x
produces a list of function names, the line number and file name on which each is defined, as well as the text of that line and prints this on the standard output.

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

The tag *main* is treated specially in C programs. The tag formed is created by prefixing **M** to the name of the file, with a trailing **.c** removed, if any, and leading pathname components also removed. This makes use of *ctags* practical in directories with more than one program.

EXAMPLE

```
ctags *.c *.h
```

puts the tags from all the **.c** and **.h** files into the tags file **tags**.

FILES

/usr/bin/ctags	
tags	output tags file

SEE ALSO

ex(1), vi(1).

BUGS

Not all warning diagnostics are suppressed by **–w**.

If *ctags* is interrupted while executing under the **–u** option, a temporary file named **OTAGS** is left in the current directory.

NAME

ctrace – C program debugger

SYNOPSIS

ctrace *[options] [file]*

DESCRIPTION

The *ctrace* command allows you to follow the execution of a C program, statement-by-statement. The effect is similar to executing a shell procedure with the **–x** option. *ctrace* reads the C program in **file** (or from standard input if you do not specify **file**), inserts statements to print the text of each executable statement and the values of all variables referenced or modified, and writes the modified program to the standard output. You must put the output of *ctrace* into a temporary file because the *cc*(1) command does not allow the use of a pipe. You then compile and execute this file.

As each statement in the program executes it will be listed at the terminal, followed by the name and value of any variables referenced or modified in the statement, followed by any output from the statement. Loops in the trace output are detected and tracing is stopped until the loop is exited or a different sequence of statements within the loop is executed. A warning message is printed every 1000 times through the loop to help you detect infinite loops. The trace output goes to the standard output so you can put it into a file for examination with an editor or the *bfs*(1) or *tail*(1) commands.

The options commonly used are:

–f ***functions***
Trace only these *functions.*

–v ***functions***
Trace all but these *functions.*

You may want to add to the default formats for printing variables. Long and pointer variables are always printed as signed integers. Pointers to character arrays are also printed as strings if appropriate. Char, short, and int variables are also printed as signed integers and, if appropriate, as characters. Double variables are printed as floating point numbers in scientific notation. You can request that variables be printed in additional formats if appropriate, with the following options:

–o
Octal

–x
Hexadecimal

–u
Unsigned

–e
Floating point

The following options are used only in special circumstances:

–l ***n***
Check *n* consecutively executed statements for looping trace output, instead of the default of 20. Use 0 to get all the trace output from loops.

–s
Suppress redundant trace output from simple assignment statements and string copy function calls. This option can hide a bug caused by use of the = operator in place of the == operator.

–t *n*
Trace *n* variables per statement instead of the default of 10 (the maximum number is 20). The **DIAGNOSTICS** section explains when to use this option.

–P
Run the C preprocessor on the input before tracing it. You can also use the **–D**, **–I**, and **–U** *cpp*(1) options.

These options are used to tailor the run-time trace package when the traced program will run in a non-UNIX System environment:

–b
Use only basic functions in the trace code, i.e., those in *ctype*(3C), *printf*(3S), and *string*(3C). These are usually available even in cross-compilers for microprocessors. In particular, this option is needed when the traced program runs under an operating system that does not have *signal*(2), *fflush*(3S), *longjmp*(3C), or *setjmp*(3C).

–p *string*
Change the trace print function from the default of 'printf('. For example, 'fprintf(stderr,' would send the trace to the standard error output.

–r *f*
Use file *f* in place of the *runtime.c* trace function package. This lets you change the entire print function, instead of just the name and leading arguments (see the **–p** option).

EXAMPLE

If the file *lc.c* contains this C program:

```
1 #include <stdio.h>
2 main()         /* count lines in input */
3 {
4       int c, nl;
5
6       nl = 0;
7       while ((c = getchar()) != EOF)
8               if (c = '\n')
9                       ++nl;
10      printf("%d\n", nl);
11 }
```

and you enter these commands and test data:

```
cc lc.c
a.out
1
(CTRL-d)
```

the program will be compiled and executed. The output of the program will be the number 2, which is not correct because there is only one line in the test data. The error in this program is common, but subtle. If you invoke *ctrace* with these commands:

```
ctrace lc.c >temp.c
cc temp.c
a.out
```

the output will be:

```
2 main()
6       nl = 0;
        /* nl == 0 */
7       while ((c = getchar()) != EOF)
```

The program is now waiting for input. If you enter the same test data as before, the output will be:

```
        /* c == 49 or '1' */
8               if (c = '\n')
                /* c == 10 or '\n' */
9                       ++nl;
                        /* nl == 1 */
7       while ((c = getchar()) != EOF)
        /* c == 10 or '\n' */
8               if (c = '\n')
                /* c == 10 or '\n' */
9                       ++nl;
                        /* nl == 2 */
7       while ((c = getchar()) != EOF)
```

If you now enter an end of file character (**CTRL-d**), the final output will be:

```
        /* c == -1 */
10      printf("%d\n", nl);
        /* nl == 2 */2
         return
```

Note that the program output printed at the end of the trace line for the **nl** variable. Also note the **return** comment added by *ctrace* at the end of the trace output. This shows the implicit return at the terminating brace in the function.

The trace output shows that variable c is assigned the value '1' in line 7, but in line 8 it has the value '\n'. Once your attention is drawn to this **if** statement, you will probably realize that you used the assignment operator (=) in place of the equality operator (==). You can easily miss this error during code reading.

EXECUTION-TIME TRACE CONTROL

The default operation for *ctrace* is to trace the entire program file, unless you use the **–f** or **–v** options to trace specific functions. This does not give you statement-by-statement control of the tracing, nor does it let you turn the tracing off and on when executing the traced program.

You can do both of these by adding *ctroff*() and *ctron*() function calls to your program to turn the tracing off and on, respectively, at execution time. Thus, you can code arbitrarily complex criteria for trace control with *if* statements, and you can even conditionally include this code because *ctrace* defines the CTRACE preprocessor variable. For example:

```
#ifdef CTRACE
        if (c == '!' && i > 1000)
                ctron();
#endif
```

You can also call these functions from *sdb*(1) if you compile with the **–g** option. For example, to trace all but lines 7 to 10 in the main function, enter:

```
sdb a.out
main:7b ctroff()
main:11b ctron()
r
```

You can also turn the trace off and on by setting static variable tr_ct_ to 0 and 1, respectively. This is useful if you are using a debugger that cannot call these functions directly.

DIAGNOSTICS

This section contains diagnostic messages from both *ctrace* and *cc*(1), since the traced code often gets some *cc* warning messages. You can get *cc* error messages in some rare cases, all of which can be avoided.

ctrace Diagnostics

warning: some variables are not traced in this statement
Only 10 variables are traced in a statement to prevent the C compiler "out of tree space; simplify expression" error. Use the **–t** option to increase this number.

warning: statement too long to trace
This statement is over 400 characters long. Make sure that you are using tabs to indent your code, not spaces.

cannot handle preprocessor code, use –P option
This is usually caused by **#ifdef/#endif** preprocessor statements in the middle of a C statement, or by a semicolon at the end of a **#define** preprocessor statement.

'if ... else if' sequence too long
Split the sequence by removing an **else** from the middle.

possible syntax error, try –P option
Use the **–P** option to preprocess the *ctrace* input, along with any appropriate **–D**, **–I**, and **–U** preprocessor options. If you still get the error message, check the WARNINGS section.

Cc Diagnostics

warning: illegal combination of pointer and integer

warning: statement not reached

warning: sizeof returns 0
Ignore these messages.

compiler takes size of function
See the *ctrace* "possible syntax error" message above.

yacc stack overflow
See the *ctrace* "'if ... else if' sequence too long" message above.

out of tree space; simplify expression
Use the **–t** option to reduce the number of traced variables per statement from the default of 10. Ignore the **ctrace: too many variables to trace** warnings you will now get.

redeclaration of signal
Either correct this declaration of *signal*(2), or remove it and **#include <signal.h>**.

SEE ALSO

bfs(1), tail(1), signal(2), ctype(3C), fclose(3S), printf(3S), setjmp(3C), string(3C).

WARNINGS

You will get a *ctrace* syntax error if you omit the semicolon at the end of the last element declaration in a structure or union, just before the right brace (}). This is optional in some C compilers.

Defining a function with the same name as a system function may cause a syntax error if the number of arguments is changed. Just use a different name.

ctrace assumes that BADMAG is a preprocessor macro, and that EOF and **NULL** are #defined constants. Declaring any of these to be variables, e.g., **int** EOF;, will cause a syntax error.

BUGS

ctrace does not know about the components of aggregates like structures, unions, and arrays. It cannot choose a format to print all the components of an aggregate when an assignment is made to the entire aggregate. *ctrace* may choose to print the address of an aggregate or use the wrong format (e.g., 3.149050e-311 for a structure with two integer members) when printing the value of an aggregate.

Pointer values are always treated as pointers to character strings.

The loop trace output elimination is done separately for each file of a multi-file program. This can result in functions called from a loop still being traced, or the elimination of trace output from one function in a file until another in the same file is called.

FILES

/usr/lib/ctrace/runtime.c run-time trace package

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:

–s*speed*
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.

–l*line*
specifies a device name to use as the communication line. This can be used to override the search that would otherwise take place for the first available line 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 both used together, *cu* will search the **Devices** file to check if the requested speed for the requested line is available. If so, the connection is made at the requested speed; otherwise, an error message is printed and the call will not be made. The specified device is generally a directly connected asynchronous line (e.g., **/dev/tty***ab*) 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 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 generated for data sent to the remote system.

telno
When using an automatic dialer, the argument is the telephone number with equal signs for secondary dial tone or minus signs placed appropriately for delays of 4 seconds.

systemname
A uucp system name may be used rather than a telephone number; in this case, *cu* will obtain an appropriate direct line or telephone number from **/usr/lib/uucp/Systems.** Note that the *systemname* option should not be used with the **–l** and **–s** options because *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 that **~!cd** will cause the command to be run by a sub-shell, probably not what was intended.

~%take *src* [*dest*]
copy file *src* (on the remote system) to file *dest* on the local system. If *dest* is omitted, the *src* argument is used in both places.

~%put *src* [*dest*]
copy file *src* (on local system) to file *dest* on remote system. If *dest* is omitted, the *src* argument is used in both places.

For both **~%take** and **put** commands, as each block of the file is transferred, consecutive single digits are printed to the terminal.

~~ *line*
send the line ~ *line* to the remote system.

~%break
transmit a **BREAK** to the remote system (which can also be specified as **~%b**).

~%debug
toggles the **-d** debugging option on or off (which can also be specified as **~%d**).

~t
prints the values of the termio structure variables for the user's terminal (useful for debugging).

~l
prints the values of the termio structure variables for the remote communication line (useful for debugging).

~%nostop
toggles between DC3/DC1 input control protocol and no input control. This is useful in case the remote system is one that 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 the 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 being expanded to spaces.

When *cu* is used on system X to connect to system Y and subsequently used on system Y to connect to system Z, commands on system Y can be executed by using ~~. Executing a *tilde* command reminds the user of the local system uname. For example, uname can be executed on Z, X, and Y as follows:

```
uname
Z
~[X]!uname
X
~~[Y]!uname
Y
```

In general, ~ causes the command to be executed on the original machine, ~~ causes the command to be executed on the next machine in the chain.

EXAMPLES

To dial a system whose telephone number is 9 201 555 1212 using 1200 baud (where dialtone is expected after the 9):

cu -s1200 9=12015551212

If the speed is not specified, "Any" is the default value.

To login to a system connected by a direct line:

cu -l /dev/ttyXX

or

cu -l ttyXX

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

cu -s1200 -l ttyXX

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

cu -l culXX 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 data base parlance, it implements the projection of a relation. The fields as specified by *list* can be fixed length, i.e., character positions as on a punched card (–c option) or the length can vary from line to line and be marked with a field delimiter character like *tab* (–**f** option). *cut* can be used as a filter; if no files are given, the standard input is used. In addition, a file name of "–" explicitly refers to standard input.

The meanings of the options are:

list A comma-separated list of integer field numbers (in increasing order), with optional – to indicate ranges (e.g., **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 (e.g., –**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**); e.g., –**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 (i.e., 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" "`
to set **name** to current login name.

DIAGNOSTICS

ERROR: line too long
A line can have no more than 1023 characters or fields, or there is no newline 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 does not exist. If multiple file names are present, processing continues.

SEE ALSO
grep(1), paste(1).

NAME

cxref – generate C program cross-reference

SYNOPSIS

cxref *[options] files*

DESCRIPTION

The *cxref* command analyzes a collection of C files and attempts to build a cross-reference table. *cxref* uses a special version of *cpp* to include **#defined** information in its symbol table. It produces a listing on standard output of all symbols (auto, static, and global) in each file separately, or, with the –c option, in combination. Each symbol contains an asterisk (*) before the declaring reference.

In addition to the **–D**, **–I** and **–U** options (which are interpreted just as they are by *cc*(1) and *cpp*(1)), the following *options* are interpreted by *cxref*:

–c Print a combined cross-reference of all input files.

–w<*num*>
Width option which formats output no wider than <num> (decimal) columns. This option will default to 80 if <num> is not specified or is less than 51.

–o file Direct output to *file*.

–s Operate silently; do not print input file names.

–t Format listing for 80-column width.

FILES

LLIBDIR	usually **/usr/lib**
LLIBDIR/xcpp	special version of the C preprocessor.

SEE ALSO

cc(1), cpp(1).

DIAGNOSTICS

Error messages are unusually cryptic, but usually mean that you cannot compile these files.

BUGS

cxref considers a formal argument in a **#define** macro definition to be a declaration of that symbol. For example, a program that **#includes ctype.h**, will contain many declarations of the variable **c**.

NAME

date – print and set the date

SYNOPSIS

date [mmddhhmm[yy]] | +format]

DESCRIPTION

If no argument is given, or if the argument begins with +, the current date and time are printed. Otherwise, the current date is set. The first *mm* is the month number; *dd* is the day number in the month; *hh* is the hour number (24 hour system); the second *mm* is the minute number; *yy* is the last 2 digits of the year number and is optional. For example:

date 10080045

sets the date to Oct 8, 12:45 AM. The current year is the default if no year is mentioned. The system operates in GMT. *date* takes care of the conversion to and from local standard and daylight time. Only the superuser may change the date.

If the argument begins with +, the output of *date* is under the control of the user. All output fields are of fixed size (zero padded if necessary). Each field descriptor is preceded by % and will be replaced in the output by its corresponding value. A single % is encoded by %%. All other characters are copied to the output without change. The string is always terminated with a newline character.

Field Descriptors:

n	insert a newline character
t	insert a tab character
m	month of year – 01 to 12
d	day of month – 01 to 31
y	last 2 digits of year – 00 to 99
D	date as mm/dd/yy
H	hour – 00 to 23
M	minute – 00 to 59
S	second – 00 to 59
T	time as HH:MM:SS
j	day of year – 001 to 366
w	day of week – Sunday = 0
a	abbreviated weekday – Sun to Sat
h	abbreviated month – Jan to Dec
r	time in AM/PM notation

EXAMPLE

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

would have generated output similar to:

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

DIAGNOSTICS

No permission	if you are not the superuser and you try to change the date or if there is a lock (typically set by realtime processes via *sysrealix(2)*) preventing the modification of system time.
bad conversion	if the date set is syntactically incorrect;
bad format character	if the field descriptor is not recognizable.

FILES

/dev/kmem

WARNING

Should you need to change the date while the system is running multiuser, use *sysadm*(1) *datetime*, which will deal with interactions with the *cron*(1) utility.

Should you wish to change the date other than at system setup time, note that file creation times will be based on the new value specified for the time. Caution is advisable when using utilities such as *make*(1), as file creation times may not give the real order in which files were created.

Note also that changing the date and time will change the system-wide realtime timer of type TIMEOFDAY. If this timer is in use by process interval timers (as may be true if realtime applications are running) there could be unwanted side effects. See *absinterval*(2) or *incinterval*(2) for details. It is recommended that any realtime applications be shut down before changes are made to the system date and time. Also, before scheduling interval timers to guard the system date and time from updates via a call to *sysrealix(2)*, with the RLXDATELCK argument.

SEE ALSO

sysadm(1), absinterval(2), incinterval(2), sysrealix(2).

NAME

dbx – debugger

SYNOPSIS

dbx [**–r**] [**–i**] [**–k**] [**–I** *dir*] [**–c** *file*] [**-a** *pid*]
[**-d** *pid*] [**-m** *pgm*] [**-x**] [*objfile* [*coredump*]]

DESCRIPTION

Dbx is a tool for source level debugging and execution of programs under UNIX. The *objfile* is an object file produced by a compiler with the appropriate flag (usually "–g") specified to produce symbol information in the object file. Though originally intended for object files produced by compilers supported by Berkeley Software Distributions, this version is instead targeted for compilers producing COFF (Common Object File Format) as supported on System V. Some differences may be found, and are described at the end of this man page. The machine level facilities of *dbx* can be used on any program.

The object file contains a symbol table that includes the name of the all the source files translated by the compiler to create it. These files are available for perusal while using the debugger.

If a file named "core" exists in the current directory or a *coredump* file is specified, *dbx* can be used to examine the state of the program when it faulted.

If the file ".dbxinit" exists in the current directory then the debugger commands in it are executed. *Dbx* also checks for a ".dbxinit" in the user's home directory if there isn't one in the current directory.

The command line options and their meanings are:

–r Execute *objfile* immediately. If it terminates successfully *dbx* exits. Otherwise the reason for termination will be reported and the user offered the option of entering the debugger or letting the program fault. *Dbx* will read from "/dev/tty" when **–r** is specified and standard input is not a terminal.

–i Force *dbx* to act as though standard input is a terminal.

–k Map memory addresses, useful for kernel debugging. (Unsupported for SYS V: use Crash(1M))

–I *dir* Add *dir* to the list of directories that are searched when looking for a source file. Normally *dbx* looks for source files in the current directory and in the directory where *objfile* is located. The directory search path can also be set with the **use** command.

–c *file* Execute the *dbx* commands in the *file* before reading from standard input.

–a *pid* Attach to the existing process whose id is *pid*. When *dbx* attaches to a process, it may control and interrogate the process just as it may its own child. Coredumps and the **–r** option are ignored. This option is subject to user permissions, operating system features, and process availability (i.e., only user processes are available; kernel daemons or other processes that do not execute user level code are unavailable).

–d *pid* Detach an existing process (specified by the **-a** option) from another *dbx* instantiation whose process id is *pid*; the attached process is attached to the new *dbx* being invoked. Effectively, this transfers the attached process from one debugger to another; it is used by the debugger to give a new child its own debugger.

–m *pgm* Mark the session as a multitask session, giving each child process created its own

debugger. The specified *pgm* provides an independent screen and keyboard stream to the process, permitting multiple tasks that require keyboard input to run concurrently. The only pgm supported in this implementation is *xdbx*. This option must be specified by the user when initiating a multitask debugging session; it is inherited by all child processes.

-x Mark the traced process as expendable, meaning that the attached process should be deleted when the debugger exits. When the first task of a multitask session is spawned from *dbx*, all the task's children are automatically marked as expendable so they will disappear when the debugger tracing the first task exits. This option requires both **-a** and **-d** options; it is inherited.

Unless **-r** is specified, *dbx* just prompts and waits for a command.

Execution and Tracing Commands

run [*args*] [< *filename*] [> *filename*]
rerun [*args*] [< *filename*] [> *filename*]
Start executing *objfile*, passing *args* as command line arguments; < or > can be used to redirect input or output in the usual manner. When **rerun** is used without any arguments the previous argument list is passed to the program; otherwise it is identical to **run**. If *objfile* has been written since the last time the symbolic information was read in, *dbx* will read in the new information.

trace [**in** *procedure/function*] [**if** *condition*]
trace *source-line-number* [**if** *condition*]
trace *procedure/function* [**in** *procedure/function*] [**if** *condition*]
trace *expression* **at** *source-line-number* [**if** *condition*]
trace *variable* [**in** *procedure/function*] [**if** *condition*]
Have tracing information printed when the program is executed. A number is associated with the command that is used to turn the tracing off (see the **delete** command).

The first argument describes what is to be traced. If it is a *source-line-number*, then the line is printed immediately prior to being executed. Source line numbers in a file other than the current one must be preceded by the name of the file in quotes and a colon, e.g. "mumble.p":17.

If the argument is a procedure or function name then every time it is called, information is printed telling what routine called it, from what source line it was called, and what parameters were passed to it. In addition, its return is noted, and if it's a function then the value it is returning is also printed.

If the argument is an *expression* with an **at** clause then the value of the expression is printed whenever the identified source line is reached.

If the argument is a variable then the name and value of the variable is printed whenever it changes. Execution is substantially slower during this form of tracing.

If no argument is specified then all source lines are printed before they are executed. Execution is substantially slower during this form of tracing.

The clause "**in** *procedure/function*" restricts tracing information to be printed only while executing inside the given procedure or function.

Condition is a boolean expression and is evaluated prior to printing the tracing information; if it is false then the information is not printed.

stop if *condition*
stop at *source-line-number* [**if** *condition*]
stop in *procedure/function* [**if** *condition*]
stop *variable* [**if** *condition*]
Stop execution when the given line is reached, procedure or function called, variable changed, or condition true.

status [> *filename*]
Print out the currently active **trace** and **stop** commands.

delete *command-number ...*
The traces or stops corresponding to the given numbers are removed. The numbers associated with traces and stops are printed by the **status** command.

catch *number*
catch *signal-name*
ignore *number*
ignore *signal-name*
Start or stop trapping a signal before it is sent to the program. This is useful when a program being debugged handles signals such as interrupts. A signal may be specified by number or by a name (e.g., SIGINT). Signal names are case insensitive and the "SIG" prefix is optional. By default all signals are trapped except SIGCONT, SIGCHILD, SIGALRM and SIGKILL.

cont *integer*
cont *signal-name*
Continue execution from where it stopped. If a signal is specified, the process continues as though it received the signal. Otherwise, the process is continued as though it had not been stopped.

Execution cannot be continued if the process has "finished", that is, called the standard procedure "exit". *Dbx* does not allow the process to exit, thereby letting the user to examine the program state.

step Execute one source line.

next Execute up to the next source line. The difference between this and **step** is that if the line contains a call to a procedure or function the **step** command will stop at the beginning of that block, while the **next** command will not.

return [*procedure*]
Continue until a return to *procedure* is executed, or until the current procedure returns if none is specified.

call *procedure(parameters)*
Execute the object code associated with the named procedure or function.

Printing Variables and Expressions

Names are resolved first using the static scope of the current function, then using the dynamic scope if the name is not defined in the static scope. If static and dynamic searches do not yield a result, an arbitrary symbol is chosen and the message "[using *qualified name*]" is printed. The name resolution procedure may be overridden by qualifying an identifier with a block name, e.g., "*module.variable*". For C, source files are treated as modules named by the file name without ".c".

Expressions are specified with an approximately common subset of C and Pascal (or equivalently Modula-2) syntax. Indirection can be denoted using either a prefix "*" or a postfix "^" and array expressions are subscripted by brackets ("[ ]"). The field reference operator (".") can be used with pointers as well as records, making the C operator "-> " unnecessary (although it is supported).

Types of expressions are checked; the type of an expression may be overridden by using "*type-name*(*expression*)". When there is no corresponding named type the special constructs "&*type-name*" and "$$*tag-name*" can be used to represent a pointer to a named type or C structure tag.

assign *variable* = *expression*
Assign the value of the expression to the variable.

dump [*procedure*] [> *filename*]
Print the names and values of variables in the given procedure, or the current one if none is specified. If the procedure given is ".", then the all active variables are dumped.

print *expression* [, *expression* ...]
Print out the values of the expressions.

whatis *name*
Print the declaration of the given name, which may be qualified with block names as above.

which *identifier*
Print the full qualification of the given identifer, i.e. the outer blocks that the identifier is associated with.

up [*count*]
down [*count*]
Move the current function, which is used for resolving names, up or down the stack *count* levels. The default *count* is 1.

where Print out a list of the active procedures and function.

whereis *identifier*
Print the full qualification of all the symbols whose name matches the given identifier. The order in which the symbols are printed is not meaningful.

Accessing Source Files

/*regular expression*[/]
?*regular expression*[?]
Search forward or backward in the current source file for the given pattern.

edit [*filename*]
edit *procedure/function-name*
Invoke an editor on *filename* or the current source file if none is specified. If a *procedure* or *function* name is specified, the editor is invoked on the file that contains it. Which editor is invoked by default depends on the installation. The default can be overridden by setting the environment variable EDITOR to the name of the desired editor.

file [*filename*]
Change the current source file name to *filename*. If none is specified then the current source file name is printed.

func [*procedure/function*]
Change the current function. If none is specified then print the current function. Changing the current function implicitly changes the current source file to the one that contains

the function; it also changes the current scope used for name resolution.

list [*source-line-number* [, *source-line-number*]]
list *procedure/function*
List the lines in the current source file from the first line number to the second inclusive. If no lines are specified, the next 10 lines are listed. If the name of a procedure or function is given lines *n-k* to *n+k* are listed where *n* is the first statement in the procedure or function and *k* is small.

use *directory-list*
Set the list of directories to be searched when looking for source files. The format is a space-separated list, e.g,

```
use /dir1 /dir2 /dir3
```

Command Aliases and Variables

alias *name name*
alias *name* "*string*"
alias *name* (*parameters*) "*string*"
When commands are processed, dbx first checks to see if the word is an alias for either a command or a string. If it is an alias, then dbx treats the input as though the corresponding string (with values substituted for any parameters) had been entered. For example, to define an alias "rr" for the command "rerun", one can say

```
alias rr rerun
```

To define an alias called "b" that sets a stop at a particular line one can say

```
alias b(x) "stop at x"
```

Subsequently, the command "b(12)" will expand to "stop at 12".

set *name* [= *expression*]
The set command defines values for debugger variables. The names of these variables cannot conflict with names in the program being debugged, and are expanded to the corresponding expression within other commands. The following variables have a special meaning:

$frame
Setting this variable to an address causes dbx to use the stack frame pointed to by the address for doing stack traces and accessing local variables. This facility is of particular use for kernel debugging. It is unsupported for non-BSD systems that support *crash(1M)*.

$hexchars
$hexints
$hexoffsets
$hexstrings
When set, dbx prints out out characters, integers, offsets from registers, or character pointers respectively in hexadecimal.

$listwindow

The value of this variable specifies the number of lines to list around a function or when the **list** command is given without any parameters. Its default value is 10.

$mapaddrs
Setting (unsetting) this variable causes dbx to start (stop) mapping addresses. As with "$frame", this is useful for kernel debugging. It is unsupported for non-BSD systems that support *crash(1M)*.

$unsafecall
$unsafeassign
When "$unsafecall" is set, strict type checking is turned off for arguments to subroutine or function calls (*e.g.* in the **call** statement). When "$unsafeassign" is set, strict type checking between the two sides of an **assign** statement is turned off. These variables should be used only with great care, because they severely limit dbx's usefulness for detecting errors.

unalias *name*
Remove the alias with the given name.

unset *name*
Delete the debugger variable associated with *name*.

Machine Level Commands

tracei [*address*] [**if** *cond*]
tracei [*variable*] [**at** *address*] [**if** *cond*]
stopi [*address*] [**if** *cond*]
stopi [**at**] [*address*] [**if** *cond*]
Turn on tracing or set a stop using a machine instruction address.

stepi

nexti Single step as in **step** or **next**, but do a single instruction rather than source line.

address ,*address*/ [*mode*]
address / [*count*] [*mode*]
Print the contents of memory starting at the first *address* and continuing up to the second *address* or until *count* items are printed. If the address is ".", the address following the one printed most recently is used. The *mode* specifies how memory is to be printed; if it is omitted the previous mode specified is used. The initial mode is "X". The following modes are supported:

i	print the machine instruction
d	print a short word in decimal
D	print a long word in decimal
o	print a short word in octal
O	print a long word in octal
x	print a short word in hexadecimal
X	print a long word in hexadecimal
b	print a byte in octal
c	print a byte as a character
s	print a string of characters terminated by a null byte
f	print a single precision real number
g	print a double precision real number

Symbolic addresses are specified by preceding the name with an "&". Registers are denoted by "$rN" where N is the number of the register. Addresses may be expressions made up of other addresses and the operators "+", "-", and indirection (unary "*"). In the current implementation, register addresses may not be mixed with these operators.

value = [*count*] [*mode*]
Print the value as specified. If the value is a register symbol, as described above, then the value of the register is printed. The count field may be used to permit any aggregate of the registers to be displayed. An *additional* mode (S) may be used with this syntax to provide a symbol name corresponding to the address value input.

Miscellaneous Commands

gripe Invoke a mail program to send a message to the person in charge of *dbx*. This is not typically available on production versions of *dbx*.

help [*cmds*]
Print out a synopsis of *dbx* commands. If commands are specified, only detailed help for those commands is provided.

quit Exit *dbx*.

sh *command-line*
Pass the command line to the shell for execution. The SHELL environment variable determines which shell is used.

source *filename*
Read *dbx* commands from the given *filename*.

FILES

a.out	object file
.dbxinit	initial commands

SEE ALSO

cc(1), cprs(1)

COMMENTS

Dbx suffers from the same "multiple include" malady as did *sdb*. If you have a program consisting of a number of object files and each is built from source files that include header files, the symbolic information for the header files is replicated in each object file. Since about one debugger start-up is done for each link, having the linker (ld) re-organize the symbol information would not save much time, though it would reduce some of the disk space used.

This problem is an artifact of the unrestricted semantics of #include's in C; for example an include file can contain static declarations that are separate entities for each file in which they are included. However, even with Modula-2 there is a substantial amount of duplication of symbol information necessary for inter-module type checking.

Note that many duplicate debugging symbols can be eliminated in COFF files by *cprs(1)*.

Some problems remain with the support for individual languages. Fortran problems include: inability to assign to logical, logical*2, complex and double complex variables; inability to represent parameter constants which are not type integer or real; peculiar representation for the values of dummy procedures (the value shown for a dummy procedure is actually the first few bytes of the procedure text; to find the location of the procedure, use "&" to take the address of the variable).

COFF compilers introduce a few problems, presumably in part because debuggers have not heretofore provided a print declaration facility, e.g., the "whatis" command of *dbx*. Where objects are defined in terms of typedefs, the symbol table may not show the typedef for the object type, but may rather use the typedef's definition. Though semantically equivalent, this defeats the purpose of having such debugger commands. Additionally, if a typedef occurs prior to the definition of its type, e.g., the typedef is a pointer to a yet to be defined structure, the symbol entry for the typedef emitted may not contain the resolved type. Under this condition, *dbx* may indicate that the typedef is a pointer to an unknown type. Placing the typedef after the structure definition solves this anomaly.

It should be stated that different COFF compilers may vary in how such symbols are generated, and that *dbx* will exhibit somewhat different behavior as a result.

Because *dbx* uses its own internal symbol table, the start up time is impacted by translating COFF symbols to the internal format required. For very large programs, this may be significant; the *cprs(1)* utility may improve start up time.

On the m88k series of processors, register values reported by *dbx* may not always be correct, specifically when reported for a frame environment other than the top of stack. This is because compiler generated and assembly code may either reuse registers without saving the original values, or because in general the flow of execution must be known to determine if and where original values were saved. Argument registers (i.e., $r2 thru $r9), return values (e.g., $r2) and procedure temporary registers (e.g., $r10 thru $r13) may not be correct. Caution is advised in interpreting the values reported for these registers for stack frames other than the top of stack.

Symbol names on the m88k are often prepended with an underscore ('_'). For convenience, *dbx* permits the use of the unprepended symbol name in most instances, but if this name references a different symbol, the prepended name should be explicitly specified. For example, '**main**' may be used to reference the function **_main**, if no other object uses the unprepended name, but sometimes '**main**' may refer to a file. If a command appears to fail given an unprepended name, the type of symbol referenced can be determined by "*whatis* **name**".

The current implementation may not fully support languages other than 'C'. In particular, type information is problematic, since COFF does not naturally represent types of other languages. For instance, the Fortran LOGICAL type is usually treated as a signed int by the compiler, where zero is '.false.' and non-zero is '.true.'. *Dbx* will print the signed int value when so queried, and will report an instantiation of the type as a signed int. Differences may exist for other types and for other languages.

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, otherwise, the scale factor is ignored.

!
Interprets the rest of the line as a command to the shell.

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.

; :
Used by *bc*(1) for array operations.

EXAMPLE

This example prints the first ten values of n!:

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

SEE ALSO

bc(1).

DIAGNOSTICS

x is unimplemented
where x is an octal number.

stack empty
for not enough elements on the stack to do what was asked.

Out of space
when the free list is exhausted (too many digits).

Out of headers
for too many numbers being kept around.

Out of pushdown
for too many items on the stack.

Nesting Depth
for too many levels of nested execution.

NAME

dcpy – copy removeable media

SYNOPSIS

dcpy *from to*

DESCRIPTION

dcpy provides a method for non-root users to make copies of removeable media file systems. To actually copy the file systems, it calls the *dcopy* utility.

There are no options for *dcpy*. *from* and *to* are the names of removable media filesystems, and must be specified.

The *from* filesystem must be present and readable.

The *to* filesystem must be present and writeable.

The blocksize of the *to* filesystem will be made the same as the blocksize of the *from* filesystem. *dcpy* itself uses 1Kb blocks.

FILES

/etc/filesys	permissions file
/etc/dcopy	system utility program

SEE ALSO

dcopy(1M).

DIAGNOSTICS

dcpy will complain, **No read permission on input file.** if the input file (the *from* file) does not exist or is not readable.

dcpy will complain, **No write permission on output file.** if the output device (the *to* file) does not exist or is not readable.

dcpy will print **Failed to exec dcopy**" if, for some reason, it was unable to execute **/etc/dcopy.**

Additional error messages may be printed by the *dcopy* program itself.

NAME

delta – make a delta (change) to an SCCS file

SYNOPSIS

delta [**–r***SID*] [**–s**] [**–n**] [**–g***list*] [**–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 *gets* for editing (**get –e**) on the same SCCS file were done by the same person (login name). The SID value specified with the **–r** keyletter can be either the SID specified on the *get* command line or the SID to be made as reported by the *get* command (see *get*(1)). A diagnostic results if the specified SID is ambiguous, or, if necessary and omitted on the command line.

–s

Suppresses the issue, on the standard output, of the created delta's SID, as well as the number of lines inserted, deleted and unchanged in the SCCS file.

–n

Specifies retention of the edited *g-file* (normally removed at completion of delta processing).

–g*list*

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 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) that 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**.

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) (5)) 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.

SEE ALSO

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

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

depend – generate C program makefile dependencies

SYNOPSIS

depend [cppopts] [opts] [+]file...

DESCRIPTION

The *depend* command generates makefile dependencies and (optionally) rules for each *file* specified on its invocation line. The contents of *file* must be acceptable input to the C preprocessor *cpp*. The output of *depend* is written to standard output.

For each *file* specified *depend* produces as output one or more target names followed by a list of dependents and (optionally) a set of rules for building the target in a format conforming to that required by the *make* command:

target1 [target2 ...] :[:] [dependent1 ...] [; rule1 ...]

Typically the output of the *depend* command will be appended to a makefile using redirection of the output. By default the target name is generated by using as a root name all characters of *file* up to, but not including, the last . and any succeeding characters found in the filename, and appending a .o to the root formed. Thus for input filenames of file.c, file.x.c, and file the target names produced would be file.o, file.x.o and file.o, respectively.

depend generates the dependents by reading the contents of *file*, passing them to the preprocessor, and filtering and outputting the filename portion of all #include statements produced in the preprocessor's output. Additional dependents may be forcibly added to the list using options described below. The preprocessor rules and options for the searching of #include files are preserved.

If *file* is preceded by a + then *file* will not be included in the list of dependents unless explicitly specified via the DEPENDENT[S]= option.

The following *options* to *depend* are recognized:

cppopts is a subset of the options recognized by *cpp* and consists of the -I, -D, -U, -C and -T options. These options and any associated arguments are passed along by *depend* to *cpp*. Any other options specified, whether recognized by *cpp* or not, are ignored.

opts are options which allow the user to alter the default output of *depend*. Multiple options may be specified and all options (except where noted) may be specified as many times as needed. Option assignments must be enclosed in quotes to ensure they are passed to depend as a single argument. The following options are recognized and apply to each *file* specified:

TARGET[S]="*target1 [target2 ...]***"** Overrides the target name generated by *depend*. If, in addition to other *target(s)*, the default target name is desired it too must be specified via this option.

DEPENDENT[S]="*dependent1 [dependent2 ...]***"** Adds *dependent(s)* to the list generated by *depend*. If multiple "DEPENDENT[S]=" options are present the *dependents* are output in the order the options are specified on the command line scanning from left to right.

RULE[S]="*rule1 [; rule2 ...]***"** Adds the specified *rule(s)* or commands for building the target to the output. RULE[S]= is of use when the implicit *make* rules are not adequate. If multiple "RULE[S]=" options are present the *rules* are output in the order the options are specified on the command line scanning from left to right. If more than one *rule* is specified in one invocation of this option, then the second and additional rules must be preceded by a ';'.

INCRT[S]="*root1 newroot1 [root2 newroot2 ...]***"** Allows part or all of the directory name portion of a dependency generated by *depend* to be modified. The pattern specified in *root* is matched against all dependent names and replaced by *newroot*. If multiple *root newroot* pairs are specified, only the first *root* match scanning from left to right in the command line is applied. All characters represent themselves; no meta characters are supported.

COLON=:|:: Specifies whether to terminate a target list on a dependency line with a : or :: . This option may only be specified once. If not specified the default value is : .

EXAMPLES

Assume that:

```
depend file.c
```

produces as output:

```
file.o: \
    file.c \
    /usr/include/includefile.h
```

Then the output made can be modified using the preceding options as follows:

```
depend +file.c
```

produces:

```
file.o: \
    /usr/include/includefile.h
```

```
depend 'TARGET=target.o' file.c
```

produces:

```
target.o: \
    file.c \
    /usr/include/includefile.h
```

```
depend 'DEPENDENT=dependentfile' file.c
```

produces:

```
file.o: \
    file.c \
    /usr/include/includefile.h \
    dependentfile
```

```
depend 'RULE = cc file.c -o file' 'RULE = cp file /bin' file.c
```

produces:

```
file.o: \
    file.c \
    /usr/include/includefile.h
    cc file.c -o file
    cp file /bin
```

```
depend 'INCRT=/usr/include $(INCRT)' file.c
```

produces:

```
file.o: \
    file.c \
```

```
$(INCRT)/includefile.h
```

depend 'COLON=::' file.c

produces:

```
file.o:: \
    file.c \
    /usr/include/includefile.h
```

SEE ALSO

cpp(1), make(1).

DIAGNOSTICS

Error messages will be generated by *cpp* if the contents of *file* are unsuitable. Those generated by *depend* are, for the most part, self-explanatory. Errors can be generated due to special characters being interpreted improperly, so watch for characters that need to be escaped. Any argument not recognized as an option will be treated as a filename. The exit code is zero for a normal exit, nonzero if errors were detected by *depend* or *cpp*.

NAME

deroff – remove nroff/troff, tbl, and eqn constructs

SYNOPSIS

deroff [**–mx**] [**–w**] [*files*]

DESCRIPTION

deroff reads each of the *files* in sequence and removes all *troff*(1) requests, macro calls, backslash constructs, *eqn*(1) constructs (between **.EQ** and **.EN** lines, and between delimiters), and *tbl*(1) descriptions, perhaps replacing them with white space (blanks and blank lines), and writes the remainder of the file on the standard output. *deroff* follows chains of included files (**.so** and **.nx** *troff* commands); if a file has already been included, a **.so** naming that file is ignored and a **.nx** naming that file terminates execution. If no input file is given, *deroff* reads the standard input.

The **–m** option may be followed by an **m**, **s**, or **l**. The **–mm** option causes the macros to be interpreted so that only running text is output (i.e., no text from macro lines). The **–ml** option forces the **–mm** option and also causes deletion of lists associated with the **mm** macros.

If the **–w** option is given, the output is a word list, one "word" per line, with all other characters deleted. Otherwise, the output follows the original, with the deletions mentioned above. In text, a "word" is any string that *contains* at least two letters and is composed of letters, digits, ampersands (&), and apostrophes ('); in a macro call, however, a "word" is a string that *begins* with at least two letters and contains a total of at least three letters. Delimiters are any characters other than letters, digits, apostrophes, and ampersands. Trailing apostrophes and ampersands are removed from "words."

SEE ALSO

eqn(1), nroff(1), tbl(1), troff(1) in the *DOCUMENTER'S WORKBENCH Software Release 2.0 Technical Discussion and Reference Manual.*

BUGS

deroff is not a complete *troff* interpreter, so it can be confused by subtle constructs. Most such errors result in too much rather than too little output.

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

NAME

df – report number of free disk blocks and i-nodes

SYNOPSIS

df [**–ft** | **–b**] [**–l**] [*file-system* | *path-name* | *mounted-resource*]

DESCRIPTION

df prints out the number of free blocks and free i-nodes in file systems, pathnames, or mounted resources by examining the counts kept in the superblocks.

file-system may be specified either by device name (e.g., **/dev/dsk/m323_0s0**) or by mount point directory name (e.g., **/usr**).

path-name can be a directory, regular file, or FIFO name. The report presents information for the device that contains the pathname.

mounted-resource can be a remote resource name. The report presents information for the remote device that contains the resource.

If no arguments are used, the free space on all locally and remotely mounted file systems is printed.

df uses the following options:

–l
reports on local file systems only.

–t
causes the figures for total allocated blocks and i-nodes to be reported as well as the free blocks and i-nodes.

–f
makes an actual count of the blocks in the free list instead of taking the figure from the superblock (free i-nodes are not reported). This option does not print any information about mounted remote resources.

–b
produces *df* output in the BSD 4.x table format. File system information is reported in 512-byte blocks.

NOTES

If multiple remote resources are listed that reside on the same file system on a remote machine, each listing after the first one is marked with an asterisk.

If the device on which a file system is mounted cannot be determined, the string ****UNKNOWN**** is printed in place of the physical device name. This can occur whenever *df* tries to access a remote, networked file system, or whenever **/etc/mnttab** is corrupted.

When the **–b** option is selected, it is possible to see file system utilization of more than 100%. The numbers in the columns **Total**, **Used**, **Avail**, and **%Full** are usually based only on the space available to users and do not show any data on the space reserved by the System Administrator. However, when the user space is 100% full, the space used by the System Administrator is added to the figure in the **%Full** column to show a number greater than 100. Note also that for small systems (less than 1000 512-byte blocks), the number in the **%Full** column can exceed 100 plus the percentage of space reserved by the System Administrator. This is due to rounding off in integer calculations.

When the **–f** option is used on a mounted file system, the results may be inaccurate or an error message may appear.

FILES

/dev/dsk/*
/etc/mnttab

SEE ALSO

mkfs(1M), tunefs(1M), mount(1M), fs(4), mnttab(4).

NAME

diff – differential file comparator

SYNOPSIS

diff [**–efbh**] *file1 file2*

DESCRIPTION

diff tells what lines must be changed in two files to bring them into agreement. If *file1* (*file2*) is –, the standard input is used. If *file1* (*file2*) is a directory, then a file in that directory with the name *file2* (*file1*) is used. The normal output contains lines of these forms:

n1 a n3,n4
n1,n2 d n3
n1,n2 c n3,n4

These lines resemble *ed* commands to convert *file1* into *file2*. The numbers after the letters pertain to *file2*. In fact, by exchanging **a** for **d** and reading backward one may ascertain equally how to convert *file2* into *file1*. As in *ed*, identical pairs, where *n1* = *n2* or *n3* = *n4*, are abbreviated as a single number.

Following each of these lines come all the lines that are affected in the first file flagged by <, then all the lines that are affected in the second file flagged by >.

The **–b** option causes trailing blanks (spaces and tabs) to be ignored and other strings of blanks to compare equal.

The **–e** option produces a script of *a*, *c*, and *d* commands for the editor *ed*, which will recreate *file2* from *file1*. The **–f** option produces a similar script, not useful with *ed*, in the opposite order. In connection with **–e**, the following shell program may help maintain multiple versions of a file. Only an ancestral file ($1) and a chain of version-to-version *ed* scripts ($2,$3,...) made by *diff* need be on hand. A "latest version" appears on the standard output.

```
(shift; cat $*; echo '1,$p') | ed - $1
```

Except in rare circumstances, *diff* finds a smallest sufficient set of file differences.

Option **–h** does a fast, half-hearted job. It works only when changed stretches are short and well separated, but does work on files of unlimited length. Options **–e** and **–f** are unavailable with **–h**.

FILES

/tmp/d?????
/usr/lib/diffh for –h

SEE ALSO

bdiff(1), cmp(1), comm(1), ed(1).

DIAGNOSTICS

Exit status is 0 for no differences, 1 for some differences, 2 for trouble.

BUGS

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

WARNINGS

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

NAME

diff – differential file and directory comparator

SYNOPSIS

diff [**-l**] [**-r**] [**-s**] [**-cefhn**] [**-biwt**] dir1 dir2
diff [**-cefhn**] [**-biwt**] file1 file2
diff [**-D***string*] [**-biw**] file1 file2

DESCRIPTION

If both arguments are directories, *diff* sorts the contents of the directories by name, and then runs the regular file *diff* algorithm (described below) on text files which are different. Binary files which differ, common subdirectories, and files which appear in only one directory are listed. Options when comparing directories are:

-l long output format; each text file *diff* is piped through *pr*(1) to paginate it, other differences are remembered and summarized after all text file differences are reported.

-r causes application of *diff* recursively to common subdirectories encountered.

-s causes *diff* to report files which are the same, which are otherwise not mentioned.

-Sname
starts a directory *diff* in the middle beginning with file *name*.

When run on regular files, and when comparing text files which differ during directory comparison, *diff* tells what lines must be changed in the files to bring them into agreement. Except in rare circumstances, *diff* finds a smallest sufficient set of file differences. If neither *file1* nor *file2* is a directory, then either may be given as '-', in which case the standard input is used. If *file1* is a directory, then a file in that directory whose file-name is the same as the file-name of *file2* is used (and vice versa).

There are several options for output format; the default output format contains lines of these forms:

n1 a *n3,n4*
n1,n2 d *n3*
n1,n2 c *n3,n4*

These lines resemble *ed* commands to convert *file1* into *file2*. The numbers after the letters pertain to *file2*. In fact, by exchanging 'a' for 'd' and reading backward one may ascertain equally how to convert *file2* into *file1*. As in *ed*, identical pairs where *n1* = *n2* or *n3* = *n4* are abbreviated as a single number.

Following each of these lines come all the lines that are affected in the first file flagged by '<', then all the lines that are affected in the second file flagged by '>'.

Except for **-b, -w, -i** or **-t** which may be given with any of the others, the following options are mutually exclusive:

-e produces a script of *a, c* and *d* commands for the editor *ed*, which will recreate *file2* from *file1*. In connection with **-e**, the following shell program may help maintain multiple versions of a file. Only an ancestral file ($1) and a chain of version-to-version *ed* scripts ($2,$3,...) made by *diff* need be on hand. A 'latest version' appears on the standard output.

```
(shift; cat $*; echo '1,$p') | ed - $1
```

Extra commands are added to the output when comparing directories with **-e**, so that the result is a *sh*(1) script for converting text files which are common to the two directories from their state in *dir1* to their state in *dir2*.

–f produces a script similar to that of **–e**, not useful with *ed*, and in the opposite order.

–n produces a script similar to that of **–e**, but in the opposite order and with a count of changed lines on each insert or delete command. This is the form used by *rcsdiff*(1).

–c produces a diff with lines of context. The default is to present 3 lines of context and may be changed, e.g to 10, by **–c10**. With **–c** the output format is modified slightly: the output beginning with identification of the files involved and their creation dates and then each change is separated by a line with a dozen *'s. The lines removed from *file1* are marked with '– '; those added to *file2* are marked '+ '. Lines which are changed from one file to the other are marked in both files with with '! '.

Changes which lie within <context> lines of each other are grouped together on output. (This is a change from the previous "diff -c" but the resulting output is usually much easier to interpret.)

–h does a fast, half-hearted job. It works only when changed stretches are short and well separated, but does work on files of unlimited length.

–Dstring

causes *diff* to create a merged version of *file1* and *file2* on the standard output, with C preprocessor controls included so that a compilation of the result without defining *string* is equivalent to compiling *file1*, while defining *string* will yield *file2*.

–b causes trailing blanks (spaces and tabs) to be ignored, and other strings of blanks to compare equal.

–w is similar to **–b** but causes whitespace (blanks and tabs) to be totally ignored. E.g., "if (a == b)" will compare equal to "if(a==b)".

–i ignores the case of letters. E.g., "A" will compare equal to "a".

–t will expand tabs in output lines. Normal or **–c** output adds character(s) to the front of each line which may screw up the indentation of the original source lines and make the output listing difficult to interpret. This option will preserve the original source's indentation.

FILES

/tmp/d?????
/usr/ucb/lib/diffh for **–h**
/bin/ucb/bin/diff for directory diffs
/bin/pr

SEE ALSO

cmp(1), cc(1), comm(1), ed(1), diff(1), diff3(1).

DIAGNOSTICS

Exit status is 0 for no differences, 1 for some, 2 for trouble.

BUGS

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

When comparing directories with the **–b**, **-w** or **-i** options specified, *diff* first compares the files ala *cmp*, and then decides to run the *diff* algorithm if they are not equal. This may cause a small amount of spurious output if the files then turn out to be identical because the only differences are insignificant blank string or case differences.

NAME

diff3 – 3-way differential file comparison

SYNOPSIS

diff3 [**–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 is suppressed.

Under the **–e** option, *diff3* publishes a script for the editor *ed* that will incorporate into *file1* all changes between *file2* and *file3*, i.e., the changes that normally would be flagged ==== and ====3. Option **–x** (**–3**) produces a script to incorporate only changes flagged ==== (====3). The following command will apply the resulting script to *file1*:

(cat script; echo ´1,$p´) | ed – file1

FILES

/tmp/d3*
/usr/lib/diff3prog

SEE ALSO

diff(1).

BUGS

Text lines that consist of a single **.** will defeat **–e**.
Files longer than 64Kb 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 file names common to both directories have the same contents.

–d
Compare the contents of files with the same name in both directories and output a list telling what must be changed in the two files to bring them into agreement. The list format is described in *diff*(1).

–s
Suppress messages about identical files.

–w*n*
Change the width of the output line to *n* characters. The default width is 72.

SEE ALSO

cmp(1), diff(1).

NAME

dis – object code disassembler

SYNOPSIS

dis [**–o**] [**–V**] [**–L**] [**–d** *sec*] [**–da** *sec*] [**–F** *function*] [**–t** *sec*] [**–l** *string*] *file* ...

DESCRIPTION

The *dis* command produces an assembly language listing of *file*, which may be an object file or an archive of object files. The listing includes assembly statements and an octal or hexadecimal representation of the binary that produced those statements.

The following *options* are interpreted by the disassembler and may be specified in any order:

–o
print numbers in octal. The default is hexadecimal.

–V
print, on standard error, the version number of the disassembler being executed.

–L
lookup source labels in the symbol table for subsequent printing. This option works only if the file was compiled with additional debugging information (e.g., the **–g** option of *cc*(1)).

–d *sec*
disassemble the named section as data, printing the offset of the data from the beginning of the section.

–da *sec*
disassemble the named section as data, printing the actual address of the data.

–F *function*
disassemble only the named function in each object file specified on the command line. The **–F** option may be specified multiple times on the command line.

–t *sec*
disassemble the named section as text.

–l *string*
disassemble the library file specified by *string*. For example, issue the command **dis –l x –l z** to disassemble **libx.a** and **libz.a**. All libraries are assumed to be in **LIBDIR**.

If the **–d**, **–da** or **–t** options are specified, only those named sections from each user-supplied filename are disassembled. Otherwise, all sections containing text are disassembled.

On output, a number enclosed in brackets at the beginning of a line, i.e., **[5]**, represents that the break-pointable line number starts with the following instruction. These line numbers are printed only if the file was compiled with additional debugging information (e.g., the **–g** option of *cc*(1)). An expression such as **<40>** in the operand field or in the symbolic disassembly, following a relative displacement for control transfer instructions, is the computed address within the section to which control is transferred. A function name appears in the first column, followed by **( )**.

FILES

LIBDIR usually **/lib**.

SEE ALSO

as(1), cc(1), ld(1), a.out(4).

DIAGNOSTICS

The self-explanatory diagnostics indicate errors in the command line or problems encountered with the specified files.

NAME

dnp – patch program with **NULL** pointer dereference bug

SYNOPSIS

dnp [–u] *filename*

DESCRIPTION

On the REAL/IX BOS, the kernel is set up so that any program that attempts to dereference a pointer at a **NULL** (or very low) address will dump core. This is because the REAL/IX BOS system does not guarantee there will be zeroes at address 0. However, a workaround has been provided in the kernel. If the *timdat* field in the file header (filehdr) structure has a value of 1, the kernel will provide a block of zeroes at address 0 to allow the program to run until the source code can be fixed.

The *dnp* program changes the value of the *timdat* field in an executable file to 1, so that the utility may be run.

If the **–u** is used, a previous *dnp* on that file will be undone; the *timdat* field in the header will be set to the current time.

When the *dnp* utility is run, several diagnostic messages are printed: the pathname of the file, the *magic number* of the file to be patched (only files with magic number 0555 can be patched), and the old and new values of the *timdat* field.

The *file* command recognizes a file that has been patched in this way, indicating that it is *dnp* kludged.

EXAMPLES

dnp foo

FILES

/usr/lbin/dnp

DIAGNOSTICS

can't change non-0555 files!
When the file you are attempting to patch is not a current REAL/IX executable file with a *magic number* of 0555.

Error: fopen on *filename* failed
When the file you are attempting to patch does not exist or cannot be read or written.

NOTES

The *dnp* utility should only be used to patch programs until the source code can be fixed.

Utilities may dump core for reasons other than **NULL** pointer problems; if the utility continues to dump core after being *dnp* patched, there is some other problem.

Please be sure to report to your representative any utilities in which you discover **NULL** pointer dereference problems, so that they may be fixed in code.

Library routines (in *libc.a* and other libraries) may dump core with a **NULL** pointer if they are passed **NULL** pointers; for example, a *strcmp*() where one of the arguments is a **NULL** pointer. This should be fixed in the code that calls *strcmp*().

NAME

domainname – set or display name of current NIS domain

SYNOPSIS

domainname [*nameofdomain*]

DESCRIPTION

Without an argument, *domainname* displays the name of the current domain. Only the superuser can set the domain name by giving an argument; this is usually done in the startup script **/etc/rc2.d/S55yp**. (The System Administrator should run **/usr/etc/nfs.config** or **sysadm nfsmgmt** to set the domain name and enable appropriate NIS (Network Information Service) and NFS daemons. See also the *ONC User's Guide*). Domains are used by the NIS to refer collectively to a group of hosts.

FILES

/usr/bin/domainname

SEE ALSO

ypinit(1M)

NAME

dump – dump selected parts of an object file

SYNOPSIS

dump [*options*] *files*

DESCRIPTION

The *dump* command dumps selected parts of each of its object *file* arguments.

This command will accept both object files and archives of object files. It processes each file argument according to one or more of the following options:

–a
Dump the archive header of each member of each archive file argument.

–g
Dump the global symbols in the symbol table of an archive.

–f
Dump each file header.

–o
Dump each optional header.

–h
Dump section headers.

–s
Dump section contents.

–r
Dump relocation information.

–l
Dump line number information.

–t
Dump symbol table entries.

–z~*name*
Dump line number entries for the named function.

–c
Dump the string table.

–L
Interpret and print the contents of the *.lib* sections.

The following *modifiers* are used in conjunction with the options listed above to modify their capabilities.

–d~*number*
Dump the section number, *number*, or the range of sections starting at *number* and ending at the *number* specified by **+d**.

+d~*number*
Dump sections in the range either beginning with first section or beginning with section specified by **–d**.

–n~*name*
Dump information pertaining only to the named entity. This *modifier* applies to **–h**, **–s**, **–r**, **–l**, and **–t**.

−p
Suppress printing of the headers.

−t *index*
Dump only the indexed symbol table entry. The **−t** used in conjunction with **+t**, specifies a range of symbol table entries.

+t *index*
Dump the symbol table entries in the range ending with the indexed entry. The range begins at the first symbol table entry or at the entry specified by the **−t** option.

−u
Underline the name of the file for emphasis.

−v
Dump information in symbolic representation rather than numeric (e.g., C_STATIC instead of **0X02**). This *modifier* can be used with all the above options except **−s** and **−o** options of *dump*.

−z *name,number*
Dump line number entry or range of line numbers starting at *number* for the named function.

+z *number*
Dump line numbers starting at either function *name* or *number* specified by **−z**, up to *number* specified by **+z**.

Blanks separating an *option* and its *modifier* are optional. The comma separating the name from the number modifying the **−z** option may be replaced by a blank.

The *dump* command attempts to format the information it dumps in a meaningful way, printing certain information in character, hex, octal or decimal representation as appropriate.

SEE ALSO

a.out(4), ar(4).

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).

CAVEATS

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

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 to say, the file is read into *ed*'s buffer so that it can be edited.

–s
Suppresses the printing of character counts by *e*, *r*, and *w* commands, of diagnostics from *e* and *q* commands, and of the ! prompt after a !*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 or not. 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.

–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 will only allow editing of files in the current directory. It 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 **stty –tabs** or **stty tab3** mode (see *stty*(1)), the specified tab stops will automatically be used when scanning *file*. For example, if the first line of a file contained:

<:t5,10,15 s72:>

tab stops would be set at columns 5, 10, and 15, and a maximum line length of 72 would be imposed.

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. Every command that requires addresses has default addresses, so that the addresses can very often be omitted.

In general, only one command may appear on a line. Certain commands allow the input of text. This text is placed in the appropriate place in the buffer. While *ed* is accepting text, it is said to be in *input mode*. In this mode, *no* commands are recognized; all input is merely collected. 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 (e.g., *s*) to specify portions of a line that are to be substituted. A regular expression (RE) specifies a set of character strings. A member of this set of strings is said to be *matched* by the RE. The REs allowed by *ed* are constructed as follows:

The following *one-character* RE s match a *single* character:

1.1 An ordinary character (*not* one of those discussed in 1.2) is a one-character RE that matches itself.

1.2 A backslash (\) followed by any special character is a one-character RE that matches the special character itself. The special characters are:

a. ., *, [, and \ (period, asterisk, left square bracket, and backslash, respectively), which are always special, *except* when they appear within square brackets ([]; see 1.4).

b. ^ (caret or circumflex), which is special at the *beginning* of an *entire* RE (see 3.1 and 3.2 below), or when it immediately follows the left of a pair of square brackets ([]) (see 1.4 below).

c. $ (dollar sign), which is special at the *end* of an entire RE (see 3.2).

d. The character used to bound (i.e., delimit) an entire RE, which is special for that RE (for example, see how slash (/) is used in the *g* command, below.)

1.3 A period (.) is a one-character RE that matches any character except 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 ^ has this special meaning *only* if it occurs first in the string.

The minus (–) may be used to indicate a range of consecutive ASCII characters; for example, **[0–9]** is equivalent to **[0123456789]**. The – loses this special meaning if it occurs first (after an initial ^, if any) or last in the string. The right square bracket (]) does not terminate such a string when it is the first character within it (after an initial ^, if any); e.g., **[]a–f]** matches either a right square bracket (]) or one of the letters **a** through **f** inclusive. The four characters listed in 1.2.a 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 also the last paragraph before the **FILES** section.

To understand addressing in *ed* it is necessary to know that at any time there is a *current line*. Generally, the current line is the last line affected by a command; the exact effect on the current line is discussed under the description of each command. *Addresses* are constructed as follows:

1. The character . addresses the current line.
2. The character $ addresses the last line of the buffer.
3. A decimal number *n* addresses the *n*-th line of the buffer.
4. 'x addresses the line marked with the mark name character *x*, which must be 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 also the last paragraph before the **FILES** section.
6. A RE enclosed in question marks (?) addresses the first line found by searching *backward* from the line *preceding* the current line toward the beginning of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the end of the buffer and continues up to and including the current line. See also the last paragraph before the **FILES**.
7. An address followed by a plus sign (+) or a minus sign (-) followed by a decimal number specifies that address plus (respectively minus) the indicated number of lines. The plus sign may be omitted.
8. If an address begins with + or -, the addition or subtraction is taken with respect to the current line; e.g., -5 is understood to mean .-5.
9. If an address ends with + or -, then 1 is added to or subtracted from the address, respectively. As a consequence of this rule and of Rule 8, immediately above, the address - refers to the line preceding the current line. (To maintain compatibility with earlier versions of the editor, the character ^ in addresses is entirely equivalent to -.) Moreover, trailing + and - characters have a cumulative effect, so -- refers to the current line less 2.
10. For convenience, a comma (,) stands for the address pair 1,$, while a semicolon (;) stands for the pair .,$.

Commands may require zero, one, or two addresses. Commands that require no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when an insufficient number of addresses is given; if more addresses are given than such a command requires, the last one(s) are used.

Typically, addresses are separated from each other by a comma (,). They may also be separated by a semicolon (;). In the latter case, the current line (.) is set to the first address, and only then is the second address calculated. This feature can be used to determine the starting line for forward and backward searches (see Rules 5 and 6). The second address of any two-address sequence must correspond to a line that follows, in the buffer, the line corresponding to the first address.

In the following list of *ed* commands, the default addresses are shown in parentheses. The parentheses are *not* part of the address; they show that the given addresses are the default.

It is generally illegal for more than one command to appear on a line. However, any command (except *e*, *f*, *r*, or *w*) may be suffixed by **l**, **n**, or **p** in which case the current line is either listed, numbered or printed, respectively, as discussed below under the *l*, *n*, and *p* commands.

(.)a
<text>
.

The *a*ppend command reads the given text and appends it after the addressed line; . is left at the last inserted line, or, if there were none, at the addressed line. Address 0 is legal for this command: it causes the "appended" text to be placed at the beginning of the buffer. The maximum number of characters that may be entered from a terminal is 256 per line (including the 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 there were none, 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 entered.

(.,.)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, then the named file to be read in; . is set to the last line of the buffer. If no file name is given, the currently remembered file name, if any, is used (see the *f* command). The number of characters read is typed; *file* is remembered for possible use as a default file name in subsequent *e*, *r*, and *w* commands. If *file* is replaced by !, the rest of the line is taken to be a shell (*sh*(1)) command whose output is to be read. Such a *shell* command is *not* remembered as the current file name. See the **DIAGNOSTICS** section.

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*ile-name command changes the currently remembered file name to *file*; otherwise, it prints the currently remembered file name.

(1,$)g/*RE***/***command list*

In the *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 also **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, . is changed to that line, and any *one* command (other than one of the *a*, *c*, *i*, *g*, *G*, *v*, and *V* commands) may be input and is 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 there were none, 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 may be entered from a terminal is 256 per line (including the newline character).

(.,.+1)j

The *j*oin command joins contiguous lines by removing the appropriate newline characters. If exactly one address is given, this command does nothing.

(.)k*x*

The mar*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 (e.g., *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 other 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 may be appended to any other 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 may be appended to any other command other than *e*, *f*, *r*, or *w*. For example, *dp* deletes the current line and prints the new current line.

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 *q*uit command causes *ed* to exit. No automatic write of a file is done; however, see **DIAGNOSTICS**.

Q

The editor exits without checking if changes have been made in the buffer since the last *w* command.

(.)r *file*

The *r*ead command reads in the given file after the addressed line. If no file name is given, the currently remembered file name, if any, is used (see *e* and *f* commands). The currently remembered file name is *not* changed unless *file* is the very first file name mentioned since *ed* was invoked. Address 0 is legal for *r* and causes the file to be read at the beginning of the buffer.

If the read is successful, the number of characters read is typed; . is set to the last line read in. If *file* is replaced by !, the rest of the line is taken to be a *shell* (*sh*(1)) command whose output is to be read. For example, "$r !ls" appends current directory to the end of the file being edited. Such a shell command is *not* remembered as the current file name.

(.,.)s/*RE*/*replacement*/ or
(.,.)s/*RE*/*replacement*/**g** or
(.,.)s/*RE*/*replacement*/**n** n = 1-512

The *s*ubstitute 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 also 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 may be split by substituting a newline character into it. The newline in the *replacement* must be escaped by preceding it by \. 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 file name is *not* changed unless *file* is the very first file name mentioned since *ed* was invoked.

If no file name is given, the currently remembered file name, if any, is used (see *e* and *f* commands); . is unchanged. If the command is successful, the number of characters written is typed. If *file* is replaced by !, the rest of the line is taken to be a shell (*sh*(1)) command whose standard input is the addressed lines. Such a shell command is *not* remembered as the current file name.

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 (*sh*(1)) to be interpreted as a command. Within the text of that command, the unescaped character % is replaced with the remembered file name; if a ! appears as the first character of the shell command, it is

replaced with the text of the previous shell command. Thus, !! will repeat the last shell command. If any expansion is performed, the expanded line is echoed; . is unchanged.

(.+1)<newline>
An address alone on a line causes the addressed line to be printed. 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 of a replacement string (e.g., /) would be the last character before a newline, that delimiter may be omitted, in which case the addressed line is printed. The following pairs of commands are equivalent:

s/s1/s2	s/s1/s2/p
g/s1	g/s1/p
?s1	?s1?

FILES

$ 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 environmental 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.

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)).

SEE ALSO

edit(1), ex(1), grep(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 *h*elp and *H*elp commands for detailed explanations).

If changes have been made in the buffer since the last *w* command that wrote the entire buffer, *ed* warns the user if an attempt is made to destroy *ed*'s buffer via the *e* or *q* commands. It prints ? and allows one to continue editing. A second *e* or *q* command at this point will take effect. The –s command-line option inhibits this feature.

WARNINGS

The encryption options and commands are provided with the Security Administration Utilities package, which is 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 (see *sh*(1)).

The sequence **\n** in a RE does not match a newline character.

If the editor input is coming from a command file (e.g., **ed file < ed-cmd-file**), the editor will exit at the first failure.

NAME

edit – text editor (variant of ex for casual users)

SYNOPSIS

edit [–r] [–x] [–C] *name...*

DESCRIPTION

edit is a variant of the text editor *ex* recommended for new or casual users who wish to use a command-oriented editor. It operates precisely as *ex* (1) with the following options automatically set:

novice	ON
report	ON
showmode	ON
magic	OFF

These options can be turned on or off via the *set* command in *ex* (1).

–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 or not. See *crypt* (1). Also, see the **WARNING** section at the end of this manual page.

–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, you begin with the command *edit name* to the shell. *edit* makes a copy of the file that you can then edit, and tells you how many lines and characters are in the file. To create a new file, you also begin with the command *edit* with a file name: *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, then you will have some lines 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 carriage return (as you should after all *edit* commands), the current line will be printed. 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 one.

If you start with an empty file or wish to add some new lines, then the *append* (*a*) command can be used. After you execute this command (typing a carriage return after the word *append*), *edit* will read lines from your terminal until you type a line consisting of just a dot (.); it 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**, then *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 some 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 you want to replace *old* with.

The command *file* (*f*) will tell you how many lines there are in the buffer you are editing and will say [Modified] if you have changed 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, i.e., **3,5c**. You can print lines this way too: **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 carriage return. To look at a number of lines, type **CTRL-d** (while holding down the control key, press **d**) rather than carriage return. This will show you a half-screen of lines on a CRT or 12 lines on a hardcopy terminal. You can look at 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.11** to display five lines before and five lines after the current line. (Typing **z.***n*, when *n* is an odd number, displays a total of *n* lines, centered about the current line; when *n* is an even number, it displays *n*–1 lines, so that the lines displayed are centered around the current line.) You can give counts after other commands; for example, you can delete 5 lines starting with the current line with the command **d5**.
To find things in the file, you can use line numbers if you happen to know them; since the line numbers change when you insert and delete lines this is somewhat unreliable. You can search backwards and forwards in the file for strings by giving commands of the form /*text*/ to search forward for *text* or ?*text*? to search backward for *text* . If a search reaches the end of the file without finding *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 leave off the trailing / or ? in these commands.

The current line has the symbolic name dot (**.**); this is most useful in a range of lines as in **.,$p** which 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 line **$–5** is the fifth before the last and **.+20** is 20 lines after the current line.

You can find out the current line by typing **.=** . This is useful if you want 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; following the *e* command with the name of the other file you wish to edit, i.e., *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)

WARNING

The encryption options are provided as a separate package only to source product customers in the United States.

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 \(and \), with the addition of:

1. A full regular expression followed by + that matches one or more occurrences of the full regular expression.
2. A full regular expression followed by ? that matches 0 or 1 occurrences of the full regular expression.
3. Full regular expressions separated by | or by a 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 expression*, because they are also meaningful to the shell. It is safest to enclose the entire *full regular expression* in single quotes ´...´.

The order of precedence of operators is [], then * ? +, then concatenation, then | and newline.

If no files are specified, *egrep* assumes standard input. Normally, each line found is copied to the standard output. The file name is printed before each line found if there is more than one input file.

Command line options are:

–b
Precede each line by the block number on which it was found. This can be useful in locating block numbers by context (first block is 0).

–c
Print only a count of the lines that contain the pattern.

–i
Ignore upper-lowercase distinction during comparisons.

–l
Print the names of files with matching lines once, separated by newlines. It does not repeat the names of files when the pattern is found more than once.

–n
Precede each line by its line number in the file (first line is 1).

–v
Print all lines except those that contain the pattern.

–e *special_expression*
Search for a *special expression* (*full regular expression* that begins with a –).

–f *file*
Take the list of *full regular expressions* from *file*.

SEE ALSO

ed(1), fgrep(1), grep(1), sed(1), sh(1)

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files (even if matches were found).

Some complex regular expressions may cause *egrep* to overflow an internal table. If this happens, the following message displays:

egrep: regular expression too long

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. Options for use with *disable* are:

–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 the disabling of 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, then 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

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 is a command 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, and uses the terminal capability data base (see *terminfo* (4)) and the type of the terminal you are using from the environmental variable TERM to determine how to drive your terminal efficiently. The editor makes use of features such as insert and delete character and line in its **visual** command (which can be abbreviated *vi*) and which is the central mode of editing when using *vi* (1).

ex contains a number of features for easily viewing the text of the file. The *z* command gives easy access to windows of text. Typing **CTRL-d** causes the editor to scroll down 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 file names, or a pattern as used by the shell to specify a new set of files to be dealt with. In general, file names in the editor may be formed with full shell metasyntax. The metacharacter '%' is also available in forming file names 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 where it is available to be inserted elsewhere in the file. The contents of these 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 regular expressions which match words to be constructed. 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 tailor it to your liking. One option which is very useful is the *autoindent* option that allows the editor to supply leading white space to align text automatically. You can then use **CTRL-d** as a backtab and space or tab to move forward to align new code easily.

Miscellaneous useful features include an intelligent *join* (*j*) command that supplies white space between joined lines automatically, commands "**<**" and "**>**" which shift groups of lines, and the ability to filter portions of the buffer through commands such as *sort* (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 or not. 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.

−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

forced encryption	**C**	heuristic encryption	**X**
resubst	**&**	print next	CR
rshift	**>**	lshift	**<**
scroll	**CTRL-d**	window	**z**
shell escape	**!**		

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
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*

Most useful options and their abbreviations

autoindent	**ai**	supply indent
autowrite	**aw**	write before changing files
directory		pathname of directory for temporary work files
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
/tmp/Ex*nnnnn*	editor temporary
/tmp/Rx*nnnnn*	named buffer temporary
/usr/preserve/*login*	preservation directory (where *login* is the user's login)

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**, a **-r** option that is not followed with an option-argument has been replaced by **-L**, and **+***command* has been replaced by **-c** *command*.

SEE ALSO

crypt(1), ed(1), edit(1), grep(1), sed(1), sort(1), vi(1), curses(3X), term(4), terminfo(4).

WARNINGS

The encryption options and commands are provided as a separate package only to source product customers 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

expand, unexpand – expand tabs to spaces, and vice versa

SYNOPSIS

expand [-tabstop] [-tab1,tab2,...,tabn] [file ...]
unexpand [-a] [file ...]

DESCRIPTION

Expand processes the named files or the standard input writing the standard output with tabs changed into blanks. Backspace characters are preserved into the output and decrement the column count for tab calculations. *Expand* is useful for pre-processing character files (before sorting, looking at specific columns, etc.) that contain tabs.

If a single *tabstop* argument is given, then tabs are set *tabstop* spaces apart instead of the default 8. If multiple tabstops are given then the tabs are set at those specific columns.

Unexpand puts tabs back into the data from the standard input or the named files and writes the result on the standard output. By default, only leading blanks and tabs are reconverted to maximal strings of tabs. If the **-a** option is given, then tabs are inserted whenever they would compress the resultant file by replacing two or more characters.

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" (i.e., begin with ^) and, therefore, ^ is not a special character, in that context. Normally, the matching operator returns the number of characters matched (**0** on failure). Alternatively, the \(...\) pattern symbols can be used to return a portion of the first argument.

EXAMPLES

1. a=`expr $a + 1`

adds 1 to the shell variable **a**.

2. # 'For $a equal to either "/usr/abc/file" or just "file"'
expr $a : '.*/\(.*\)' \| $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).

3. # 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.

4. expr $VAR : '.*'

   returns the number of characters in $VAR.

SEE ALSO

ed(1), sh(1).

DIAGNOSTICS

As a side effect of expression evaluation, *expr* returns the following exit values:

0	if the expression is neither **NULL** nor **0**
1	if the expression *is* **NULL** or **0**
2	for invalid expressions.
syntax error	for operator/operand errors
non-numeric argument	if arithmetic is attempted on such a string

BUGS

After argument processing by the shell, *expr* cannot tell the difference between an operator and an operand except by the value. If $a is an =, the command:

expr $a = '='

looks like:

expr = = =

as the arguments are passed to *expr* (and they will all be taken as the = operator). The following works:

expr X$a = X=

NAME

factor – obtain the prime factors of a number

SYNOPSIS

factor [*integer*]

DESCRIPTION

When you use *factor* without an argument, it waits for you to give it an integer. After you give it a positive integer less than or equal to 10^{14}, it factors the integer, prints its prime factors the proper number of times, and then waits for another integer. *factor* exits if it encounters a zero or any non-numeric character.

If you invoke *factor* with an argument, it factors the integer as described above, and then it exits.

The maximum time to factor an integer is proportional to $\sqrt{n}$. *factor* will take this time when n is prime or the square of a prime.

DIAGNOSTICS

factor prints the error message, Ouch, for input out of range or for garbage input.

NAME

fdivfix – patch XC88100 COFF executables to avoid fdiv deadlock

SYNOPSIS

fdivfix [–bhlv] [–#< *str*>] *files*

DESCRIPTION

fdivfix is a binary patch program for XC88100 COFF executables. The XC88100 RCXXE CPU errata version 1.1, dated August 31, 1990, describes a hardware bug, erratum #4, as:

> "If an fdiv enters the add pipe while stage 2 is empty, stage 3 is full and the add pipe is waiting for a write- back slot, the 88100 will fail to execute the fdiv. However, the scoreboard bit for the destination register remains set."

The fact that the destination register scoreboard bit remains set causes an internal deadlock in the XC88100 chip on the next access to that register. A reset is the only way to get out of the deadlock.

fdivfix locates all fdiv instructions in existing executable files, and changes them into a branch to a small code stub. This stub executes a "trap never" instruction, followed by a copy of the fdiv instruction, and then returns execution to the next instruction after the original fdiv. The "trap never" instruction allows the pipelines to drain before executing the fdiv, thus avoiding any possibility of the deadlock occuring.

fdivfix accepts as arguments a list of filenames. For each file that is recognized as a COFF executable, it scans the text section looking for fdiv instructions, replacing them as described above while adding the patches to the end of a copy of the text section, and rebuilding the output file using the new text section.

The *fdivfix* command has the following options:

–b
Keep a copy of the original file as a backup. The copy is kept in the same directory as the original, with the backup file name derived from the original file name by prepending a single # character to the name. Since the original files are overwritten, links are preserved, as are file ownership and permissions.

–h
Print a copy of the internal mini-manual page.

–l
Generate a list of files which need to be patched, but perform no actual patches.

–v
Enable verbose mode, which prints information about each file as it is processed.

–#< str>
Enable the internal debugging system, if compiled in. Most versions will not have this option available.

EXAMPLES

Print a copy of the internal mini-manual page.

```
# fdivfix –h
```

Scan a directory to see what files potentially need to be patched (have unpatched fdiv instructions). No files are changed by this command:

```
# fdivfix –l /bin/*
/bin/acctcom needs 16 patches
/bin/ar needs 4 patches
/bin/asa needs 2 patches
```

/bin/cat needs 2 patches
(etc)

Scan a directory, as in the previous example, but print more information about each file:

 # fdivfix –lv junk/*
junk/acctcom: reading, 16 fdiv's, 0 previous patches, 16 new patches, done.
junk/ar: reading, 4 fdiv's, 4 previous patches, 0 new patches, done.
junk/asa: reading, 2 fdiv's, 0 previous patches, 2 new patches, done.
junk/cat: reading, 2 fdiv's, 0 previous patches, 2 new patches, done.
junk/dirname: reading, not a COFF file, done.

Patch all files in a directory, printing information about each file processed, and saving a backup file of each file that gets patched:

 # fdivfix –vb junk/*
junk/acctcom: reading, 16 fdiv's, 0 previous patches, 16 new patches, writing, done.
junk/ar: reading, 4 fdiv's, 4 previous patches, 0 new patches, done.
junk/asa: reading, 2 fdiv's, 0 previous patches, 2 new patches, writing, done.
junk/cat: reading, 2 fdiv's, 0 previous patches, 2 new patches, writing, done.
junk/dirname: reading, not a COFF file, done.

NOTES

The impact on performance for floating point intensive programs, e.g., benchmarks, can be significant if fdiv is used extensively.

Debuggers may have problems with the patched files since the mapping of source line numbers to virtual text addresses will not take into account the branch to the stub and the return.

The patched executable can be expected to be at least 64Kb larger than unpatched executables because of data section alignment constraints imposed by the virtual memory system on the lay-out of the COFF file. Since the data section has to move, and can move only by multiples of 64K without relocating it (in the sense of changing the virtual addresses at which the data objects are bound), a 64Kb "hole" is left in the COFF file between the end of original text and the start of data. Some of this hole is filled with the new patch stubs.

Certain pathological cases cannot be handled. In particular, the following code segment will not patch correctly because removal of the delay slot generates a branch offset that is out of range of a 16-bit signed instruction offset:

```
        text
begin:
        zero        (32 * 1024 * 4)
code:
        bb1.n        0,r0,begin
        fdiv.sss     r0,r0,r0
```

BUGS

fdivfix only understands simple COFF files with a single text section, and has a rigid view of where various COFF file components are located. For example, there is no requirement under COFF that the symbol table immediately follows the last line number section, with no "hole" in the file. However, *fdivfix* will create COFF files that have this property, even if the original did not.

NAME

fgrep – search a file for a character string

SYNOPSIS

fgrep [*options*] *string* [*file* ...]

DESCRIPTION

fgrep (fast *grep*) searches files for a character string and prints all lines that contain that string. *fgrep* is different from *grep(1)* and *egrep(1)* because it searches for a string, instead of searching for a pattern that matches an expression. It uses a fast and 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. The file name is printed before each line found if there is more than one input file.

Command line options are:

–b
Precede each line by the block number on which it was found. This can be useful in locating block numbers by context (first block is 0).

–c
Print only a count of the lines that contain the pattern.

–i
Ignore upper/lowercase distinction during comparisons.

–l
Print the names of files with matching lines once, separated by newlines. It does not repeat the names of files when the pattern is found more than once.

–n
Precede each line by its line number in the file (first line is 1).

–v
Print all lines except those that contain the pattern.

–x
Print only lines matched entirely.

–e *special_string*
Search for a *special string* (*string* begins with a –).

–f *file*
Take the list of *strings* from *file*.

SEE ALSO

ed(1), egrep(1), grep(1), sed(1), sh(1).

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files (even if matches were found).

BUGS

Ideally there should be only one *grep* command, but there is not a single algorithm that spans a wide enough range of space-time tradeoffs. Lines are limited to BUFSIZ characters; longer lines are truncated. BUFSIZ is defined in **/usr/include/stdio.h**.

NAME

file – determine file type

SYNOPSIS

file [**–c**] [**–f** *ffile*] [**–m** *mfile*] *arg* ...

DESCRIPTION

file performs a series of tests on each argument in an attempt to classify it. If an argument appears to be ASCII, *file* examines the first 512 bytes and tries to guess its language. If an argument is an executable **a.out**, *file* will print the version stamp, provided it is greater than 0.

–c
The –c option causes *file* to check the magic file for format errors. This validation is not normally carried out for reasons of efficiency. No file typing is done under **–c**.

–f
If the **–f** option is given, the next argument is taken to be a file containing the names of the files to be examined.

–m
The **–m** option instructs *file* to use an alternate magic file.

file uses the file **/etc/magic** to identify files that have some sort of *magic number*, i.e., any file containing a numeric or string constant that indicates its type. Commentary at the beginning of **/etc/magic** explains its format.

FILES

/etc/magic

SEE ALSO

filehdr(4).

NAME

find – find files

SYNOPSIS

find *path-name-list expression*

DESCRIPTION

find recursively descends the directory hierarchy for each pathname in the *path-name-list* (that is, one or more pathnames), seeking files that match a boolean *expression* written in the primaries given below. *find* does *not* follow symbolic links to the resulting file or directory. Instead, it applies the selection criteria to the symbolic link itself. In the descriptions, the argument *n* is used as a decimal integer where $+n$ means more than n, $-n$ means less than n and n means exactly n. Valid expressions are:

–name *file*
True if *file* matches the current file name. Normal shell argument syntax may be used if escaped (watch out for [, ? and *).

[**–perm**] *–onum*
True if file-permission flags exactly match the octal number *onum* (see *chmod*(1)). If *onum* is prefixed by a minus sign, only the bits that are set in *onum* are compared with the file permission flags, and the expression evaluates true if they match.

–type *c*
True if the type of the file is *c*, where *c* is **l**, **b**, **c**, **d**, **p**, or **f** for symbolic-link, block special-file, character special-file, directory, fifo (a.k.a. named pipe), or plain file, respectively.

–links *n*
True if the file has *n* links.

–user *uname*
True if the file belongs to the user *uname*. If *uname* is numeric and does not appear as a login name in the **/etc/passwd** file, it is taken as a user ID.

–group *gname*
True if the file belongs to the group *gname*. If *gname* is numeric and does not appear in the **/etc/group** file, it is taken as a group ID. A message will be displayed if the entry for *gname* in **/etc/group** is incorrect. See the *DIAGNOSTICS* section.

–size *n*[c]
True if the file is *n* blocks long (512 bytes per block). If *n* is followed by a **c**, the size is in characters.

–atime *n*
True if the file has been accessed in *n* days. The access time of directories in *path-name-list* is changed by *find* itself.

–mtime *n*
True if the file has been modified in *n* days.

–ctime *n*
True if the file has been changed in *n* days.

–exec *cmd*
True if the executed *cmd* returns a zero value as exit status. The end of *cmd* must be punctuated by an escaped semicolon. A command argument **{ }** is replaced by the current 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 be printed.

–cpio *device*
Always true; write the current file on *device* in *cpio* (1) format (5120-byte records).

–newer *file*
True if the current file has been modified more recently than the argument *file*.

–depth
Always true; causes descent of the directory hierarchy to be done so that all entries in a directory are acted on before the directory itself. This can be useful when *find* is used with *cpio*(1) to transfer files that are contained in directories without write permission.

–mount
Always true; restricts the search to the file system containing the directory specified, or if no directory was specified, the current directory.

–follow
Always true; forces resolution of symbolic links.

–local
True if the file physically resides on the local system.

(*expression* **)**
True if the parenthesized expression is true. (Parentheses are special to the shell and must be escaped.)

The primaries may be combined using the following operators, in order of decreasing precedence:

1) The negation of a primary (**!** is the unary *not* operator).
2) Concatenation of primaries (the *and* operation is implied by the juxtaposition of two primaries).
3) Alternation of primaries (**–o** is the *or* operator).

EXAMPLE

To remove all files named **a.out** or ***.o** that have not been accessed for a week:

find / \(–name a.out –o –name '*.o' \) –atime +7 –exec rm {} \;

FILES

/etc/passwd, /etc/group

DIAGNOSTICS

find(1) will output one of the following error messages if it finds an incorrect entry in the **/etc/group** file.

fgetgrent: too many users in group < *group*>
fgetgrent: line exceeds system limit (*limit*)

where:

group
is the invalid entry

limit
is the system limit for the length of a line.

The output of one of these messages does not result in the failure of the command.

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

finger – user information lookup program

SYNOPSIS

finger [options] name ...

DESCRIPTION

By default *finger* lists the login name, full name, terminal name and write status (as a '*' before the terminal name if write permission is denied), idle time, login time, and office location and phone number (if they are known) for each current UNIX user. (Idle time is minutes if it is a single integer, hours and minutes if a ':' is present, or days and hours if a 'd' is present.)

A longer format also exists and is used by *finger* whenever a list of people's names is given. (Account names as well as first and last names of users are accepted.) This format is multi-line, and includes all the information described above as well as the user's home directory and login shell, any plan which the person has placed in the file *.plan* in their home directory, and the project on which they are working from the file *.project* also in the home directory.

Finger may be used to lookup users on a remote machine. The format is to specify the user as "user@host." If the user name is left off, the standard format listing is provided on the remote machine.

Finger options include:

–m Match arguments only on user name.

–l Force long output format.

–p Suppress printing of the *.plan* files

–s Force short output format.

FILES

/etc/utmp	who file
/etc/passwd	for users names, offices, ...
~/.plan	plans
~/.project	projects

SEE ALSO

fingerd(1M), who(1).

AUTHOR

Earl T. Cohen

BUGS

Only the first line of the *.project* file is printed.

The encoding of the gcos field is UCB dependent – it knows that an office '197MC' is '197M Cory Hall', and that '529BE' is '529B Evans Hall'. It also knows that a four digit office phone number should have a "x2-" prepended.

There is no way to pass arguments to the remote machine as *finger* uses an internet standard port.

A user information data base is in the works and will radically alter the way the information that *finger* uses is stored. Finger will require extensive modification when this is implemented.

NAME

fmt – disk initializer

SYNOPSIS

fmt [*options*] *alias*

DESCRIPTION

fmt(1) checks to see that *alias* is in the **permissions** file and that format permission is given. *fmt* takes the same options as *dinit*(1M). Options are added to those found in the *format_pgm* field of the appropriate **permissions** file entry. The *format_pgm* is then executed with these options on slice 7 of the raw device corresponding to the *slice* entry in the **permissions** file.

If no *alias* is specified, the **floppy** alias is assumed.

FILES

/etc/ddefs	
/etc/dskdefs/*	disk definition files
/etc/dinit	
/etc/filesys	permissions file

SEE ALSO

dinit(1M), filesys(4).

NAME

fold – fold long lines for finite width output device

SYNOPSIS

fold [–width] [file ...]

DESCRIPTION

Fold is a filter which will fold the contents of the specified files, or the standard input if no files are specified, breaking the lines to have maximum width *width*. The default for *width* is 80. *Width* should be a multiple of 8 if tabs are present.

BUGS

If underlining is present it may be messed up by folding.

NAME

from – who is my mail from?

SYNOPSIS

from [–s sender] [user]

DESCRIPTION

From prints out the mail header lines in your mailbox file to show you who your mail is from. If *user* is specified, then *user*'s mailbox is examined instead of your own. If the -s option is given, then only headers for mail sent by *sender* are printed.

FILES

/usr/mail/*

SEE ALSO

biff(1B), mail(1)

NAME

fs – construct a file system

SYNOPSIS

fs [*disk* [*blocks*[:*inodes*]]]

DESCRIPTION

fs(1) builds a file system with a single empty directory on it. The argument *disk* and the **permissions** file are used to determine the device to build a file system on. This device will be the first match of *disk* and the *slice* or *alias* entries in the **permissions** file. If the *disk* argument is not given, then the first *alias* of **floppy** in the **permissions** file will be used. *fs* actually uses the raw version of the listed *slice* (by prepending an **r** to the name).

The size of the file system is the value of *blocks* interpreted as a decimal number. This is the number of *physical* disk blocks the file system occupies. This value may not be larger than the default value specified in the **permissions** file. If the number of blocks is not specified, the default value in the **permissions** file is used. The boot program is left uninitialized. If the optional number of inodes is not given, the default is the number of logical blocks divided by four.

FILES

/etc/mkfs
/etc/filesys permissions file

SEE ALSO

mkfs(1M), dir(4), fs(4), filesys(4).

NAME

ftp – ARPANET file transfer program

SYNOPSIS

ftp [–v] [–n] [–d] [–i] [–g] [*host*]

DESCRIPTION

ftp is the user interface to the ARPANET standard File Transfer Protocol. The program allows a user to transfer files to and from a remote network site.

The client host with which *ftp* is to communicate may be specified on the command line. If this is done, *ftp* immediately attempts to establish a connection to an FTP server on that host; otherwise, *ftp* enters its command interpreter and await instructions from the user. When *ftp* is awaiting commands from the user, the prompt "**ftp>** " is provided to the user. The following commands are recognized by *ftp*:

! [*command* [*args*]]
Invoke an interactive shell on the local machine. If there are arguments, the first is taken to be a command to execute directly, with the rest of the arguments as its arguments.

append *local-file* [*remote-file*]
Append a local file to a file on the remote machine. If *remote-file* is left unspecified, the local filename is used in naming the remote file. File transfer uses the current settings for *type*, *mode*, and *struct* (structure).

bell
Toggle to sound a bell after each file transfer command is completed. The default is bell off.

bye
Terminate the FTP session with the remote server and exit *ftp*. An EOF also terminates the session and exits.

cd *remote-directory*
Change the working directory on the remote machine to *remote-directory*.

close
Terminate the FTP session with the remote server, and return to the command interpreter. *close* and *disconnect* are synonymous.

connect
A synonym for *open*.

delete *remote-file*
Delete the file *remote-file* on the remote machine.

debug [*debug-value*]
Toggle debugging mode. If an optional *debug-value* is specified, it is used to set the debugging level. When debugging is on, *ftp* prints each command sent to the remote machine, preceded by the string "**-->** ". The default debug mode is off.

dir [*remote-directory*] [*local-file*]
Print a listing of the directory contents in the directory, *remote-directory*, and, optionally, placing the output in *local-file*. If no directory is specified, the current working directory on the remote machine is used. If no local file is specified, or *local-file* is -, output comes to the terminal.

disconnect
A synonym for *close*.

get *remote-file* [*local-file*]
Retrieve the *remote-file* and store it on the local machine. If the local filename is not specified, it is given the same name it has on the remote machine. The current settings for *type*, *mode*, and *struct* are used while transferring the file.

glob
Toggle filename expansion for *mdelete*, *mget*, and *mput*. If globbing is turned off with *glob*, the filename arguments are taken literally and not expanded. Globbing for *mput* is done. For *mdelete* and *mget*, each remote filename is expanded separately on the remote machine and the lists are not merged. Expansion of a directory name is likely to be different from expansion of the name of an ordinary file: the exact result depends on the foreign operating system and ftp server, and can be previewed by issuing:

mls *remote-files* - *mget* and *mput* are not meant to transfer entire directory subtrees of files. That can be done by transferring a *tar*(1) archive of the subtree (in binary mode).

The default is globbing turned on.

hash
Toggle hash-sign (#) printing for each data block transferred. The size of a data block is 1024 bytes. The default is hash-sign printing turned off.

help [*command*]
Print an informative message about the meaning of *command*. If no argument is given, *ftp* prints a list of the known commands.

lcd [*directory*]
Change the working directory on the local machine. If no *directory* is specified, the user's home directory is used.

ls[*-options*] [*remote-directory*] [*local-file*]
Print an abbreviated listing of the contents of a directory on the remote machine. *-options* are the options that *ls*(1) receives with no white spaces between them. When options are used, the current working directory is used. If a *remote-directory* and *-options* are both needed, they must be quoted as if all were part of the options (i.e., **ls "-options remote_directory"**).

If *remote-directory* is left unspecified, the current working directory is used. If no *local_file* is specified, or if *local_file* is -, the output is sent to the terminal. If you want a simple *ls* of a current directory with output to go to a *local_file*, the current directory must be specified with a dot (.) and the *local_file* (i.e., **ls . local_file**), otherwise, the *local_file* is interpreted as being the *remote_directory*.

mdelete [*remote-files*]
Delete the *remote-files* on the remote machine.

mdir *remote-files local-file*
Like *dir*, except multiple remote files may be specified. If interactive prompting is on, *ftp* prompts the user to verify that the last argument is indeed the target local file for receiving *mdir* output.

mget *remote-files*
Expand the *remote-files* on the remote machine and do a *get* for each filename thus produced. See *glob* for details on the filename expansion. Files are transferred into the local working directory, which can be changed with "**lcd** *directory*"; new local directories can be created with "**! mkdir** *directory*".

mkdir *directory-name*
Make a directory on the remote machine.

mls *remote-files local-file*
Like *ls*, except multiple remote files may be specified. If interactive prompting is on, *ftp* prompts the user to verify that the last argument is indeed the target local file for receiving *mls* output.

mode [*mode-name*]
Set the file transfer *mode* to *mode-name*. The default mode is "stream" mode.

mput *local-files*
Expand wild cards in the list of local files given as arguments and do a *put* for each file in the resulting list. See *glob* for details of filename expansion.

open *host* [*port*]
Establish a connection to the specified *host* FTP server. An optional port number may be supplied, in which case, *ftp* attempts to contact an FTP server at that port.

prompt
Toggle interactive prompting. Interactive prompting occurs during multiple file transfers to allow the user selectively to retrieve or store files. If prompting is turned off (default is off), any *mget* or *mput* transfers all files, and any *mdelete* deletes all files.

put *local-file* [*remote-file*]
Store a local file on the remote machine. If *remote-file* is left unspecified, the local filename is used. File transfer uses the current settings for *type*, *mode*, and *struct* (structure).

pwd
Print the name of the current working directory on the remote machine.

quit
A synonym for *bye*.

quote *arg1 arg2 ...*
The arguments specified are sent, verbatim, to the remote FTP server.

recv *remote-file* [*local-file*]
A synonym for *get*.

remotehelp [*command-name*]
Request help from the remote FTP server. If a *command-name* is specified, it is supplied to the server as well.

rename [*from*] [*to*]
Rename the file *from* on the remote machine, to the file *to*.

rmdir *directory-name*
Delete a directory on the remote machine.

send *local-file* [*remote-file*]
A synonym for *put*.

sendport
Toggle the use of **PORT** commands. By default, *ftp* attempts to use a **PORT** command when establishing a connection for each data transfer. The use of **PORT** commands can prevent delays when performing multiple file transfers. If the **PORT** command fails, *ftp* uses the default data port. When the use of **PORT** commands is disabled, no attempt is made to use **PORT** commands for each data transfer. This is useful for certain FTP implementations which do ignore **PORT** commands but, incorrectly, indicate they have been accepted.

status
Show the current status of *ftp*.

struct [*struct-name*]
Set the file transfer *structure* to *struct-name*. By default, file structure is used.

type [*type-name*]
Set the file transfer *type* to *type-name*. If no type is specified, the current type is printed. The available types are:

ascii
Set the file transfer *type* to network ASCII. This is the default type.

binary
Set the file transfer *type* to support binary image transfer.

image
A synonym for the binary file transfer type.

ebcdic
Set the file transfer *type* to EBCDIC.

tenex
Set the file transfer *type* to that needed to talk to TENEX machines.

user *user-name* [*password*] [*account*]
Identify yourself to the remote FTP server. If the password is not specified, and the server requires it, *ftp* prompts the user for it (after disabling local echo). If an account field is not specified, and the FTP server requires it, the user is prompted for it. If an account field is specified, an account command is relayed to the remote server after the login sequence is completed if the remote server did not require it for logging in. This process is done automatically on initial connection to the FTP server.

verbose
Toggle verbose mode. In verbose mode, all responses from the FTP server are displayed to the user. In addition, if verbose is on, when a file transfer completes, statistics regarding the efficiency of the transfer are reported. By default, verbose is on.

? [*command*]
A synonym for *help*.

Command arguments which have embedded spaces may be quoted with quote (") marks.

ABORTING A FILE TRANSFER

To abort a file transfer, use the terminal interrupt key (usually). Sending transfers are halted immediately. Receiving transfers are halted by sending an *ftp* protocol **ABOR** command to the remote server, and discarding any further data received. The speed at which this is accomplished depends upon the remote server's support for **ABOR** processing. If the remote server does not support the **ABOR** command, an **ftp>** prompt does not appear until the remote server has completed sending the requested file.

The terminal interrupt key sequence is ignored when *ftp* has completed any local processing and is awaiting a reply from the remote server. A long delay in this mode may result from the **ABOR** processing described above, or from unexpected behavior by the remote server, including violations of the ftp protocol. If the delay results from unexpected remote server behavior, the local *ftp* program must be killed by hand.

FILE NAMING CONVENTIONS

Files specified as arguments to *ftp* commands are processed according to the following rules:

1. If the filename "–" is specified, the stdin (for reading) or stdout (for writing) is used.

2. If the first character of the filename is "|", the remainder of the argument is interpreted as a shell command. *ftp* then forks a shell, using *popen*(3) with the argument supplied, and reads (writes) from the stdout (stdin). If the shell command includes spaces, the argument must be quoted; for example:

   "|ls -lt"

   A particularly useful example of this mechanism is:

   dir . |pg

3. Failing the above checks, if "globbing" is enabled, local filenames are expanded. If the *ftp* command expects a single local file (e.g., *put*), only the first filename generated by the "globbing" operation is used.
4. For *mget* commands and *get* commands with unspecified local filenames, the local filename is the remote filename.
5. For *mput* commands and *put* commands with unspecified remote filenames, the remote filename is the local filename.

FILE TRANSFER PARAMETERS

The FTP specification includes many parameters that may affect a file transfer. The *type* may be one of **ascii**, **image** (**binary**), **ebcdic**, or **tenex**.

FLAG OPTIONS

Flag options may be specified at the command line, or to the command interpreter.

The **-v** (verbose on) flag option forces *ftp* to show all responses from the remote server, as well as report on data transfer statistics.

The **-n** flag option restrains *ftp* from attempting auto-login upon initial connection. If auto-login is enabled, *ftp* uses the login name on the local machine as the user identity on the remote machine (and prompts for confirmation/change), then prompts for a password and, optionally, an account with which to login.

The **-i** flag option turns off interactive prompting during multiple file transfers.

The **-d** flag option enables debugging.

The **-g** flag option disables filename globbing.

FILES

/usr/spool/ftp

SEE ALSO

cu(1C).
Using the Networking Enhancements in the *Network User's Guide.*
DDN Protocol Handbook, Volume 2, Section 8.1.3 (RFC 959)

BUGS

Correct execution of many commands depends upon proper behavior by the remote server.

An error in the treatment of returns in the 4.2 BSD UNIX ASCII-mode transfer code has been corrected. This correction may result in incorrect transfers of binary files to and from 4.2 BSD servers using the **ascii** type. You may avoid this problem by using the **image** (binary) type.

When the verbose mode **-v** flag option is turned off, *ftp* does not echo responses from the remote server. This includes the response to the request **pwd**. Beware of this.

NAME

gcc – GLS C compiler

SYNOPSIS

gcc [*options*] *files*

DESCRIPTION

The *gcc* command is the interface to the GLS C Compilation System. The compilation tools consist of a compiler and linker. By default, the GLS C compiler produces object code directly, but certain compile options will invoke an assembly step. The *gcc* command processes the supplied options, then executes the various tools with the proper arguments. The *gcc* command accepts several types of arguments:

Arguments whose names end with *.c* are C source programs; they are compiled and left in a *.o* file in the working directory.

Arguments whose names end with *.s* are assembly source programs; they are assembled and left in a *.o* file in the working directory.

Other arguments are taken to be link editor option arguments, C-compatible object programs, or object program libraries.

The *.o* file is deleted if a single source file is compiled and linked.

The *.s* files created for each module usually are deleted.

The following options are interpreted by *gcc*:

–c Do not produce executable files; produce only object files. For each source file specified, compile the source language file into object code output and place the output in a file whose name is the same as the input source file but which ends with *.o* rather than the suffix of the input source file.

–g Generate source level symbolic debugger information in the object file, for use with a debugger such as *sdb*(1).

–ga Generate a frame for stack traces. The default compiler setting is to optimize the program to the point that stack traces become impossible on some machines. This makes program debugging difficult. Specify this option when debugging a program. (This option does not imply option **–g**.)

–o *filename*
Place the executable (or, if **–c** is specified, object file) output into *filename*. If this option is not specified, *gcc* names the executable file *a.out*. If **–c** is specified and **–o** is not used, the object filename will be the same as the source filename with the suffix changed to *.o*. *gcc* displays an error message to *stderr* if *filename* is suffixed with one of the following: *.F*, *.f*, *.c*, *.i*, *.p*, or *.s*. This protects source files with those suffixes from being overwritten. This option is ignored if you specify **–S**, or if you specify **–c** with more than one source file.

–p, **–qp**, or **–pg**
Produce code that counts the number of times each routine is called. If link editing takes place, link profiled versions of *libc.a* and *libm.a* (with **–lm** option) and call *monitor*(3C). Produce a *mon.out* file at normal termination of execution of the object program. An execution profile can then be generated with *prof*(1). **–pg** produces a monitor output file for the Berkley profiler *gprof*(1).

–v Display the command line arguments on *stderr* during each step of the compilation process.

-w Suppress warning messages.

-ansi Compile programs substantially according to the ANSI C specification. This adds another directory to the search path for include files, as well as other libraries. Use the **-X316** switch if you merely want the language changes.

-ANSI This is similar to **-ansi** except full conformance is supported. In particular, floating-point arithmetic may be slower. Use the **-X153** switch if you merely want the language changes.

-noansi Compiles programs according to traditional UNIX C syntax and semantics.

-C By default, the preprocessing compile step strips C-style comments. Specifying **-C** will cause comments to be left in the preprocessor output. This option will be passed to the GLS C compiler because it has its own internal preprocessor. If **-P** has been specified, then this option will be passed to *cpp*(1).

-D*name*[=*string*]
If *string* is not specified, define *name* to *cpp*(1) with the value 1. If *string* is specified, *name* is defined to *cpp*(1) with the value of *string*. This is equivalent to placing the directive `#define` *name string* at the beginning of the source file.

-E Do not compile the program; instead, place the output of the preprocessor on the standard output file. This is useful for debugging preprocessor macros. The integrated preprocessor cannot generate output as fast as *cpp*(1), so use *cpp*(1) for big jobs.

-H Display the names of the files on *stderr* as they are opened. This is useful for finding out why the compiler cannot find an include file. This option invokes **-X32** and passes it to the compiler.

-I*dir* Search for include file names that are not absolute (do not start with /) in directory *dir* before a standard list of directories. The standard list of directories searched is your current directory, then the */usr/include* directory. To search more than one directory, specify **-I***dir* for each directory.

-O Activate the following optimizers: static address elimination, loop invariant analysis, strength reduction, and peephole. These optimizations are safe for programs that are not loop bound. Routines with more than a thousand lines of code may require large amounts of memory to compile. If memory constraints are a concern, do not use the **-O** compile option, and break large routines into several smaller routines.

-OA Apply algorithmic changes to the program. Currently, this will change the behavior during full ANSI C compiles to maintain floating-point results in floating-point registers.

-OL Optimize the program to be as fast as possible, even if this requires making the program larger. Most of the available resources are allocated to optimizations of the innermost loops. **-OL** causes scalar multiplies to be performed in-line and performs larger block moves sequentially. It is counterproductive to specify **-OL** on code that contains no loops or that is rarely executed, because it will make the whole program larger but not faster.

-OM Allow the optimizer to assume that memory locations do not change except by explicit stores. No memory locations referenced by the compiled program are I/O device registers. These memory locations cannot be changed by external hardware or concurrent processes. **-OM** must be used with caution with device drivers, operating systems, shared memory environments, and interrupts or signals.

-OMLA This option is equivalent to **-OM** and **-OL** and **-OA**. The *M*, the *L*, and the *A* may be specified in any order, or one or more of them may be omitted to disable the corresponding optimization option.

-P Run only *cpp*(1) on the named C programs and store the result in the corresponding files that end with *.i*.

-S Compile the named programs and leave the assembly language output on corresponding files suffixed with *.s*.

-T Truncate symbols/defines to 8 characters. This is also a preprocessor *cpp*(1) option.

-U*name* Undefine the predefined preprocessor symbol *name* for this compilation. This option is equivalent to putting **#undef** *name* at the beginning of the source file.

-V Display the version of the link editor and/or assembler (if it is invoked).

-W*c,arg1*[,*arg2*]...
Pass argument[s] *argi* to pass *c* where *c* is one of [p0al] interpreted as for the **-Y** option. For example: **-Wa,-m** passes **-m** to the assembler.

-X*nnn* Set option *nnn* (where *nnn* is an unsigned integer constant). To reset an option set with **-X** *nnn*, use **-Z** *nnn*. Not all **-X** options are listed in the man pages. A complete list can be found in the *GLS Programming Guide for 97xx Systems*.

-X18 Only declared *register* variables are allocated in registers.

-X168 Do not move invariant floating-point expressions out of loops.

-X153 Support the full ANSI language in the compiler, without using the ANSI libraries or include files.

-Y[*p0alSILU*],*dirname*
Specify a new pathname, *dirname*, for the locations of the tools and directories designated in the first argument. [*p0alSILU*] represents:

- *p* preprocessor *cpp*(1); applicable only when using the **-P** option because GLS C has its own internal preprocessor
- *0* GLS C compiler
- *a* assembler
- *l* link editor
- *S* directory containing the startup routines
- *I* default include directory searched for by *cpp*(1)
- *L* first default library directory searched for by *ld*(1)
- *U* second default library directory searched for by *ld*(1)

-Z*nnn* Turn off option number *nnn*. This is the reverse of the **-X** option. This option is useful for turning off options that are on by default.

FILES

file.c	C source file
file.o	object file
file.s	assembly language file
a.out	link edited output (executable)
mon.out	profile output for use by *prof*(1)
/usr/bin /gcc	driver for compiler
/usr/lib/ccom88	compiler
/lib/crt0.o	startup routine
/lib/mcrt0.o	profiling startup routine
/lib/libm.a	standard math library
/lib/libc.a	standard C library

/usr/include standard directory for #include files

SEE ALSO

B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Prentice-Hall, 1978
B. W. Kernighan, *Programming in C—A tutorial*
D. M. Ritchie, *C Reference Manual*
GLS Programming Guide for 97xx Systems
as(1), prof(1), sdb(1), ld(1).

DIAGNOSTICS

The diagnostics produced by the GLS GCC compiler are intended to be self-explanatory and are similar to those produced by other C compilers. Occasional messages may be produced by the assembler or link editor.

BUGS

Routines with more than 1,000 lines of code may require large amounts of memory to compile. If memory constraints are a problem, do not use the **-O** option.

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 path name does not begin with **s.**) and unreadable files are silently ignored. If a name of – is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The generated text is normally written into a file called the *g-file* whose name is derived from the SCCS file name by simply removing the leading **s.**; (see **FILES**, below).

Each of the keyletter arguments is explained below as though only one SCCS file is to be processed, but the effects of any keyletter argument applies independently to each named file.

–r*SID*

The *S*CCS *ID*entification 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 that 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, i.e., **–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 you to specify a *cutoff* date in the form: "**–c77/2/2 9:22:25**". Note that this implies that one may use the %E% and %U% identification keywords (see below) for nested *gets* within, say the input to a *send*(1C) command:

~!get "–c%E% %U%" s.file

–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 *gets* for editing on the same SID until *delta* is executed or the

j (joint edit) flag is set in the SCCS file (see *admin*(1)). Concurrent use of **get** **–e** for different SIDs is always allowed.

If the *g-file* generated by *get* with an **–e** keyletter is accidentally ruined in the process of editing it, it may be regenerated by re-executing the *get* command with the **–k** keyletter in place of the **–e** keyletter.

SCCS file protection specified via the ceiling, floor, and authorized user list stored in the SCCS file (see *admin*(1)) are enforced when the **–e** keyletter is used.

–b

Used with the **–e** keyletter to indicate that the new delta should have an SID in a new branch as shown in Table 1. This keyletter is ignored if the **b** flag is not present in the file (see *admin*(1)) or if the retrieved *delta* is not a leaf *delta*. (A leaf *delta* is one that has no successors on the SCCS file tree.)

NOTE: A branch *delta* may always be created from a non-leaf *delta*. Partial SIDs are interpreted as shown in the "SID Retrieved" column of Table 1.

–k

Suppresses replacement of identification keywords (see **–s**) 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 (e.g., **–r1**), or release and level (e.g., **–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*(5)). This keyletter is used by the *comb*(1) command; it is not a generally useful keyletter. If both the **–r** and **–a** keyletters are specified, only the **–a** keyletter is used. Care should be taken when using the **–a** keyletter with the **–e** keyletter, because the SID of the delta to be created may not be what you expects. The **–r** keyletter can be used with the **–a** and **–e** keyletters to control the naming of the SID of the delta to be created.

For each file processed, *get* responds (on the standard output) with the SID being accessed and with the number of lines retrieved from the SCCS file.

If the **–e** keyletter is used, the SID of the delta to be made appears after the SID accessed and before the number of lines generated. If there is more than one named file or if a directory or standard input is named, each file name is printed (preceded by a newline) 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.# in release > R and R exists	R.mL	R.mL.(mB+1).1
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". Thus, for example, "R.mL" means "the maximum level number within release R"; "R.L.(mB+1).1" means "the first sequence number on the *new* branch (i.e., maximum branch number plus one) of level L within release R". Note that if the SID specified is of the form "R.L", "R.L.B", or "R.L.B.S", each of the specified components *must* exist.

** "hR" is the highest *existing* release that is lower than the specified, *nonexistent*, release R.

*** This is used to force creation of the *first* delta in a *new* release.

\# Successor.

† The **–b** keyletter is effective only if the **b** flag (see *admin*(1)) is present in the file. An entry of – means "irrelevant".

‡ This case applies if the **d** (default SID) flag is *not* present in the file. If the **d** flag *is* present in the file, then the SID obtained from the **d** flag is interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies.

IDENTIFICATION KEYWORDS

Identifying information is inserted into the text retrieved from the SCCS file by replacing *identification keywords* with their value wherever they occur. The following keywords may be used in the text stored in an SCCS file:

Keyword	Value
%M%	Module name: either the value of the **m** flag in the file (see *admin*(1)), or if absent, the name of the SCCS file with the leading **s.** removed.
%I%	SCCS identification (SID) (%R%.%L%.%B%.%S%) of the retrieved text.
%R%	Release.
%L%	Level.
%B%	Branch.
%S%	Sequence.
%D%	Current date (YY/MM/DD).
%H%	Current date (MM/DD/YY).
%T%	Current time (HH:MM:SS).
%E%	Date newest applied delta was created (YY/MM/DD).
%G%	Date newest applied delta was created (MM/DD/YY).
%U%	Time newest applied delta was created (HH:MM:SS).
%Y%	Module type: value of the **t** flag in the SCCS file
%F%	SCCS file name.
%P%	Fully qualified SCCS file name.
%Q%	The value of the **q** flag in the file (see *admin*(1)).
%C%	Current line number. This keyword is intended for identifying messages output by the program such as "this should not have happened" type errors. It is *not* intended to be used on every line to provide sequence numbers.
%Z%	The 4-character string @(#) recognizable by *what*(1).
%W%	A shorthand notation for constructing *what*(1) strings for UNIX program files. %W%#=#%Z%%M%<horizontal-tab>%I%
%A%	Another shorthand notation for constructing *what*(1) strings for non-UNIX program files. %A% = %Z%%Y% %M% %I%%Z%

Several auxiliary files may be created by *get*. These files are known generically as the **g-file**, **l-file**, **p-file**, and **z-file**. The letter before the hyphen is called the tag. An auxiliary file name is formed from the SCCS file name: the last component of all SCCS file names must be of the form **s.***module-name*, the auxiliary files are named by replacing the leading **s** with the tag. The **g-file** is an exception to this scheme: the **g-file** is named by removing the **s.** prefix. For example, **s.xyz.c**, the auxiliary file names would be **xyz.c**, **l.xyz.c**, **p.xyz.c**, and **z.xyz.c**, respectively.

The **g-file**, which contains the generated text, is created in the current directory (unless the **–p** keyletter is used). A **g-file** is created in all cases, whether 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; other *.

b. A blank character if the delta was applied or was not applied and ignored; * if the delta was not applied and was not ignored.

c. A code indicating a "special" reason why the delta was or was not applied:
"I": Included.
"X": Excluded.
"C": Cut off (by a **-c** keyletter).

d. Blank.

e. SCCS identification (SID).

f. Tab character.

g. Date and time (in the form YY/MM/DD HH:MM:SS) of creation.

h. Blank.

i. Login name of person who created *delta*.

The comments and MR data follow on subsequent lines, indented one horizontal tab character. A blank line terminates each entry.

The **p-file** is used to pass information resulting from a *get* with an **-e** keyletter along to *delta*. Its contents are also used to prevent a subsequent execution of *get* with an **-e** keyletter for the same SID until *delta* is executed or the joint edit flag, **j**, (see *admin*(1)) is set in the SCCS file. The *p-file* is created in the directory containing the SCCS file and the effective user must have write permission in that directory. Its mode is 644 and it is owned by the effective user. The format of the **p-file** is: the gotten SID, followed by a blank, followed by the SID that the new delta will have when it is made, followed by a blank, followed by the login name of the real user, followed by a blank, followed by the date-time the *get* was executed, followed by a blank and the **-i** keyletter argument if it was present, followed by a blank and the **-x** keyletter argument if it was present, followed by a 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

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.

EXAMPLE

The following code fragment shows how you can 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 will accept any of the following as equivalent:

```
cmd -aoarg file file
cmd -a -o arg file file
cmd -oarg -a file file
cmd -a -oarg — file file
```

SEE ALSO

getopts(1), sh(1), getopt(3C).

DIAGNOSTICS

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 has been provided, *getoptcvt*. For more information about *getopts* and *getoptcvt*, see the *getopts*(1) manual page.

If an option that takes an option-argument is followed by a value that is the same as one of the options listed in *optstring* (referring to the earlier **EXAMPLE** section, but using the following command line:

cmd -o -a file

getopt will always treat **-a** as an option-argument to **-o**; it will never recognize **-a** as an option. For this case, the **for** loop in the example will shift past the *file* argument.

NAME

getopts, getoptcvt – parse command options

SYNOPSIS

getopts *optstring name* [*arg* ...]

/usr/lib/getoptcvt [–b] *file*

DESCRIPTION

getopts is used by shell procedures to parse positional parameters and to check for legal options. It supports all applicable rules of the command syntax standard (see Rules 3-10, *intro*(1)). It should be used in place of the *getopt*(1) command (see **WARNING**).

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 UNIX system. **/usr/lib/getoptcvt** modifies the shell script in *file* so that when the resulting shell script is executed, it determines at run time whether to invoke *getopts*(1) or *getopt*(1).

So all new commands will 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**).

EXAMPLE

The following fragment of a shell program shows how you can 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 will accept any of the following as equivalent:

```
cmd –a –b –o "xxx z yy" file
cmd –a –b –o "xxx z yy" — file
cmd –ab –o xxx,z,yy file
cmd –ab –o "xxx z yy" file
cmd –o xxx,z,yy –b –a file
```

SEE ALSO

intro(1), sh(1), getopts(3C),

WARNING

Although the following command syntax rule (see *intro*(1)) relaxations are permitted under the current implementation, they should not be used because they may not be supported in future releases of the system. As in the **EXAMPLE** section, **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

glossary – definitions of terms and symbols

SYNOPSIS

[**help**] **glossary** [*term*]

DESCRIPTION

The Help Facility command *glossary* 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 may choose one of the terms or may 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 be displayed. Some of the symbols must be escaped if requested at shell level in order for the facility to understand the symbol. The following is a table that lists the symbols and their escape sequence.

Symbol	Escape Sequence
""	\"\"
''	\'\'
[]	\\[\\]
` `	\`\`
#	\#
&	\&
*	*
\	\\\\
|	\\|

From any screen in the Help Facility, a user may execute a command via the shell (*sh*(1)) by typing a ! and the command to be executed. The screen will be redrawn if the command that was executed 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

help(1), locate(1), sh(1), starter(1), usage(1), helpadm(1M), term(5).

WARNINGS

If the shell variable TERM (see *sh*(1)) 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

gprof – display call graph profile data

SYNOPSIS

gprof [options] [a.out [gmon.out ...]]

DESCRIPTION

gprof produces an execution profile of C programs. The effect of called routines is incorporated in the profile of each caller. The profile data is taken from the call graph profile file (*gmon.out* default) which is created by programs which are compiled with the **–qg** option of *cc*. That option also links in versions of the library routines which are compiled for profiling. The symbol table in the named object file (*a.out* default) is read and correlated with the call graph profile file. If more than one profile file is specified, the *gprof* output shows the sum of the profile information in the given profile files.

First, a flat profile is given, similar to that provided by *prof*(1). This listing gives the total execution times and call counts for each of the functions in the program, sorted by decreasing time.

Next, these times are propagated along the edges of the call graph. Cycles are discovered, and calls into a cycle are made to share the time of the cycle. A second listing shows the functions sorted according to the time they represent including the time of their call graph descendents. Below each function entry is shown its (direct) call graph children, and how their times are propagated to this function. A similar display above the function shows how this function's time and the time of its descendents is propagated to its (direct) call graph parents.

Cycles are also shown, with an entry for the cycle as a whole and a listing of the members of the cycle and their contributions to the time and call counts of the cycle.

The following options are available:

–a suppresses the printing of statically declared functions. If this option is given, all relevant information about the static function (*e.g.,* time samples, calls to other functions, calls from other functions) belongs to the function loaded just before the static function in the *a.out* file.

–b suppresses the printing of a description of each field in the profile.

–c the static call graph of the program is discovered by a heuristic which examines the text space of the object file. Static-only parents or children are indicated with call counts of 0. This option is accepted but does nothing in REAL/IX.

–e *name*
suppresses the printing of the graph profile entry for routine *name* and all its descendants (unless they have other ancestors that aren't suppressed). More than one **–e** option may be given. Only one *name* may be given with each **–e** option.

–E *name*
suppresses the printing of the graph profile entry for routine *name* (and its descendants) as **–e**, above, and also excludes the time spent in *name* (and its descendants) from the total and percentage time computations. (For example, **–E** *mcount* **–E** *mcleanup* is the default.)

–f *name*
prints the graph profile entry of only the specified routine *name* and its descendants. More than one **–f** option may be given. Only one *name* may be given with each **–f** option.

-F *name*

prints the graph profile entry of only the routine *name* and its descendants (as **-f**, above) and also uses only the times of the printed routines in total time and percentage computations. More than one **-F** option may be given. Only one *name* may be given with each **-F** option. The **-F** option overrides the **-E** option.

-s a profile file *gmon.sum* is produced which represents the sum of the profile information in all the specified profile files. This summary profile file may be given to subsequent executions of gprof (probably also with a **-s**) to accumulate profile data across several runs of an *a.out* file.

-z displays routines which have zero usage (as indicated by call counts and accumulated time). This is useful in conjunction with the **-c** option for discovering which routines were never called.

FILES

a.out	the namelist and text space.
gmon.out	dynamic call graph and profile.
gmon.sum	summarized dynamic call graph and profile.

SEE ALSO

monitor(3), profil(2), cc(1), prof(1)
"gprof: A Call Graph Execution Profiler", by Graham, S.L., Kessler, P.B., McKusick, M.K.; *Proceedings of the SIGPLAN '82 Symposium on Compiler Construction*, SIGPLAN Notices, Vol. 17, No. 6, pp. 120-126, June 1982.

BUGS

Beware of quantization errors. The granularity of the sampling is shown, but remains statistical at best. We assume that the time for each execution of a function can be expressed by the total time for the function divided by the number of times the function is called. Thus the time propagated along the call graph arcs to parents of that function is directly proportional to the number of times that arc is traversed.

Parents which are not themselves profiled will have the time of their profiled children propagated to them, but they will appear to be spontaneously invoked in the call graph listing, and will not have their time propagated further. Similarly, signal catchers, even though profiled, will appear to be spontaneous (although for more obscure reasons). Any profiled children of signal catchers should have their times propagated properly, unless the signal catcher was invoked during the execution of the profiling routine, in which case all is lost.

The profiled program must call *exit*(2) or return normally for the profiling information to be saved in the gmon.out file.

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 *terminals* 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).
eqn(1), mm(1), nroff(1) in the *DOCUMENTER'S WORKBENCH Software Release 2.0 Technical Discussion and Reference Manual.*
environ(5), greek(5), term(5).

NAME

grep – search a file for a pattern

SYNOPSIS

grep [*options*] *limited regular expression* [*file* ...]

DESCRIPTION

grep searches files for a pattern and prints all lines that contain that pattern. *grep* uses limited regular expressions (expressions that have string values that use a subset of the possible alphanumeric and special characters) like those used with *ed*(1) to match the patterns. It uses a compact non-deterministic algorithm.

Be careful using the characters $, *, [, ^, |, (,), and \ in the *limited regular expression* because they are also meaningful to the shell. It is safest to enclose the entire *limited regular expression* in single quotes ´...´.

If no files are specified, *grep* assumes standard input. Normally, each line found is copied to standard output. The file name is printed before each line found if there is more than one input file.

Command line options are:

–b
Precede each line by the block number on which it was found. This can be useful in locating block numbers by context (first block is 0).

–c
Print only a count of the lines that contain the pattern.

–i
Ignore upper/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

ed(1), egrep(1), fgrep(1), sed(1), sh(1).

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files (even if matches were found).

BUGS

Lines are limited to BUFSIZ characters; longer lines are truncated. BUFSIZ is defined in **/usr/include/stdio.h**.

If there is a line with embedded **NULL**s, *grep* will only match up to the first **NULL**; if it matches, it will print the entire line.

NAME

head – give first few lines

SYNOPSIS

head [–count] [file ...]

DESCRIPTION

This filter gives the first *count* lines of each of the specified files, or of the standard input. If *count* is omitted it defaults to 10.

SEE ALSO

tail(1).

NAME

help – 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 users who desire general information or specific assistance with the 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:

Command	Description
starter	information about the system for the beginning user
locate	locate commands using function-related keywords
usage	command usage information
glossary	definitions of technical terms

The user may choose one of the above commands by entering its corresponding letter (given in the menu), or exit to the shell by typing **q** (for "quit").

With arguments, *help* directly invokes the named online 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 how many backslashes to use for each, refer to the *glossary*(1) manual page.

From any screen in the Help Facility, a user may execute a command via the shell (*sh*(1)) by typing a **!** and the command to be executed. The screen will be redrawn if the command that was executed 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 may be either message numbers (which normally appear in parentheses following messages) or command names, of one 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 (e.g., ge3 for message 3 from the *get* command).

type2
Does not contain numerics (as a command, e.g., *get*).

type3
Is all numeric (e.g., 212).

SEE ALSO

glossary(1), helpadm(1M), locate(1), sh(1), starter(1), usage(1)
admin(1), cdc(1), comb(1), delta(1), get(1), prs(1), rmdel(1), sact(1), sccsdiff(1), unget(1), val(1), vc(1), what(1), profile(4), sccsfile(4), term(5).

WARNINGS

If the shell variable TERM (see *sh*(1)) is not set in the user's environment (see *environment*(5)) file, TERM defaults to the terminal value type 450 (a hard-copy terminal). For a list of valid terminal types, refer to *term*(5).

NAME

hostname – set or print name of current host system

SYNOPSIS

hostname [*nameofhost*]

DESCRIPTION

The *hostname* command prints the name of the current host, as given before the "login" prompt. The superuser can set the hostname by giving an argument; this is usually done in a startup script in **/etc/rc2.d**.

SEE ALSO

gethostname(2N), sethostname(2N)

NAME

ipcrm – remove a message queue, semaphore set, or shared memory ID

SYNOPSIS

ipcrm [*options*]

DESCRIPTION

ipcrm will remove one or more specified messages, semaphore or shared memory identifiers. The identifiers are specified by the following *options*:

–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.

Any of these options can specify one or more keys or identifiers. All arguments must be separated by commas or dashes. If specified in this manner, no error messages will be displayed.

For example:

ipcrm -s 3,6-9,15,19-22

will remove semaphore identifiers 3,6,7,8,9,15,19,20,21,22.

Arguments can also be given in C-style syntax:

ipcrm -s 3,6-9,15,0x13-0x16

The details of the removes are described in ***msgctl***(2), ***shmctl***(2), and ***semctl***(2). The identifiers and keys may be found by using *ipcs*(1).

SEE ALSO

ipcs(1), msgctl(2), msgget(2), msgop(2), semctl(2), semget(2), semop(2), shmctl(2), shmget(2), shmop(2).

IPCS(1)

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, semaphores, and binary semaphores that are currently active in the system. Otherwise, the information that is displayed is controlled by the following *options*:

–q Print information about active message queues.

–m Print information about active shared memory segments.

–s Print information about active semaphores.

–i Print information about active binary semaphores.

If any of the options **–q**, **–m**, **–s**, or **–i** are specified, information about only those indicated will be printed. If none of these four are specified, information about all four will be printed subject to these options:

–b Print biggest allowable size information. (Maximum number of bytes in messages on queue for message queues, size of segments for shared memory, and number of semaphores in each set for semaphores.) See below for meaning of columns in a listing.

–c Print creator's login name and group name. See below.

–o Print information on outstanding usage. (Number of messages on queue and total number of bytes in messages on queue for message queues and number of processes attached to shared memory segments.)

–p Print process number information. (Process ID of last process to send a message and process ID of last process to receive a message on message queues and process ID of creating process and process ID of last process to attach or detach on shared memory segments) See below.

–t Print time information. (Time of the last control operation that changed the access permissions for all facilities. Time of last *msgsnd* and last *msgrcv* on message queues, last *shmat* and last *shmdt* on shared memory, last *semop*(2) on semaphores.) See below.

–a Use all print *options*. (This is a shorthand notation for **–b**, **–c**, **–o**, **–p**, and **–t**.)

–C *corefile*

Use the file *corefile* in place of **/dev/kmem**.

–N *namelist*

The argument will be taken as the name of an alternate *namelist* (**/realix** is the default).

The column headings and the meaning of the columns in an *ipcs* listing are given below; the letters in parentheses indicate the *options* that cause the corresponding heading to appear; **all** means that the heading always appears. Note that these *options* only determine what information is provided for each facility; they do *not* determine which facilities will be listed.

T (all) Type of the facility:

q message queue;

m shared memory segment;

s semaphore;
b binary semaphore.

ID (all) The identifier for the facility entry.

KEY (all) The key used as an argument to *msgget*, *semget*, *shmget*, or *bsget* 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 follows:

The first two characters are:

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 if read permission is granted;
w if write permission is granted;
a if alter permission is granted;
– if the indicated permission is *not* granted.

OWNER (all) The login name of the owner of the facility entry.

GROUP (all) The group name of the group of the owner of the facility entry.

CREATOR (a,c) The login name of the creator of the facility entry.

CGROUP (a,c) The group name of the group of the creator of the facility entry.

CBYTES (a,o) The number of bytes in messages currently outstanding on the associated message queue.

QNUM (a,o) The number of messages currently outstanding on the associated message queue.

QBYTES (a,b) The maximum number of bytes allowed in messages outstanding on the associated message queue.

LSPID (a,p) The process ID of the last process to send a message to the 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.

IPCS(1)

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

/realix system namelist
/dev/kmem memory
/etc/passwd user names
/etc/group group names

SEE ALSO

msgop(2), semop(2), shmop(2).

BUGS

Things can change while *ipcs* is running; the picture it gives is only a close approximation of reality.

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 options below 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.

EXAMPLE

The following command line will join the password file and the group file, matching on the numeric group ID, and outputting the login name, the group name and the login directory. It is assumed that the files have been sorted in ASCII collating sequence on the group ID fields:

join –j1 4 –j2 3 –o 1.1 2.1 1.6 –t: /etc/passwd /etc/group

SEE ALSO

awk(1), comm(1), sort(1), uniq(1)

BUGS

With default field separation, the collating sequence is that of **sort –b**; with **–t**, the sequence is that of a plain sort.

The conventions of *join*, *sort*, *comm*, *uniq* and *awk*(1) are wildly incongruous. File names that are numeric may cause conflict when the **-o** option is used right before listing file names.

KILL(1)

NAME

kill – terminate a process

SYNOPSIS

kill [– *signo*] *PID* ...

DESCRIPTION

kill sends signal 15 (terminate) to the specified processes. This will normally kill processes that do not catch or ignore the signal. The process number of each asynchronous process started with & is reported by the shell (unless more than one process is started in a pipeline, in which case the number of the last process in the pipeline is reported). Process numbers can also be found by using *ps*(1).

The details of the kill are described in *kill*(2). For example, if process number 0 is specified, all processes in the process group are signaled.

The killed process must belong to the current user unless he is the super-user.

If a signal number preceded by – is given as first argument, that signal is sent instead of terminate (see *signal*(2)). In particular "kill –9 ..." is a sure kill.

SEE ALSO

ps(1), sh(1), kill(2), signal(2).

NAME

ld – link editor for common object files

SYNOPSIS

ld [*options*] *filename*

DESCRIPTION

The *ld* command combines several object files into one, performs relocation, resolves external symbols, and supports symbol table information for symbolic debugging. In the simplest case, the names of several object programs are given, and *ld* combines the objects, producing an object module that can either be executed or, if the **–r** option is specified, used as input for a subsequent *ld* run. The output of *ld* is left in **a.out**. By default, this file is executable if no errors occurred during the load and the **–r** option is not specified. If any input file, *filename*, is not an object file, *ld* assumes it is either an archive library or a text file containing link editor directives.

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. The library (archive) symbol table (see *ar*(4)) is searched sequentially with as many passes as are necessary to resolve external references which can be satisfied by library members. Thus, the ordering of library members is functionally unimportant, unless there exist multiple library members defining the same external symbol.

The following options are recognized by *ld*:

–e *epsym*
set the default entry point address for the output file to be that of the symbol **epsym**.

–f *fill*
set the default fill pattern for *holes* within an output section as well as initialized *bss* sections. The argument *fill* is a 2-byte constant.

–l*x*
search a library **lib***x***.a**, where *x* is up to nine characters. A library is searched when its name is encountered, so the placement of a **–l** is significant. By default, libraries are located in **LIBDIR** or **LLIBDIR**.

–m
produce a map or listing of the input/output sections on the standard output.

–o *outfile*
produce an output object file by the name **outfile**. The name of the default object file is **a.out**.

–r
retain relocation entries in the output object file. Relocation entries must be saved if the output file is to become an input file in a subsequent *ld* run. The link editor does not complain about unresolved references, and the output file is not executable.

–a
create an absolute file. This is the default if the **–r** option is not used. Used with the **–r** option, **–a** allocates memory for common symbols and generates complaints about unresolved references.

–s
strip line number entries and symbol table information from the output object file.

–t
turn off the warning about multiply-defined symbols that are not the same size.

-u *symname*
enter *symname* as an undefined symbol in the symbol table. This is useful for loading entirely from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine. The placement of this option on the *ld* line is significant; it must be placed before the library which defines the symbol.

-x
do not preserve local symbols in the output symbol table; enter external and static symbols only. This option saves some space in the output file.

-z
do not bind anything to address zero. This is done by shifting all linked elements up by .5 gigabytes. This option allows runtime detection of NULL pointers, or may be used by the program to dynamically attach a shared memory segment to address zero.

-n
permit references to misaligned data. This option allows runtime emulation of instructions referencing misaligned data (on systems providing such runtime support), rather than having such accesses result in bus errors. Misaligned data may indicate that non-standard (i.e., non-portable) libraries and functions have been used.

-L *dir*
change the algorithm of searching for **lib***x***.a** to look in **dir** before looking in **LIBDIR** and **LLIBDIR**. This option is effective only if it precedes the **-l** option on the command line.

-M
output a message for each multiply-defined external definition.

-N
put the text section at the beginning of the text segment rather than after all header information, and put the data section immediately following text in the core image.

-D
do not, by default, append **lib***g***.a** to the list of libraries to be scanned during the link. Normally, the option **-lg** is implied on the *ld* command line.

-V
causes *ld* to output a message giving information about the version of *ld* being used. No other processing is performed.

-VS *num*
use *num* as a decimal version stamp identifying the **a.out** file that is produced. The version stamp is stored in the optional header.

-Y[LU],*dir*
change the default directory used for finding libraries. If **L** is specified, the first default directory that *ld* searches, **LIBDIR**, is replaced by *dir*. If **U** is specified and *ld* has been built with a second default directory, **LLIBDIR**, then that directory is replaced by *dir*. If *ld* was built with only one default directory and **U** is specified, a warning is printed and the option is ignored.

FILES

LIBDIR/libx.a	libraries
LLIBDIR/libx.a	libraries
a.out	output file
LIBDIR	usually **/lib**
LLIBDIR	usually **/usr/lib**

SEE ALSO

as(1), cc(1), exit(2), end(3C), a.out(4), ar(4),

CAVEATS

Through its options and input directives, the common link editor gives users great flexibility; however, those who use the input directives must assume some added responsibilities. Input directives and options should ensure the following properties for programs: C defines a zero pointer as NULL. A pointer to which zero has been assigned must not point to any object. To satisfy this, users must not place any object at virtual address zero in the program's address space. When the link editor is called through *cc*(1), a startup routine is linked with the user's program. This routine calls *exit*(~) (see *exit*(2)) after execution of the main program. If the user calls the link editor directly, then the user must ensure that the program always calls *exit*(~) instead of falling through the end of the entry routine.

The symbols *etext*, *edata*, and *end* (see *end*(3C)) are reserved and are defined by the link editor. It is incorrect for a user program to redefine them.

If the link editor does not recognize an input file as an object file or an archive file, it assumes that it contains link editor directives and attempts to parse it. This occasionally produces an error message complaining about *syntax errors.*

Arithmetic expressions may only have one forward referenced symbol per expression.

Shared libraries are not supported.

NAME

lex – generate programs for simple lexical tasks

SYNOPSIS

lex [**–rctvn**] [*file*] ...

DESCRIPTION

The *lex* command generates programs to be used in simple lexical analysis of text.

The input *files* (standard input default) contain strings and expressions to be searched for, and C text to be executed when strings are found.

A file **lex.yy.c** is generated which, when loaded with the library, copies the input to the output except when a string specified in the file is found; then the corresponding program text is executed. The actual string matched is left in *yytext*, an external character array. Matching is done in order of the strings in the file.

The strings may contain square brackets to indicate character classes, as in **[abx–z]** to indicate **a**, **b**, **x**, **y**, and **z**; and the operators ***, **+**, and **?** mean respectively any non-negative number of, any positive number of, and either zero or one occurrence of, the previous character or character class. The character **.** is the class of all ASCII characters except 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.

EXAMPLE

```
D       [0–9]
%%
if      printf("IF statement\n");
[a–z]+  printf("tag, value %s\n",yytext);
0{D}+   printf("octal number %s\n",yytext);
{D}+    printf("decimal number %s\n",yytext);
"++"    printf("unary op\n");
"+"     printf("binary op\n");
"/*"     skipcommnts();
%%
```

```
skipcommnts()
{
        for (;;)
        {
                while (input() != '*')
                        ;
                if (input() != '/')
                        unput(yytext[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 flag **-r** indicates RATFOR actions, **-c** indicates C actions and is the default, **-t** causes the **lex.yy.c** program to be written instead to standard output, **-v** provides a one-line summary of statistics, **-n** will not print out the **-v** summary. Multiple files are treated as a single file. If no files are specified, standard input is used.

Certain table sizes for the resulting finite state machine can be set in the definitions section:

%p *n* number of positions is *n* (default 2500)

%n *n* number of states is *n* (500)

%e *n* number of parse tree nodes is *n* (1000)

%a *n* number of transitions is *n* (2000)

%k *n* number of packed character classes is *n* (1000)

%o *n* size of output array is *n* (3000)

The use of one or more of the above automatically implies the **-v** option, unless the **-n** option is used.

SEE ALSO

yacc(1)

BUGS

The **-r** option is not yet fully operational.

NAME

line – read one line

SYNOPSIS

line

DESCRIPTION

line copies one line (up to a 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

lint – a C program checker

SYNOPSIS

lint [*option*] ... *file* ...

DESCRIPTION

The *lint* command attempts to detect features of the C program files that are likely to be bugs, non-portable, or wasteful. It also checks type usage more strictly than the compilers. Among the things that are currently detected are unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions whose value is constant. Moreover, the usage of functions is checked to find functions that return values in some places and not in others, functions called with varying numbers or types of arguments, and functions whose values are not used or whose values are used but none returned.

Arguments whose names end with **.c** are taken to be C source files. Arguments whose names end with **.ln** are taken to be the result of an earlier invocation of *lint* with either the **–c** or the **–o** option used. The **.ln** files are analogous to **.o** (object) files that are produced by the *cc*(1) command when given a **.c** file as input. Files with other suffixes are warned about and ignored.

lint will take all the **.c**, **.ln**, and **llib-l*x*.ln** (specified by **–l*x***) files and process them in their command line order. By default, *lint* appends the standard C lint library (**llib-lc.ln**) to the end of the list of files. However, if the **–p** option is used, the portable C lint library (**llib-port.ln**) is appended instead. When the **–c** option is not used, the second pass of *lint* checks this list of files for mutual compatibility. When the **–c** option is used, the **.ln** and the **llib-l*x*.ln** files are ignored.

Any number of *lint* options may be used, in any order, intermixed with file-name arguments. The following options are used to suppress certain kinds of complaints:

–a
Suppress complaints about assignments of long values to variables that are not long.

–b
Suppress complaints about **break** statements that cannot be reached. (Programs produced by *lex* or *yacc* will often result in many such complaints).

–h
Do not apply heuristic tests that attempt to intuit bugs, improve style, and reduce waste.

–u
Suppress complaints about functions and external variables used and not defined, or defined and not used. (This option is suitable for running *lint* on a subset of files of a larger program).

–v
Suppress complaints about unused arguments in functions.

–x
Do not report variables referred to by external declarations but never used.

The following arguments alter *lint*'s behavior:

–l*x*
Include additional lint library **llib-l*x*.ln**. For example, you can include a lint version of the math library **llib-lm.ln** by inserting **–lm** on the command line. This argument does not suppress the default use of **llib-lc.ln**. These lint libraries must be in the assumed directory. This option can be used to reference local lint libraries and is useful in the development of multi-file projects.

−n
Do not check compatibility against either the standard or the portable lint library.

−p
Attempt to check portability to other dialects (IBM and GCOS) of C. Along with stricter checking, this option causes all non-external names to be truncated to eight characters and all external names to be truncated to six characters and one case.

−c
Cause *lint* to produce a **.ln** file for every **.c** file on the command line. These **.ln** files are the product of *lints* first pass only and are not checked for inter-function compatibility.

−o lib
Cause *lint* to create a lint library with the name **llib-l***lib***.ln**. The **−c** option nullifies any use of the **−o** option. The lint library produced is the input that is given to *lints* second pass. The **−o** option simply causes this file to be saved in the named lint library.

To produce a **llib-l***lib***.ln** without extraneous messages, use of the **−x** option is suggested. The **−v** option is useful if the source file(s) for the lint library are just external interfaces (for example, the way the file **llib-lc** is written). These option settings are also available through the use of "lint comments" (see below).

The **−D**, **−U**, and **−I** options of *cpp*(1) and the **−g** and **−O** options of *cc*(1) are also recognized as separate arguments. The **−g** and **−O** options are ignored, but, by recognizing these options, *lints* behavior is closer to that of the *cc*(1) command. Other options are warned about and ignored. The pre-processor symbol "lint" is defined to allow certain questionable code to be altered or removed for *lint*. Therefore, the symbol "lint" should be thought of as a reserved word for all code that is planned to be checked by *lint*.

Certain conventional comments in the C source will change the behavior of *lint*:

/*NOTREACHED*/
at appropriate points stops comments about unreachable code. (This comment is typically placed just after calls to functions like *exit*(2)).

/*VARARGS n */
suppresses the usual checking for variable numbers of arguments in the following function declaration. The data types of the first *n* arguments are checked; a missing *n* is taken to be 0.

/*ARGSUSED*/
turns on the **−v** option for the next function.

/*LINTLIBRARY*/
at the beginning of a file shuts off complaints about unused functions and function arguments in this file. This is equivalent to using the **−v** and **−x** options.

lint produces its first output on a per-source-file basis. Complaints about included files are collected and printed after all source files have been processed. Finally, if the **−c** option is not used, information gathered from all input files is collected and checked for consistency. At this point, if it is not clear whether a complaint stems from a given source file or from one of its included files, the source file name will be printed followed by a question mark.
The behavior of the **−c** and the **−o** options allows for incremental use of *lint* on a set of C source files. Generally, you invoke *lint* once for each source file with the **−c** option. Each of these invocations produces a **.ln** file which corresponds to the **.c** file, and prints all messages that are about just that source file. After all the source files have been separately run through *lint*, it is invoked once more (without the **−c** option), listing all the **.ln** files with the needed **−l***x* options. This will print all the inter-file inconsistencies. This scheme works well with *make*(1); it allows *make* to be used to *lint* only the source files that have been modified since the last time the set of source files were *lint*ed.

FILES

LLIBDIR	the directory where the lint libraries specified by the **–lx** option must exist, usually /usr/lib
LLIBDIR/lint[12]	first and second passes
LLIBDIR/llib-lc.ln	declarations for C Library functions (binary format; source is in *LLIBDIR*/llib-lc)
LLIBDIR/llib-port.ln	declarations for portable functions (binary format; source is in *LLIBDIR*/llib-port)
LLIBDIR/llib-lm.ln	declarations for Math Library functions (binary format; source is in *LLIBDIR*/llib-lm)
TMPDIR/*lint*	temporaries
TMPDIR	usually /usr/tmp but can be redefined by setting the environment variable **TMPDIR** (see *tempnam*() in *tmpnam*(3S)).

SEE ALSO

cc(1), cpp(1), make(1).

BUGS

exit(2), *setjmp*(3C), and other functions that do not return are not understood; this causes various lies.

NAME

list – from a common object file produce a C source listing with line numbers

SYNOPSIS

list [**–V**] [–h] [**-F** *function*] source file [*source file . . .*]
[*object-file*]

DESCRIPTION

The **list** command produces a C source listing with line number information attached. If multiple C source files were used to create the object file, **list** will accept multiple file names. The object file is taken to be the last non-C source file argument. If no object file is specified, the default object file, **a.out**, will be used.

Line numbers will be printed for each line marked as breakpoint inserted by the compiler (generally, each executable C statement that begins a new line of source). Line numbering begins anew for each function. Line number 1 is always the line containing the left curly brace ({) that begins the function body. Line numbers will also be supplied for inner block redeclarations of local variables so that they can be distinguished by the symbolic debugger.

The following options are interpreted by **list** and may be given in any order:

–V
Print, on standard error, the version number of the *list* command executing.

–h
Suppress heading output.

–F*function*
List only the named *function*. The **–F** option may be specified multiple times on the command line.

SEE ALSO

as(1), cc(1), ld(1).

CAVEATS

Object files given to *list* must have been compiled with the **–g** option of **cc**(1).

Since *list* does not use the C preprocessor, it may be unable to recognize function definitions whose syntax has been distorted by the use of C preprocessor macro substitutions.

DIAGNOSTICS

If *name* cannot be read, *list* will produce the error message:

list: name: cannot open

If the source file names do not end in **.c** , the message is:

list: name: invalid C source name.

An invalid object file name will cause the message:

list: name: bad magic

If any of the symbolic debugging information is missing, one of the following messages will be printed:

list: name: symbols have been stripped, cannot proceed,
list: name: cannot read line numbers
list: name: not in symbol table.

The following messages are produced when *list* becomes confused by **#ifdefs** in the source file:

list: name: cannot find function in symbol table,
list: name: out of sync: too many }
list: name: unexpected end-of-file.

When either symbol debugging information is missing, or *list* has been confused by C preprocessor statements. it displays the error message:

list: name: missing or inappropriate line numbers

NAME

locate – identify a command using keywords

SYNOPSIS

[help] locate
[help] locate [*keyword1* [*keyword2*] ...]

DESCRIPTION

The *locate* command is part of the Help Facility, and provides online assistance with identifying commands.

Without arguments, the initial *locate* screen is displayed from which the user may enter keywords functionally related to the action of the commands they wish to have identified. A user may enter keywords and receive a list of commands whose functional attributes match those in the keyword list, or may 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 on 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 (*sh*(1)) by typing a ! and the command to be executed. The screen will be redrawn if the command that was executed 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

glossary(1), help(1), sh(1), starter(1), usage(1), term(5).

WARNINGS

If the shell variable TERM (see *sh*(1)) 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

lock – reserve a terminal

SYNOPSIS

lock [**–number**]

DESCRIPTION

Lock requests a password from the user, reads it again for verification and then it will normally not relinquish the terminal until the password is repeated. There are three other conditions under it will terminate: it accepts the password for root as an alternative to the one given by the user, it will timeout after some interval of time, and it may be killed by somebody with the appropriate permission. The default time limit is 15 minutes but it may be changed with the **–number** option where *number* is the time limit in minutes.

NAME

login – sign on

SYNOPSIS

login [*name* [*env-var* ...]]

DESCRIPTION

The *login* command is used at the beginning of each terminal session and allows you to identify yourself to the system. It may be invoked as a command or by the system when a connection is first established. Also, it is invoked by the system when a previous user has terminated the initial shell by typing a **CTRL–D** to indicate an EOF.

If *login* is invoked as a command, it must replace the initial command interpreter. This is accomplished by typing the following from the initial shell:

exec login

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 message:

Login incorrect

A new login prompt will appear. 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 (e.g., 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 (usually **sh**(1)) is initialized, and the file **.profile** in the working directory is executed, if it exists. 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 (i.e., **–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 to:

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:

L*n*=xxx

where *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, they replace the older value. There are two exceptions. The variables PATH and SHELL cannot be changed. This prevents people, logging into restricted shell environments, from spawning secondary shells that are not restricted. Both *login* and *getty* understand simple single-character quoting conventions. Typing a backslash in front of a character quotes it and allows the inclusion of such things as spaces and tabs.

FILES

/etc/utmp	accounting
/etc/wtmp	accounting
/usr/mail/*your-name*	**mailbox for user** *your-name*
/usr/adm/loginlog	record of failed login attempts
/etc/motd	message-of-the-day
/etc/passwd	password file
/etc/shadow	shadow password file
/etc/profile	system profile
.profile	user's login profile

SEE ALSO

mail(1), newgrp(1), sh(1), su(1M), loginlog(4), passwd(4), profile(4), environ(5).

DIAGNOSTICS

login incorrect if the user name or the password cannot be matched.
No shell, **cannot open password file**, or **no directory**: consult a UNIX system programming counselor.

No utmp entry. You must exec *login* from the lowest level **sh** if you attempted to execute *login* as a command without using the shell's *exec* internal command or from other than the initial shell.

login(1) will output one of the following messages if it finds an incorrect entry in the **/etc/group** file.

fgetgrent: too many users in group < *group*>
fgetgrent: line exceeds system limit (*limit*)

where:

group
is the invalid entry

limit
is the system limit for the length of a line.

The output of one of these messages is a warning only, and does not result in the failure of the command.

LOGNAME(1)

NAME

logname – get login name

SYNOPSIS

logname

DESCRIPTION

logname returns the user's login name.

FILES

/etc/profile

SEE ALSO

env(1), login(1), cuserid(3S), environ(5).

NAME

lorder – find ordering relation for an object library

SYNOPSIS

lorder *file* ...

DESCRIPTION

The input is one or more object or library archive *files* (see *ar*(1)). The standard output is a list of pairs of object file or archive member names, meaning that the first file of the pair refers to external identifiers defined in the second. The output may be processed by *tsort*(1) to find an ordering of a library suitable for one-pass access by *ld*(1). Note that the link editor *ld*(1) is capable of multiple passes over an archive in the portable archive format (see *ar*(4)) and does not require that *lorder*(1) be used when building an archive. The usage of the *lorder*(1) command may, however, allow for a slightly more efficient access of the archive during the link edit process.

The following example builds a new library from existing **.o** files:

```
ar -cr library lorder *.o | tsort
```

FILES

TMPDIR/*symref temporary files

TMPDIR/*symdef temporary files

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

SEE ALSO

ar(1), ld(1), tsort(1), ar(4)

CAVEAT

lorder will accept as input any object or archive file, regardless of its suffix, provided there is more than one input file. If there is but a single input file, its suffix must be **.o**.

NAME

lp, cancel – send/cancel requests to an 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 file names 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 using the file name(s) and – 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 file names, but may be listed in any order. The following options are available for *lp*:

–c
Make copies of the *files* to be printed immediately 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 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

Do not 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 assigned **immediate**, the requests are printed in the reverse order queued. If a request is currently printing on the desired printer, you have to 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 (i.e., **–o $option sub 1$ –o $option sub 2$** ... **–o $option sub n$**), or by specifying a list of options with one –o keyletter (i.e., **–o $option sub 1$, $option sub 2$,** ... *$option sub n$*). 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

Do not insert a form feed between the files given, if submitting a job to print more than one file.

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 with 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 any printer that can select the character set, and that can 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, then 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.

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.

WARNING

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 find out what print wheels are available and on which printers. For printers that have selectable character sets, you will get the standard character set if you do not use the **-S** option.

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).

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, then *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. For example:

–u "user1, user2, user3"

Specifying "all" after any keyletters that take "list" as an argument causes all information relevant to the keyletter to be printed. For example, the following command prints the status of all output requests.

lpstat –o all

The options are:

–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.

–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 the 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 path names 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

ls – list contents of directory

SYNOPSIS

ls [**–RadLCxmlnogrtucpFbqisfeE**] [names]

DESCRIPTION

For each directory argument, *ls* lists the contents of the directory; for each file argument, *ls* repeats its name and any other information requested. The output is sorted alphabetically by default. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments appear before directories and their contents.

There are three major listing formats. The default format is to list one entry per line, the **–C** and **–x** options enable multi-column formats, and the **–m** option enables stream output format. In order to determine output formats for the **–C**, **–x**, and **–m** options, *ls* uses an environment variable, **COLUMNS**, to determine the number of character positions available on one output line. If this variable is not set, the *terminfo*(4) database is used to determine the number of columns, based on the environment variable **TERM**. If this information cannot be obtained, 80 columns are assumed.

The **–e** and **–E** options are useful for files residing in REAL/IX fast file systems.

The *ls* command has the following options:

–R Recursively list subdirectories encountered.

–a List all entries, including those that begin with a dot (**.**), which are normally not listed.

–d If an argument is a directory, list only its name (not its contents); often used with **–l** to get the status of a directory.

–L If an argument is a symbolic link, list the file or directory the link references instead of the link itself.

–C Multi-column output with entries sorted down the columns.

–x Multi-column output with entries sorted across rather than down the page.

–m Stream output format; files are listed across the page, separated by commas.

–l List in long format, giving mode, number of links, owner, group, size in bytes, and time of last modification for each file (see below). If the file is a special file, the size field will instead contain the major and minor device numbers rather than a size.

–n The same as **–l**, except that the owner's **UID** and group's **GID** numbers are printed, rather than the associated character strings.

–o The same as **–l**, except that the group is not printed.

–g The same as **–l**, except that the owner is not printed.

–r Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.

–t Sort by time stamp (latest first) instead of by name. The default is the last modification time. (See **–n** and **–c**.)

–u Use time of last access instead of last modification for sorting (with the **–t** option) or printing (with the **–l** option).

–c Use time of last modification of the i-node (file created, mode changed, etc.) for sorting (**–t**) or printing (**–l**).

–p Put a slash (/) after each filename if that file is a directory.

–F Put a slash (/) after each filename if that file is a directory and put an asterisk (*) after each filename if that file is executable.

–b Force printing of non-graphic characters to be in the octal **\ddd** notation.

–q Force printing of non-graphic characters in file names as the character (?).

–i For each file, print the i-node 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.

–e Show extended inode information for files residing in REAL/IX fast file systems. Gives flags describing the allocation mechanism, the sum of the contiguous extents (in bytes), the logical size of the file (in bytes), and the file name. The following options are turned off: **–C, –x, –m, –l, –n, –o, –g, –s,** and **–f.** The flags used are:

- e contiguous extent-based file
- s physical space will shrink on truncates and creates
- g writes beyond the end of physical space will not automatically grow the file.

–E Provides same information as **–e,** with the contiguous extent list printed before the logical file size information. The extent list is a pair of numbers separated by "/"; the first number is the byte offset of the extent from the base of the file system, and the second number is the size of the extent.

The mode printed under the **–l** option consists of ten characters. The first character may be one of the following:

- **d** the entry is a directory;
- **b** the entry is a block special file;
- **c** the entry is a character special file;
- **l** the entry is a symbolic link;
- **e** the entry is a contiguous, extent-based 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 next to permissions of others in the user-group of the file, and the last to all others. Within each set, the three characters indicate permission to read, to write, and to execute the file as a program, respectively. For a directory, "execute" permission is interpreted to mean permission to search the directory for a specified file.

ls –l (the long list) prints output similar to the following:

```
–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 has login ID "smith" and the file belongs to group "dev" (perhaps indicating "development") which is probably the group to which "smith" belongs. 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 first set of examples refers to permissions:

```
–rwxr––r––
```

This describes a file that is readable, writable, and executable by the user and readable by the group and others.

```
–rwsr–xr–x
```

The second example describes a file that is readable, writable, and executable by the user, readable and executable by the group and others, and allows its user-ID to be assumed, during execution, by the user presently executing it.

```
–rw–rwl–––
```

This example describes a file that is readable and writable only by the user and the group and can be locked during access.

```
ls -a
```

This command will print the names of all files in the current directory, including those that begin with a dot (**.**), which normally do not print.

```
ls -aisn
```

This command will provide you with quite a bit of information including all files, including non-printing ones (**a**), the **i**-number—the memory address of the i-node associated with the file—printed in the left-hand column (**i**); the size (in blocks) of the files, printed in the column to the right of the i-numbers (**s**); finally, the report is displayed in the **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.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks, is printed.

The following command lists information on files in a fast file system directory. It will list the inode, flags, sum of contiguous extents, extent list, logical file size, and file name of each file.

```
ls -Ei
```

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).

NOTES

In a Remote File Sharing environment, you may not have the permissions that the output of the **ls –l** command leads you to believe.

BUGS

Unprintable characters in file names may confuse the columnar output options.

NAME

m4 – macro processor

SYNOPSIS

m4 [*options*] [*files*]

DESCRIPTION

The *m4* command is a macro processor intended as a front end for Ratfor, C, and other languages. Each of the argument files is processed in order; if there are no files, or if a file name is –, the standard input is read. The processed text is written on the standard output.

The options and their effects are as follows:

–e
Operate interactively. Interrupts are ignored and the output is unbuffered.

–s
Enable line sync output for the C preprocessor (#line ...)

–B*int*
Change the size of the push-back and argument collection buffers from the default of 4,096.

–H*int*
Change the size of the symbol table hash array from the default of 199. The size should be prime.

–S*int*
Change the size of the call stack from the default of 100 slots. Macros take three slots, and non-macro arguments take one.

–T*int*
Change the size of the token buffer from the default of 512 bytes.

To be effective, these flags must appear before any file names and before any **–D** or **–U** flags:

–D*name*[=*val*]
Defines *name* to *val* or to **NULL** in *val*'s absence.

–U*name*
undefines *name*.

Macro calls have the form:

name(arg1,arg2, ..., argn)

The (must immediately follow the name of the macro. If the name of a defined macro is not followed by a (, it is deemed to be a call of that macro with no arguments. Potential macro names consist of alphabetic letters, digits, and underscore _, where the first character is not a digit.

Leading unquoted blanks, tabs, and newlines are ignored while collecting arguments. Left and right single quotes are used to quote strings. The value of a quoted string is the string stripped of the quotes.

When a macro name is recognized, its arguments are collected by searching for a matching right parenthesis. If fewer arguments are supplied than are in the macro definition, the trailing arguments are taken to be **NULL**. Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses which happen to turn up within the value of a nested call are as effective as those in the original input text. After argument collection, the value of the macro is pushed back onto the input stream and rescanned.

m4 makes available the following built-in macros. They may be redefined, but once this is done the original meaning is lost. Their values are **NULL** unless otherwise stated.

define
the second argument is installed as the value of the macro whose name is the first argument. Each occurrence of **$***n* in the replacement text, where *n* is a digit, is replaced by the *n*-th argument. Argument 0 is the name of the macro; missing arguments are replaced by the **NULL** string; **$#** is replaced by the number of arguments; **$*** is replaced by a list of all the arguments separated by commas; **$@** is like **$***, but each argument is quoted (with the current quotes).

undefine
removes the definition of the macro named in its argument.

defn
returns the quoted definition of its argument(s). It is useful for renaming macros, especially built-ins.

pushdef
like *define*, but saves any previous definition.

popdef
removes current definition of its argument(s), exposing the previous one, if any.

ifdef
if the first argument is defined, the value is the second argument, otherwise the third. If there is no third argument, the value is **NULL**. The word *unix* is predefined on UNIX system versions of *m4*.

shift
returns all but its first argument. The other arguments are quoted and pushed back with commas in between. The quoting nullifies the effect of the extra scan that will subsequently be performed.

changequote
change quote symbols to the first and second arguments. The symbols may be up to five characters long. *Changequote* without arguments restores the original values (i.e., ` ').

changecom
change left and right comment markers from the default # and newline. With no arguments, the comment mechanism is effectively disabled. With one argument, the left marker becomes the argument and the right marker becomes newline. With two arguments, both markers are affected. Comment markers may be up to five characters long.

divert
m4 maintains 10 output streams, numbered 0-9. The final output is the concatenation of the streams in numerical order; initially stream 0 is the current stream. The *divert* macro changes the current output stream to its (digit-string) argument. Output diverted to a stream other than 0 through 9 is discarded.

undivert
causes immediate output of text from diversions named as arguments, or all diversions if no argument. Text may be undiverted into another diversion. Undiverting discards the diverted text.

divnum
returns the value of the current output stream.

dnl
reads and discards characters up to and including the next newline.

ifelse
has three or more arguments. If the first argument is the same string as the second, the value is the third argument. If not, and if there are more than four arguments, the process is repeated with arguments 4, 5, 6 and 7. Otherwise, the value is either the fourth string, or, if it is not present, **NULL**.

incr
returns the value of its argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.

decr
returns the value of its argument decremented by 1.

eval
evaluates its argument as an arithmetic expression, using 32-bit arithmetic. Operators include +, −, *, /, %, ^ (exponentiation), bitwise &, |, ^, and ~; relationals; parentheses. Octal and hex numbers may be specified as in C. The second argument specifies the radix for the result; the default is 10. The third argument may be used to specify the minimum number of digits in the result.

len
returns the number of characters in its argument.

index
returns the position in its first argument where the second argument begins (zero origin), or −1 if the second argument does not occur.

substr
returns a substring of its first argument. The second argument is a zero origin number selecting the first character; the third argument indicates the length of the substring. A missing third argument is taken to be large enough to extend to the end of the first string.

translit
transliterates the characters in its first argument from the set given by the second argument to the set given by the third. No abbreviations are permitted.

include
returns the contents of the file named in the argument.

sinclude
is identical to *include*, except that it says nothing if the file is inaccessible.

syscmd
executes the command given in the first argument. No value is returned.

sysval
is the return code from the last call to *syscmd*.

maketemp
fills in a string of XXXXX in its argument with the current process ID.

m4exit
causes immediate exit from *m4*. Argument 1, if given, is the exit code; the default is 0.

m4wrap
argument 1 will be pushed back at final EOF; example: m4wrap(`cleanup()')

errprint
prints its argument on the diagnostic output file.

M4(1)

dumpdef
prints current names and definitions, for the named items, or for all if no arguments are given.

traceon
with no arguments, turns on tracing for all macros (including built-ins). Otherwise, turns on tracing for named macros.

traceoff
turns off trace globally and for any macros specified. Macros specifically traced by *traceon* can be untraced only by specific calls to *traceoff*.

SEE ALSO

cc(1), cpp(1).

NAME

m68k, m88k, i386, pdp11, u3b, u3b2, u3b5, vax – get processor type truth value

SYNOPSIS

m68k

m88k

i386

pdp11

u3b

u3b2

u3b5

vax

DESCRIPTION

The following commands will return a true value (exit code of 0) if you are on a processor that the command name indicates.

m68k
True if you are on a Motorola M68000-Family-based microcomputer.

m88k
True if you are on a Motorola M88000-Family-based microcomputer.

i386
True if you are on an Intel 80x86-Family-based microcomputer.

pdp11
True if you are on a PDP-11/45 or PDP-11/70.

u3b
True if you are on a 3B20 computer.

u3b2
True if you are on a 3B2 computer.

u3b5
True if you are on a 3B5 computer.

vax
True if you are on a VAX-11/750 or VAX-11/780.

The commands that do not apply will return a false (non-zero) value. These commands are often used within makefiles (see **make**(1)) and shell procedures (see **sh**(1)) to increase portability.

SEE ALSO

make(1), sh(1), test(1), true(1).

NAME

mail, rmail – send mail to users or read mail

SYNOPSIS

Sending mail:

mail [**–oswt**] *persons*

rmail [**–oswt**] *persons*

Reading mail:

mail [**–ehpqr**] [**–f** *file*] [**–F** *persons*]

DESCRIPTION

Sending mail:

The command-line arguments that follow affect SENDING mail:

–o
suppresses the address optimization facility.

–s
suppresses the addition of a **NEWLINE** at the top of the letter being sent. See WARNINGS below.

–w
causes a letter to be sent to a remote user without waiting for the completion of the remote transfer program.

–t
causes a **To:** line to be added to the letter, showing the intended recipients.

A *person* is usually a user name recognized by *login*(1). When *persons* are named, *mail* assumes a message is being sent (except in the case of the **–F** option). It reads from the standard input up to an end-of-file (control-d), or until it reads a line consisting of just a period. When either of those signals is received, *mail* adds the *letter* to the *mailfile* for each *person*. A *letter* is a *message* preceded by a *postmark*. The message is preceded by the sender's name and a *postmark*. A *postmark* consists of one or more 'From' lines followed by a blank line (unless the –s argument was used).

If a letter is found to be undeliverable, it is returned to the sender with diagnostics that indicate the location and nature of the failure. If *mail* is interrupted during input, the file **dead.letter** is saved to allow editing and resending. **dead.letter** is recreated every time it is needed, erasing any previous contents.

rmail only permits the sending of mail; *uucp*(1C) uses *rmail* as a security precaution.

If the local system has the BNU installed, mail may be sent to a recipient on a remote system. Prefix *person* by the system name and exclamation point. A series of system names separated by exclamation points can be used to direct a letter through an extended network.

Reading Mail:

The command-line arguments that follow affect READING mail:

–e
causes mail not to be printed. An exit value of 0 is returned if the user has mail; otherwise, an exit value of 1 is returned.

–h
causes a window of headers to be displayed rather than the latest message. The display is followed by the '?' prompt.

-p
causes all messages to be printed without prompting for disposition.

-q
causes *mail* to terminate after interrupts. Normally an interrupt causes only the termination of the message being printed.

-r
causes messages to be printed in first-in, first-out order.

-f*file*
causes *mail* to use *file* (e.g., **mbox**) instead of the default *mailfile*.

-F*persons*
entered into an empty *mailbox*, causes all incoming mail to be forwarded to *persons*.

mail, unless otherwise influenced by command-line arguments, prints a user's mail messages in last-in, first-out order. For each message, the user is prompted with a ?, and a line is read from the standard input. The following commands are available to determine the disposition of the message:

newline, +, or n
Go on to next message.

d, or dp
Delete message and go on to next message.

d #
Delete message number #. Do not go on to next message.

dq
Delete message and quit *mail*.

h
Display a window of headers around current message.

h #
Display header of message number #.

h a
Display headers of ALL messages in the user's *mailfile*.

h d
Display headers of messages scheduled for deletion.

p
Print current message again.

-
Print previous message.

a
Print message that arrived during the *mail* session.

#
Print message number #.

r [*users* **]**
Reply to the sender, and other *user(s)*, then delete the message.

s [*files* **]**
Save message in the named *files* (**mbox** is default).

y
Same as save.

u [#]
Undelete message number # (default is last read).

w [*files*]
Save message, without its top-most header, in the named *files* (**mbox** is default).

m [*persons*]
Mail the message to the named *persons*.

q, or **ctl-d**
Put undeleted mail back in the *mailfile* and quit *mail*.

x
Put all mail back in the *mailfile* unchanged and exit *mail*.

!*command*
Escape to the shell to do *command*.

?
Print a command summary.

When a user logs in, the presence of mail, if any, is indicated. Also, notification is made if new mail arrives while using *mail*.

The *mailfile* may be manipulated in two ways to alter the function of *mail*. The *other* permissions of the file may be read-write, read-only, or neither read nor write to allow different levels of privacy. If changed to other than the default, the file will be preserved even when empty to perpetuate the desired permissions. The file may also contain the first line:

Forward to *person*

which will cause all mail sent to the owner of the *mailfile* to be forwarded to *person*. A "Forwarded by..." message is added to the header. This is especially useful in a multi-machine environment to forward all of a person's mail to a single machine, and to keep the recipient informed if the mail has been forwarded. Installation and removal of forwarding is done with the **–F** option.

To forward all of your mail to systema!user, type:

mail -Fsystema!user

To forward to more than one user, type:

mail -F"user1,systema!user2,systema!systemb!user3"

Note that when more than one user is specified, the whole list should be enclosed in double quotes so that it may all be interpreted as the operand of the **–F** option. The list can be up to 1024 bytes; either commas or white space can be used to separate users.

To remove forwarding, type:

mail –F ""

The pair of double quotes is mandatory to set a **NULL** argument for the **–F** option.

In order for forwarding to work properly the *mailfile* should have "mail" as group ID, and the group permission should be read-write.

FILES

/etc/passwd	to identify sender and locate persons
/usr/mail/user	incoming mail for *user*; i.e., the *mailfile*
$HOME/mbox	saved mail

$MAIL	variable containing pathname of *mailfile*
/tmp/ma*	temporary file
/usr/mail/*.lock	lock for mail directory
/usr/lib/uucp/Systems	machine entry for UUCP service over network
dead.letter	unmailable text

SEE ALSO

login(1), mailx(1), write(1),
System Administrator's Guide, Chapter 11, section on UUCP over TCP/IP
Managing UUCP and Usenet, O'Reilly and Associates

NOTES

It is necessary to set up a network entry in the **/usr/lib/uucp/Systems** file for mail and rmail to work. In Chapter 11 of the *System Administrator's Guide*, the section on **Setting Up the Systems File** gives a thorough explanation of how to create the entry.

DIAGNOSTICS

mail(1) will output one of the following messages if it finds an incorrect entry in the **/etc/group** file:

fgetgrent: too many users in group < *group*>
fgetgrent: line exceeds system limit (*limit*)

where:

group
is the invalid entry

limit
is the system limit for the length of a line.

The output of one of these messages is a warning only, and does not result in the failure of the command.

WARNING

The "Forward to person" feature may result in a loop, if *sys1!userb* forwards to *sys2!userb* and *sys2!userb* forwards to *sys1!userb*. The symptom is a message saying "unbounded...saved mail in dead.letter."

The **–s** option should be used with caution. It allows the text of a message to be interpreted as part of the postmark of the letter, possibly causing confusion to other *mail* programs. To allow compatibility with *mailx*(1), if the first line of the message is "Subject:...", the addition of a <newline> is suppressed whether or not the **–s** option is used.

BUGS

Conditions sometimes result in a failure to remove a lock file.

After an interrupt, the next message may not be printed; printing may be forced by typing a **p**.

NAME

mailx – interactive message processing system

SYNOPSIS

mailx [*options*] [*name...*]

DESCRIPTION

The command *mailx* provides a comfortable, flexible environment for sending and receiving messages electronically. When reading mail, *mailx* provides commands to facilitate saving, deleting, and responding to messages. When sending mail, *mailx* allows editing, reviewing and other modification of the message as it is entered.

Many of the remote features of *mailx* will only work if BNU are installed on your system.

Incoming mail is stored in a standard file for each user, called the *mailbox* for that user. When *mailx* is called to read messages, the *mailbox* is the default place to find them. As messages are read, they are marked to be moved to a secondary file for storage, unless specific action is taken, so that the messages need not be seen again. This secondary file is called the *mbox* and is normally located in the user's HOME directory (see *MBOX* **ENVIRONMENT VARIABLES** for a description of this file). Messages can be saved in other secondary files named by the user. Messages remain in a secondary file until forcibly removed.

The user can access a secondary file by using the **–f** option of the *mailx* command. Messages in the secondary file can then be read or otherwise processed using the same COMMANDS as in the primary *mailbox*. This gives rise within these pages to the notion of a current *mailbox*.

On the command line, *options* start with a dash (–) and any other arguments are taken to be destinations (recipients). If no recipients are specified, *mailx* will attempt to read messages from the *mailbox*. Command line options are:

–e
Test for presence of mail. *mailx* prints nothing and exits with a successful return code if there is mail to read.

–f [*filename*]
Read messages from *filename* instead of *mailbox*. If no *filename* is specified, the *mbox* is used.

–F
Record the message in a file named after the first recipient. Overrides the *record* variable, if set (see **ENVIRONMENT VARIABLES**).

–h *number*
The number of network "hops" made so far. This is provided for network software to avoid infinite delivery loops. (See *addsopt* under **ENVIRONMENT VARIABLES**.)

–H
Print header summary only.

–i
Ignore interrupts. (See *ignore* under **ENVIRONMENT VARIABLES**.)

–n
Do not initialize from the system default *mailx.rc* file.

–N
Do not print initial header summary.

–r *address*
Pass *address* to network delivery software. All tilde commands are disabled. (See *addsopt* under **ENVIRONMENT VARIABLES**.)

–s *subject*
Set the Subject header field to *subject*.

–u *user*
Read *user*'s *mailbox*. This is only effective if *user*'s *mailbox* is not read protected.

–U

Convert *uucp* style addresses to internet standards. Overrides the "conv" environment variable. (See *addsopt* under **ENVIRONMENT VARIABLES**.)

When reading mail, *mailx* is in *command mode*. A header summary of the first several messages is displayed, followed by a prompt indicating *mailx* can accept regular commands (see **COMMANDS**). When sending mail, *mailx* is in *input mode*. If no subject is specified on the command line, a prompt for the subject is printed. (A "subject" longer than 1024 characters will cause *mailx* to dump core) As the message is typed, *mailx* will read the message and store it in a temporary file. Commands may be entered by beginning a line with the tilde (~) escape character followed by a single command letter and optional arguments. See **TILDE ESCAPES** for a summary of these commands.

At any time, the behavior of *mailx* is governed by a set of *environment variables*. These are flags and valued parameters which are set and cleared via the *set* and *unset* commands. See **ENVIRONMENT VARIABLES** below for a summary of these parameters.

Recipients listed on the command line may be of three types: login names, shell commands, or alias groups. Login names may be any network address, including mixed network addressing. If mail is found to to undeliverable, an attempt is made to return it to the sender's *mailbox*. If the recipient name begins with a pipe symbol (|), the rest of the name is taken to be a shell command to pipe the message through. This provides an automatic interface with any program that reads the standard input, such as *lp*(1) for recording outgoing mail on paper. Alias groups are set by the alias command (see **COMMANDS**) and are lists of recipients of any type.

Regular commands are of the form

[**command**] [*msglist*] [*arguments*]

If no command is specified in *command mode*, **print** is assumed. In *input mode*, commands are recognized by the escape character, and lines not treated as commands are taken as input for the message.

Each message is assigned a sequential number, and there is at any time the notion of a current message, marked by a right angle bracket (>) in the header summary. Many commands take an optional list of messages (*msglist*) to operate on. The default for *msglist* is the current message. A *msglist* is a list of message identifiers separated by spaces, which may include:

n	Message number **n**.
.	The current message.
^	The first undeleted message.
$	The last message.
*****	All messages.
n–m	An inclusive range of message numbers.
user	All messages from **user**.
/string	All messages with **string** in the subject line (case ignored).
:c	All messages of type *c*, where *c* is one of:

d	deleted messages
n	new messages
o	old messages
r	read messages
u	unread messages

Note that the context of the command determines whether this type of message specification makes sense.

Other arguments are usually arbitrary strings whose usage depends on the command involved. File names, where expected, are expanded via the normal shell conventions (see *sh*(1)). Special characters are recognized by certain commands and are documented with the commands below.

At start-up time, *mailx* tries to execute commands from the optional system-wide file (**/usr/lib/mailx/mailx.rc**) to initialize certain parameters, then from a private start-up file (**$HOME/.mailrc**) for personalized variables. With the exceptions noted below, regular commands are legal inside start-up files. The most common use of a start-up file is to set up initial display options and alias lists. The following commands are not legal in the start-up file: **!**, Copy, edit, followup, Followup, **hold**, **mail**, **preserve**, reply, Reply, **shell**, and visual. An error in the start-up file causes the remaining lines in the file to be ignored. The **.mailrc** file is optional, and must be constructed locally.

COMMANDS

The following is a complete list of *mailx* commands:

!*shell-command*
Escape to the shell. See *SHELL* (**ENVIRONMENT VARIABLES**).

comment
NULL command (comment). This may be useful in **.mailrc** files.

=
Print the current message number.

?
Prints a summary of commands.

alias *alias name ...*
group *alias name ...*
Declare an alias for the given names. The names will be substituted when *alias* is used as a recipient. Useful in the **.mailrc** file.

alternates *name ...*
Declares a list of alternate names for your login. When responding to a message, these names are removed from the list of recipients for the response. With no arguments, **alternates** prints the current list of alternate names. See also *allnet* (**ENVIRONMENT VARIABLES**).

cd [*directory*]
chdir [*directory*]
Change directory. If *directory* is not specified, $HOME is used.

copy [*filename*]
copy [*msglist*] *filename*
Copy messages to the file without marking the messages as saved. Otherwise, equivalent to the save command.

Copy [*msglist*]
Save the specified messages in a file whose name is derived from the author of the message to be saved, without marking the messages as saved. Otherwise, equivalent to the Save command.

delete [*msglist*]
Delete messages from the *mailbox*. If *autoprint* is set, the next message after the last one deleted is printed (see **ENVIRONMENT VARIABLES**).

discard [*header-field ...*]
ignore [*header-field ...*]
Suppresses printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are "status" and "cc." The fields are included when the message is saved. The Print and Type commands override this command.

dp [*msglist*]
dt [*msglist*]
Delete the specified messages from the *mailbox* and print the next message after the last one deleted. Roughly equivalent to a delete command followed by a print command.

echo *string ...*
Echo the given strings (like *echo*(1)).

edit [*msglist*]
Edit the given messages. The messages are placed in a temporary file and the EDITOR variable is used to get the name of the editor (see **ENVIRONMENT VARIABLES**). Default editor is *ed*(1).

exit
xit
Exit from *mailx*, without changing the *mailbox*. No messages are saved in the *mbox* (see also **quit**).

file [*filename*]
folder [*filename*]
Quit from the current file of messages and read in the specified file. Several special characters are recognized when used as file names, with the following substitutions:

%	the current *mailbox*.
%user	the *mailbox* for user.
#	the previous file.
&	the current *mbox*.

Default file is the current *mailbox*.

folders
Print the names of the files in the directory set by the *folder* variable (see **ENVIRONMENT VARIABLES**).

followup [*message*]
Respond to a message, recording the response in a file whose name is derived from the author of the message. Overrides the *record* variable, if set. See also the **Followup**, **Save**, and **Copy** commands and *outfolder* (**ENVIRONMENT VARIABLES**).

Followup [*msglist*]
Respond to the first message in the *msglist*, sending the message to the author of each message in the *msglist*. The subject line is taken from the first message and the response is recorded in a file whose name is derived from the author of the first message. See also the **followup**, **Save**, and **Copy** commands and *outfolder* (**ENVIRONMENT VARIABLES**).

from [*msglist*]
Prints the header summary for the specified messages.

group *alias name ...*
alias *alias name ...*
Declare an alias for the given names. The names will be substituted when *alias* is used as a recipient. Useful in the **.mailrc** file.

headers [*message*]
Prints the page of headers which includes the message specified. The *screen* variable sets the number of headers per page (see **ENVIRONMENT VARIABLES**). See also the **z** command.

help
Prints a summary of commands.

hold [*msglist*]
preserve [*msglist*]
Holds the specified messages in the *mailbox*.

if *s* | *r*
mail-commands
else
mail-commands
endif
Conditional execution, where *s* will execute following *mail-commands*, up to an else or endif, if the program is in *send* mode, and *r* causes the *mail-commands* to be executed only in *receive* mode. Useful in the **.mailrc** file.

ignore *header-field* ...
discard *header-field* ...
Suppresses printing of the specified header fields when displaying messages on the screen. Examples of header fields to ignore are "status" and "cc." All fields are included when the message is saved. The **Print** and **Type** commands override this command.

list
Prints all commands available. No explanation is given.

mail *name* ...
Mail a message to the specified users.

Mail *name*
Mail a message to the specified user and record a copy of it in a file named after that user.

mbox [*msglist*]
Arrange for the given messages to end up in the standard *mbox* save file when *mailx* terminates normally. See *MBOX* (**ENVIRONMENT VARIALBLES**) for a description of this file. See also the exit and quit commands.

next [*message*]
Go to next message matching *message*. A *msglist* may be specified, but in this case the first valid message in the list is the only one used. This is useful for jumping to the next message from a specific user, since the name would be taken as a command in the absence of a real command. See the discussion of *msglists* above for a description of possible message specifications.

pipe [*msglist*] [*shell-command*]
| [*msglist*] [*shell-command*]
Pipe the message through the given *shell-command*. The message is treated as if it were read. If no arguments are given, the current message is piped through the command specified by the value of the *cmd* variable. If the *page* variable is set, a form feed character is inserted after each message (see **ENVIRONMENT VARIALBLES**).

preserve [*msglist*]
hold [*msglist*]
Preserve the specified messages in the *mailbox*.

Print [*msglist*]
Type [*msglist*]
Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the ignore command.

print [*msglist*]

type [*msglist*]
Print the specified messages. If *crt* is set, the messages longer than the number of lines specified by the *crt* variable are paged through the command specified by the *PAGER* variable. The default command is *pg*(1) (see **ENVIRONMENT VARIABLES**).

quit
Exit from *mailx*, storing messages that were read in *mbox* and unread messages in the *mailbox*. Messages that have been explicitly saved in a file are deleted.

Reply [*msglist*]
Respond [*msglist*]
Send a response to the author of each message in the *msglist*. The subject line is taken from the first message. If *record* is set to a file name, the response is saved at the end of that file (see **ENVIRONMENT VARIABLES**).

reply [*message*]
respond [*message*]
Reply to the specified message, including all other recipients of the message. If *record* is set to a file name, the response is saved at the end of that file (see **ENVIRONMENT VARIABLES**).

Save [*msglist*]
Save the specified messages in a file whose name is derived from the author of the first message. The name of the file is taken to be the author's name with all network addressing stripped off. See also the Copy, followup, and Followup commands and *outfolder* (**ENVIRONMENT VARIABLES**).

save [*filename*]
save [*msglist*] *filename*
Save the specified messages in the given file. The file is created if it does not exist. The message is deleted from the *mailbox* when *mailx* terminates unless *keepsave* is set (see also **ENVIRONMENT VARIABLES** and the exit and quit commands).

set
set *name*
set *name=string*
set *name=number*
Define a variable called *name*. The variable may be given a **NULL**, string, or numeric value. Set by itself prints all defined variables and their values. See **ENVIRONMENT VARIABLES** for detailed descriptions of the *mailx* variables.

shell
Invoke an interactive shell (see also *SHELL* (**ENVIRONMENT VARIABLES**)).

size [*msglist*]
Print the size in characters of the specified messages.

source *filename*
Read commands from the given file and return to command mode.

top [*msglist*]
Print the top few lines of the specified messages. If the *toplines* variable is set, it is taken as the number of lines to print (see **ENVIRONMENT VARIABLES**). The default is 5.

touch [*msglist*]
Touch the specified messages. If any message in *msglist* is not specifically saved in a file, it will be placed in the *mbox*, or the file specified in the *MBOX* environment variable, upon normal termination. See exit and quit.

Type [*msglist*]
Print [*msglist*]
Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the ignore command.

type [*msglist*]
print [*msglist*]
Print the specified messages. If *crt* is set, the messages longer than the number of lines specified by the *crt* variable are paged through the command specified by the *PAGER* variable. The default command is *pg*(1) (see **ENVIRONMENT VARIABLES**).

undelete [*msglist*]
Restore the specified deleted messages. Will only restore messages deleted in the current mail session. If *autoprint* is set, the last message of those restored is printed (see **ENVIRONMENT VARIABLES**).

unset *name* ...
Causes the specified variables to be erased. If the variable was imported from the execution environment (i.e., a shell variable) then it cannot be erased.

version
Prints the current version and release date.

visual [*msglist*]
Edit the given messages with a screen editor. The messages are placed in a temporary file and the *VISUAL* variable is used to get the name of the editor (see **ENVIRONMENT VARIABLES**).

write [*msglist*] *filename*
Write the given messages on the specified file, minus the header and trailing blank line. Otherwise, equivalent to the save command.

xit
exit
Exit from *mailx*, without changing the *mailbox*. No messages are saved in the *mbox* (see also quit).

z[+ | -]
Scroll the header display forward or backward one screen–full. The number of headers displayed is set by the *screen* variable (see **ENVIRONMENT VARIABLES**).

TILDE ESCAPES

The following commands may be entered only from *input mode*, by beginning a line with the tilde escape character (~). See *escape* (**ENVIRONMENT VARIABLES**) for changing this special character.

~ ! *shell-command*
Escape to the shell.

~ .
Simulate end of file (terminate message input).

~ : *mail-command*
~ _ *mail-command*
Perform the command-level request. Valid only when sending a message while reading mail.

~ ?
Print a summary of tilde escapes.

~ A
Insert the autograph string "Sign" into the message (see **ENVIRONMENT VARIABLES**).

~ a
Insert the autograph string "sign" into the message (see **ENVIRONMENT VARIABLES**).

~ b *name* ...
Add the *names* to the blind carbon copy (Bcc) list.

~ c *name* ...
Add the *names* to the carbon copy (Cc) list.

~ d
Read in the *dead.letter* file. See *DEAD* (**ENVIRONMENT VARIABLES**) for a description of this file.

~ e
Invoke the editor on the partial message. See also *EDITOR* (**ENVIRONMENT VARIABLES**).

~ f [*msglist*]
Forward the specified messages. The messages are inserted into the message without alteration.

~ h
Prompt for Subject line and To, Cc, and Bcc lists. If the field is displayed with an initial value, it may be edited as if you had just typed it.

~ i *string*
Insert the value of the named variable into the text of the message. For example, **~A** is equivalent to '**~i** Sign.' Environment variables set and exported in the shell are also accessible by ~i.

~ m [*msglist*]
Insert the specified messages into the letter, shifting the new text to the right one tab stop. Valid only when sending a message while reading mail.

~ p
Print the message being entered.

~ q
Quit from input mode by simulating an interrupt. If the body of the message is not **NULL**, the partial message is saved in *dead.letter*. See *DEAD* (**ENVIRONMENT VARIABLES**) for a description of this file.

~ r *filename*
~ ~< *filename*
~ ~< !*shell-command*
Read in the specified file. If the argument begins with an exclamation point (!), the rest of the string is taken as an arbitrary shell command and is executed, with the standard output inserted into the message.

~ s *string* ...
Set the subject line to *string*.

~ t *name* ...
Add the given *names* to the To list.

~ v
Invoke a preferred screen editor on the partial message. See also *VISUAL* (**ENVIRONMENT VARIABLES**).

~ w *filename*
Write the partial message onto the given file, without the header.

~ x
Exit as with ~q except the message is not saved in *dead.letter*.

~ | *shell-command*
Pipe the body of the message through the given *shell-command*. If the *shell-command* returns a successful exit status, the output of the command replaces the message.

ENVIRONMENT VARIABLES

The following are environment variables taken from the execution environment and are not alterable within *mailx*:

HOME=*directory*
The user's base of operations.

MAILRC=*filename*
The name of the start-up file. Default is **$HOME/.mailrc**.

The following variables are internal *mailx* variables. They may be imported from the execution environment or set via the **set** command at any time. The **unset** command may be used to erase variables.

addsopt
Enabled by default. If **/bin/mail** is not being used as the deliverer, **noaddsopt** should be specified. (See **WARNING**.)

allnet
All network names whose last component (login name) match are treated as identical. This causes the *msglist* message specifications to behave similarly. Default is **noallnet**. See also the **alternates** command and the *metoo* variable.

append
Upon termination, append messages to the end of the *mbox* file instead of prepending them. Default is **noappend**.

askcc
Prompt for the Cc list after message is entered. Default is **noaskcc**.

asksub
Prompt for subject if it is not specified on the command line with the **-s** option. Enabled by default.

autoprint
Enable automatic printing of messages after **delete** and **undelete** commands. Default is **noautoprint**.

bang
Enable the special-casing of exclamation points (!) in shell escape command lines as in *vi*(1). Default is **nobang**.

cmd=*shell-command*
Set the default command for the **pipe** command. No default value.

conv=*conversion*
Convert uucp addresses to the specified address style. The only valid conversion now is *internet*, which requires a mail delivery program conforming to the RFC822 standard for electronic mail addressing. Conversion is disabled by default. See also *sendmail* and the **-U** command line option.

crt=*number*
Pipe messages having more than *number* lines through the command specified by the value of the *PAGER* variable (*pg*(1) by default). Disabled by default.

DEAD=*filename*
The name of the file in which to save partial letters in case of untimely interrupt. Default is **$HOME/dead.letter**.

debug
Enable verbose diagnostics for debugging. Messages are not delivered. Default is **nodebug**.

dot
Take a period on a line by itself during input from a terminal as end-of-file. Default is **nodot**.

EDITOR=*shell-command*
The command to run when the edit or ~e command is used. Default is *ed*(1).

escape=*c*
Substitute *c* for the ~ escape character. Takes effect with next message sent.

folder=*directory*
The directory for saving standard mail files. User-specified file names beginning with a plus (+) are expanded by preceding the file name with this directory name to obtain the real file name. If *directory* does not start with a slash (/), $HOME is prepended to it. In order to use the plus (+) construct on a *mailx* command line, folder must be an exported *sh* environment variable. There is no default for the folder variable. See also *outfolder* below.

header
Enable printing of the header summary when entering *mailx*. Enabled by default.

hold
Preserve all messages that are read in the *mailbox* instead of putting them in the standard *mbox* save file. Default is **nohold**.

ignore
Ignore interrupts while entering messages. Handy for noisy dial-up lines. Default is **noignore**.

ignoreeof
Ignore end-of-file during message input. Input must be terminated by a period (.) on a line by itself or by the ~. command. Default is **noignoreeof**. See also *dot* above.

keep
When the *mailbox* is empty, truncate it to zero length instead of removing it. Disabled by default.

keepsave
Keep messages that have been saved in other files in the *mailbox* instead of deleting them. Default is **nokeepsave**.

MBOX=*filename*
The name of the file to save messages which have been read. The xit command overrides this function, as does saving the message explicitly in another file. Default is **$HOME/mbox**.

metoo
If your login appears as a recipient, do not delete it from the list. Default is **nometoo**.

LISTER=*shell-command*
The command (and options) to use when listing the contents of the *folder* directory. The default is *ls*(1).

onehop
When responding to a message that was originally sent to several recipients, the other recipient addresses are normally forced to be relative to the originating author's machine for the response. This flag disables alteration of the recipients' addresses, improving efficiency in a network where all machines can send directly to all other machines (i.e., one hop away).

outfolder
Causes the files used to record outgoing messages to be located in the directory specified by the *folder* variable unless the pathname is absolute. Default is **nooutfolder**. See *folder* above and the **Save**, **Copy**, **followup**, and **Followup** commands.

page
Used with the **pipe** command to insert a form feed after each message sent through the pipe. Default is **nopage**.

PAGER=*shell-command*
The command to use as a filter for paginating output. This can also be used to specify the options to be used. Default is *pg*(1).

prompt=*string*
Set the *command mode* prompt to *string*. Default is "? ".

quiet
Refrain from printing the opening message and version when entering *mailx*. Default is **noquiet**.

record=*filename*
Record all outgoing mail in *filename*. Disabled by default. See also *outfolder* above.

save
Enable saving of messages in *dead.letter* on interrupt or delivery error. See *DEAD* for a description of this file. Enabled by default.

screen=*number*
Sets the number of lines in a screen-full of headers for the **headers** command.

sendmail=*shell-command*
Alternate command for delivering messages. Default is */bin/rmail*(1).

sendwait
Wait for background mailer to finish before returning. Default is **nosendwait**.

SHELL=*shell-command*
The name of a preferred command interpreter. Default is *sh*(1).

showto
When displaying the header summary and the message is from you, print the recipient's name instead of the author's name.

sign=*string*
The variable inserted into the text of a message when the **~a** (autograph) command is given. No default (see also ~i (**TILDE ESCAPES**)).

Sign=*string*
The variable inserted into the text of a message when the **~A** command is given. No default (see also ~i (**TILDE ESCAPES**)).

toplines=*number*
The number of lines of header to print with the **top** command. Default is 5.

VISUAL=*shell-command*
The name of a preferred screen editor. Default is *vi*(1).

FILES

$HOME/.mailrc	personal start-up file
$HOME/mbox	secondary storage file
/usr/mail/*	post office directory
/usr/lib/mailx/mailx.help*	help message files
/usr/lib/mailx/mailx.rc	optional global start-up file
/tmp/R[emqsx]*	temporary files

SEE ALSO
ls(1), mail(1), pg(1).

WARNINGS
The **–h**, **–r** and **–U** options can be used only if *mailx* is built with a delivery program other than **/bin/mail**.

BUGS
Where *shell-command* is shown as valid, arguments are not always allowed. Experimentation is recommended.

Internal variables imported from the execution environment cannot be **unset**.

The full internet addressing is not fully supported by *mailx*. The new standards need some time to settle down.

Attempts to send a message having a line consisting only of a "." are treated as the end of the message by *mail*(1) (the standard mail delivery program).

MAKE(1)

NAME

make – maintain, update, and regenerate groups of programs

SYNOPSIS

make [**–f** makefile] [**–p**] [**–i**] [**–k**] [**–s**] [**–r**] [**–n**] [**–b**] [**–e**] [**–t**] [**–q**] [names]

DESCRIPTION

The *make* command allows the programmer to maintain, update, and regenerate groups of computer programs. The following is a brief description of all options and some special names:

–f *makefile* Description file name. *makefile* is assumed to be the name of a description file.

–p Print out the complete set of macro definitions and target descriptions.

–i Ignore error codes returned by invoked commands. This mode is entered if the fake target name **.IGNORE** appears in the description file.

–k Abandon work on the current entry if it fails, but continue on other branches that do not depend on that entry.

–s Silent mode. Do not print command lines before executing. This mode is also entered if the fake target name **.SILENT** appears in the description file.

–r Do not use the built-in rules.

–n No execute mode. Print commands, but do not execute them. Even lines beginning with an @ are printed.

–b Compatibility mode for old makefiles.

–e Environment variables override assignments within makefiles.

–t Touch the target files (causing them to be up-to-date) rather than issue the usual commands.

–q Question. The *make* command returns a zero or non-zero status code depending on whether the target file is or is not up-to-date.

.DEFAULT

If a file must be made but there are no explicit commands or relevant built-in rules, the commands associated with the name **.DEFAULT** are used if it exists.

.PRECIOUS

Dependents of this target will not be removed when quit or interrupt are hit.

.SILENT Same effect as the **–s** option.

.IGNORE Same effect as the **–i** option.

make executes commands in *makefile* to update one or more target *names*. *Name* is typically a program. If no **–f** option is present, **makefile**, **Makefile**, and the Source Code Control System(SCCS) files **s.makefile**, and **s.Makefile** are tried in order. If *makefile* is –, the standard input is taken. More than one **–** *makefile* argument pair may appear.

make updates a target only if its dependents are newer than the target. All prerequisite files of a target are added recursively to the list of targets. Missing files are deemed to be out-of-date.

makefile contains a sequence of entries that specify dependencies. The first line of an entry is a blank-separated, non-null list of targets, then a :, then a (possibly null) list of prerequisite files or dependencies. Text following a ; and all following lines that begin with a tab are shell commands to be executed to update the target. The first non-empty line that does not begin with a tab or #

begins a new dependency or macro definition. Shell commands may be continued across lines with the <backslash> <new-line> sequence. Everything printed by make (except the initial tab) is passed directly to the shell as is. Thus,

```
echo a\
b
```

will produce

```
ab
```

exactly the same as the shell would.

A sharp (#) in column one starts a comment. A sharp anywhere else starts a comment only if the text is not part of a shell command sequence. Comments continue to the next unescaped newline. A sharp that would otherwise start a comment can be escaped by doubling (##).

The following *makefile* says that **pgm** depends on two files **a.o** and **b.o**, and that they in turn depend on their corresponding source files (**a.c** and **b.c**) and a common file **incl.h**:

```
pgm: a.o b.o
        cc a.o b.o -o pgm
a.o: incl.h a.c
        cc -c a.c
b.o: incl.h b.c
        cc -c b.c
```

Command lines are executed one at a time, each by its own shell. The **SHELL** environment variable can be used to specify which shell *make* should use to execute commands. The default is */bin/sh*. The first one or two characters in a command can be the following: **-**, **@**, **-@**, or **@-**. If **@** is present, printing of the command is suppressed. If **-** is present, *make* ignores an error. A line is printed when it is executed unless the **-s** option is present, or the entry **.SILENT:** is in *makefile*, or unless the initial character sequence contains a **@**. The **-n** option specifies printing without execution; however, if the command line has the string **$(MAKE)** in it, the line is always executed (see discussion of the **MAKEFLAGS** macro under *Environment*). The **-t** (touch) option updates the modified date of a file without executing any commands.

Commands returning non-zero status normally terminate *make*. If the **-i** option is present, or the entry **.IGNORE:** appears in *makefile*, or the initial character sequence of the command contains **-**. the error is ignored. If the **-k** option is present, work is abandoned on the current entry, but continues on other branches that do not depend on that entry.

The **-b** option allows old makefiles (those written for the old version of *make*) to run without errors.

Interrupt and quit cause the target to be deleted unless the target is a dependent of the special name **.PRECIOUS**.

Environment

The environment is read by *make*. All variables are assumed to be macro definitions and processed as such. The environment variables are processed before any makefile and after the internal rules; thus, macro assignments in a makefile override environment variables. The **-e** option causes the environment to override the macro assignments in a makefile. Suffixes and their associated rules in the makefile will override any identical suffixes in the built-in rules.

The **MAKEFLAGS** environment variable is processed by *make* as containing any legal input option (except **-f** and **-p**) defined for the command line. Further, upon invocation, *make* "invents" the variable if it is not in the environment, puts the current options into it, and passes it on to invocations of commands. Thus, **MAKEFLAGS** always contains the current input options. This proves very useful for "super-makes". In fact, as noted above, when the **-n** option

is used, the command **$(MAKE)** is executed anyway; hence, one can perform a **make –n** recursively on a whole software system to see what would have been executed. This is because the **–n** is put in **MAKEFLAGS** and passed to further invocations of **$(MAKE)**. This is one way of debugging all of the makefiles for a software project without actually doing anything.

Include Files

If the string *include* appears as the first seven letters of a line in a *makefile*, and is followed by a blank or a tab, the rest of the line is assumed to be a file name and will be read by the current invocation, after substituting for any macros.

Macros

Entries of the form *string1 = string2* are macro definitions. *String2* is defined as all characters up to a comment character or an unescaped new-line. Subsequent appearances of $(*string1*[:*subst1*=[*subst2*]]) are replaced by *string2*. The parentheses are optional if a single character macro name is used and there is no substitute sequence. The optional :*subst1*=*subst2* is a substitute sequence. If it is specified, all non-overlapping occurrences of *subst1* in the named macro are replaced by *subst2*. Strings (for the purposes of this type of substitution) are delimited by blanks, tabs, new-line characters, and beginnings of lines. An example of the use of the substitute sequence is shown under *Libraries*.

Internal Macros

There are five internally maintained macros which are useful for writing rules for building targets.

$*** The macro **$* stands for the file name part of the current dependent with the suffix deleted. It is evaluated only for inference rules.

$@** The **$@ macro stands for the full target name of the current target. It is evaluated only for explicitly named dependencies.

$<** The **$< macro is only evaluated for inference rules or the **.DEFAULT** rule. It is the module which is out-of-date with respect to the target (i.e., the "manufactured" dependent file name). Thus, in the **.c.o** rule, the **$<** macro would evaluate to the **.c** file. An example for making optimized **.o** files from **.c** files is:

```
.c.o:
        cc -c -O $*.c
```

or:

```
.c.o:
        cc -c -O $<
```

$?** The **$? macro is evaluated when explicit rules from the makefile are evaluated. It is the list of prerequisites that are out-of-date with respect to the target; essentially, those modules which must be rebuilt.

$% The **$%** macro is only evaluated when the target is an archive library member of the form **lib(file.o)**. In this case, **$@** evaluates to **lib** and **$%** evaluates to the library member, **file.o**.

Four of the five macros can have alternative forms. When an upper case **D** or **F** is appended to any of the four macros, the meaning is changed to "directory part" for **D** and "file part" for **F**. Thus, **$(@D)** refers to the directory part of the string **$@**. If there is no directory part, **./** is generated. The only macro excluded from this alternative form is **$?**.

Suffixes

Certain names (for instance, those ending with **.o**) have inferable prerequisites such as **.c**, **.s**, etc. If no update commands for such a file appear in *makefile*, and if an inferable prerequisite exists, that prerequisite is compiled to make the target. In this case, *make* has inference rules which allow building files from other files by examining the suffixes and determining an appropriate

inference rule to use. The current default inference rules are:

```
.c .c~ .f .f~ .sh .sh~
.c.o .c.a .c~.o .c~.c .c~.a
.f.o .f.a .f~.o .f~.f .f~.a
.h~.h .s.o .s~.o .s~.s .s~.a .sh~.sh
.l.o .l.c .l~.o .l~.l .l~.c
.y.o .y.c .y~.o .y~.y .y~.c
```

The internal rules for *make* are contained in the source file **rules.c** for the *make* program. These rules can be locally modified. To print out the rules compiled into the *make* on any machine in a form suitable for recompilation, the following command is used:

```
make -fp - 2> /dev/null < /dev/null
```

A tilde in the above rules refers to an SCCS file [see *sccsfile*(4)]. Thus, the rule **.c~.o** would transform an SCCS C source file into an object file (**.o**). Because the **s.** of the SCCS files is a prefix, it is incompatible with *make*'s suffix point of view. Hence, the tilde is a way of changing any file reference into an SCCS file reference.

A rule with only one suffix (i.e., **.c:**) is the definition of how to build *x* from *x*.c. In effect, the other suffix is null. This is useful for building targets from only one source file (e.g., shell procedures, simple C programs).

Additional suffixes are given as the dependency list for **.SUFFIXES**. Order is significant; the first possible name for which both a file and a rule exist is inferred as a prerequisite. The default list is:

.SUFFIXES: .o .c .c~ .y .y~ .l .l~ .s .s~ .sh .sh~ .h .h~ .f .f~

Here again, the above command for printing the internal rules will display the list of suffixes implemented on the current machine. Multiple suffix lists accumulate; **.SUFFIXES:** with no dependencies clears the list of suffixes.

Inference Rules

The first example can be done more briefly.

```
pgm: a.o b.o
        cc a.o b.o -o pgm
a.o b.o: incl.h
```

This is because *make* has a set of internal rules for building files. The user may add rules to this list by simply putting them in the *makefile*.

Certain macros are used by the default inference rules to permit the inclusion of optional matter in any resulting commands. For example, **CFLAGS**, **LFLAGS**, and **YFLAGS** are used for compiler options to *cc*(1), *lex*(1), and *yacc*(1), respectively. Again, the previous method for examining the current rules is recommended.

The inference of prerequisites can be controlled. The rule to create a file with suffix **.o** from a file with suffix **.c** is specified as an entry with **.c.o:** as the target and no dependents. Shell commands associated with the target define the rule for making a **.o** file from a **.c** file. Any target that has no slashes in it and starts with a dot is identified as a rule and not a true target.

Libraries

If a target or dependency name contains parentheses, it is assumed to be an archive library, the string within parentheses referring to a member within the library. Thus **lib(file.o)** and **$(LIB)(file.o)** both refer to an archive library which contains **file.o**. (This assumes the **LIB** macro has been previously defined.) The expression **$(LIB)(file1.o file2.o)** is not legal. Rules pertaining to archive libraries have the form *.XX*.**a** where the *XX* is the suffix from which the archive member is to be made. An unfortunate byproduct of the current implementation requires

the *XX* to be different from the suffix of the archive member. Thus, one cannot have **lib(file.o)** depend upon **file.o** explicitly. The most common use of the archive interface follows. Here, we assume the source files are all C type source:

```
lib:    lib(file1.o) lib(file2.o) lib(file3.o)
        @echo lib is now up-to-date
.c.a:
        $(CC) -c $(CFLAGS) $<
        $(AR) $(ARFLAGS) $@ $*.o
        rm -f $*.o
```

In fact, the **.c.a** rule listed above is built into *make* and is unnecessary in this example. A more interesting, but more limited example of an archive library maintenance construction follows:

```
lib:    lib(file1.o) lib(file2.o) lib(file3.o)
        $(CC) -c $(CFLAGS) $(?:.o=.c)
        $(AR) $(ARFLAGS) lib $?
        rm $?  @echo lib is now up-to-date
.c.a:;
```

Here the substitution mode of the macro expansions is used. The **$?** list is defined to be the set of object file names (inside **lib**) whose C source files are out-of-date. The substitution mode translates the **.o** to **.c**. (Unfortunately, one cannot as yet transform to **.c~**; however, this may become possible in the future.) Note also, the disabling of the **.c.a:** rule, which would have created each object file, one by one. This particular construct speeds up archive library maintenance considerably. This type of construct becomes very cumbersome if the archive library contains a mix of assembly programs and C programs.

Parallel operation

Make by default builds targets serially. To build selected targets in parallel, the **PARALLEL** environment variable must be set to the list of permissible targets. A target being in this list does not guarantee it will actually be built in parallel with anything else; however, a target not in this list will never build concurrently with any other target (ie, *make* will drop into fully serial operation for that build).

The target names in **PARALLEL** may be intermixed with patterns contains shell-like wildcards that match many targets. For example, a below pattern allows all targets with a one character suffix to run in parallel:

```
PARALLEL = *.?
```

This pattern is particularly recommended, since it automatically tends to prevents virtual targets (`all`, `clobber`, recursive *make* calls, etc) from running in parallel, which is usually what is desired.

The wildcard rules are similar to that of *sh*(1), except that the characters appearing within brackets follow the rules of *regex*(3x). Example: "`*.[^a]`" maps to all single-suffix targets whose names do not end in "`a`".

Use of wildcards in **PARALLEL** sometimes causes a few targets to be included which the user did not want. Rather than complicating the expression, the user can simply exclude the undesirables by setting the environment variable **SERIAL** to their names. More formally, **SERIAL** removes targets from the parallel list after it is created. It follows the same syntax and wildcard rules as **PARALLEL**.

The target names that *make* supplies to the pattern matcher do not include archive library member names. For example, "`lib.a(x.o)`" is reduced to '`lib.a`". The user must make the same reduction to the pattern and target names he specifies.

Make uses the dependency rules it already knows about to avoid spinning off a target build until the targets' dependents have finished. For example, the build that makes gram.o from gram.c will stall until the build that makes gram.c from gram.y completes. For many makefiles, this restriction is sufficient to avoid collision.

Other targets have serialization restrictions beyond that imposed by the dependency rules. For example, only one target build using *ar*(1) should run at a time. Since it is not practical or safe to have *make* examine targets for these rules, the environment variable **MUTEX** has been introduced to provide the necessary mutual exclusion facility.

MUTEX identifies one or more sets of targets, of which for each set only one target is allowed to run at a time. Each set may be thought of as being protected by a Dikstra semaphore; when a target begins building it locks the semaphores of the sets it is a member of, and when it completes it unlocks them. Thus, **MUTEX** may be used to provide a minimal degree of interlocking without forcing *make* to drop into full serial mode. The following example might be typical of its use:

```
MUTEX=*.a, *gram.? *y.tab.?, *lex.? *lex.yy.?
```

The comma is used to separate sets of targets. Within each set, only one target can be building. If *make* encounters another runnable target in the same set, *make* will stall until the previous set member completes. An English translation of the intent of this example might be: "run at most one *yacc* build, one *lex* build, and one build whose destination is an archive file. In addition, if a target is both, say, an archive file and a *yacc* user, then stall until both categories are free."

The **NPROC** variable lets the user place a cap on the number of concurrent builds which take place. If **NPROC** is not specified or is the null string, then the cap is a dynamic value based upon the current system load, as determined by *getsysload*(2). **NPROC** may be a constant in any base, or an arbitrarily complex integer expression of such constants.

If the variable **COLLATE** is set **=YES**, then the standard output and standard error of each target build will not be intermixed with the output of other target builds.

The default value for **NPROC**, **PARALLEL**, **SERIAL**, **MUTEX**, and **COLLATE** is the null string.

FILES

[Mm]akefile and s.[Mm]akefile
/bin/sh
/tmp/mk*

SEE ALSO

cc(1), cd(1), lex(1), sh(1), yacc(1), getsysload(2), printf(3S), sccsfile(4).

NOTES

Some commands return non-zero status inappropriately; use **-i** to overcome the difficulty.

CAVEATS AND BUGS

File names with the characters = : @ will not work. Commands that are directly executed by the shell, notably *cd*(1), are ineffectual across new-lines in *make*. The syntax **(lib(file1.o file2.o file3.o)** is illegal. You cannot build **lib(file.o)** from **file.o**. The macro **$(a:.o=.c~)** does not work. Named pipes are not handled well.

Parallel operation

Make should not stall on a build, it should skip over it and look for other builds it can spin off, and come back later when no other work can be done. Users annoyed by the lack of this feature most likely can minimize it by reordering their makefiles appropriately.

It is possible to come up with pathological makefiles in which stalling above and beyond that described here will occur.

MUTEX supports only 32 pattern sets. If more than 32 sets are present, then the excess sets are appended to the first 32 modulo 32, ie, set 32 is appended to set 0, set 33 appended to set 1, and so on. Other than providing less parallel execution than expected, this behavior is harmless.

Except when **COLLATE=YES**, the standard and error output of concurrently running target builds will be intermixed. Surprisingly, intermixed output often remains readable.

The **COLLATE** option has some problems. Its implementation is expensive; a target build's standard output and standard error are spooled to files, which are printed when that build completes. The user sees no output until a build is done, then he gets everything. Target builds that test the output device (eg, use *isatty*(3C)) might behave differently. Broken target builds that generate runaway output may fill up a substantial part of the file system and take an unusually long time to run, without giving any indication of the failure to the user. And target builds which do not wait for their own child processes to exit can result in the terminal output of those children being lost.

NAME

makekey – generate encryption key

SYNOPSIS

/usr/lib/makekey

DESCRIPTION

Makekey improves the usefulness of encryption schemes that depend on a key by increasing the amount of time required to search the key space. It reads 10 bytes from its standard input and writes 13 bytes on its standard output. The output depends on the input in a way intended to be difficult to compute (i.e., to require a substantial fraction of a second).

The first eight input bytes (the *input key*) can be arbitrary ASCII characters. The last two (the *salt*) are best chosen from the set of digits, ., /, and upper and lowercase letters. The salt characters are repeated as the first two characters of the output. The remaining 11 output characters are chosen from the same set as the salt and constitute the *output key*.

The transformation performed is essentially the following: the salt is used to select one of 4,096 cryptographic machines, all based on the National Bureau of Standards DES algorithm but broken in 4,096 different ways. Using the *input key* as key, a constant string is fed into the machine and recirculated a number of times. The 64 bits that come out are distributed into the 66 *output key* bits in the result.

makekey is intended for programs that perform encryption (such as *ed*(1) and *crypt*(1)). Usually its input and output are pipes.

SEE ALSO

crypt(1), ed(1), passwd(4).

NAME

man – print entries in this manual

SYNOPSIS

man [options] [section] title [[section] title] ...
man -k search-pattern

DESCRIPTION

man locates and prints the entry of this manual named *title* in the specified *section*. (For historical reasons, the word "page" is often used as a synonym for "entry" in this context.) The *title* is entered in lowercase characters. The *section* number may not have a letter suffix. If no *section* is specified, the whole manual is searched for *title* and all occurrences of it are printed. *title* may contain shell metacharacters. All titles specified on the command line are expanded according to translation files in the /usr/lib/xlate directory (see **man**(4)).

Valid *options* and their meanings are:

–T*term* Print the entry as appropriate for terminal type *term*.

–w Print on the standard output only the pathnames of the entries, relative to **/usr/catman**, or to the directory specified with the **–d** option.

–d*directory*
Search **directory** rather than **/usr/catman**.

–k Search the files **/usr/lib/xlate/*.apropos** for any line that matches the argument given as the *search-pattern*. For example,

man -k realtime

will find any manual page title which contains the string *realtime*.

–c Invoke *col*(1); note that *col*(1) is invoked automatically by *man* unless *term* is one of **300**, **300s**, **450**, **37**, **4000a**, **382**, **4014**, **tek**, and **1620**.

man examines the environment variable **$TERM** (see *environ*(5)) and attempts to select options that adapt the output to the terminal being used. The **–T***term* option overrides the value of **$TERM**; in particular, one should use **–Tlp** when sending the output of *man* to a line printer.

Section may be changed before each *title*.

As an example:

man man

would reproduce on the terminal this entry, as well as any other entries named *man* that may exist in other sections of the manual, e.g., *man*(4).

If the environment variable MANPAGER is non-null, all entries will be piped through the command named in this variable when being displayed.

FILES

/usr/catman/?_man/man[1-8][a-z]/*	
/usr/catman/?_man/man[1-8]/*	Preformatted manual entries
/usr/lib/xlate/*.alias	Translation files
/usr/lib/xlate/*.apropos	List of all manual page titles

SEE ALSO

man(4), environ(5), term(5).

NAME

mcs – manipulate the object file comment section

SYNOPSIS

mcs [*options*] *object-files*

DESCRIPTION

The *mcs* command manipulates the comment section in an object file. It is used to add to, delete, print, and compress the contents of the **.comment** section in a UNIX object file. If the object file is an archive, the file is treated as a set of individual object files. As a result, if the **–a** option is specified, the string is appended to the comment section of each archive element. The following options are available.

–a *string*
Appends *string* to the comment section of the object files.

–c
Compresses the contents of the comment section. All duplicate entries are removed. The order of the remaining entries is not disturbed.

–d
Deletes the contents of the comment section from the object file. Also removes the object-file comment-section header.

–n *name*
Specifies the name of the section to access. By default, *mcs* deals with the one named **.comment**. This option can be used to specify another section.

–p
Prints the contents of the comment section on the standard output. When more than one name is specified, tags each printed entry with the name of the file from which it was extracted.

–P
Prints the contents of the comment section, tagging each line with the name of the file from which it was extracted. The format used is "filename:string."

EXAMPLES

mcs -p file # Print *file*'s *comment* section.

mcs -a string file # Append *string* to *file*'s *comment* section

SEE ALSO

cpp(1), a.out(4).

MDEL(1)

NAME

mdel – delete a MSDOS file

SYNOPSIS

mdel [**-v**] msdosfile [msdosfiles...]

DESCRIPTION

Mdel deletes a file on a MSDOS formatted diskette. Sub directories are supported with either the '/' or '\' separator. The use of the '\' separator or wildcards will require the names to be enclosed in quotes to protect them from the shell.

Mdel will allow the following command line option:

v Verbose mode. Echo the file names as they are processed.

Mdel will ask for verification prior to removing a read–only file.

The environment variable MCWD may be used to establish a current working directory (relative to the MSDOS diskette).

The environment variable MFLOPPY must be set to the name of the floppy drive you wish to access. For example, to access floppy drive /dev/rdsk/m320_0hd2, type: MFLOPPY=/dev/rdsk/m320_0hd2;export MFLOPPY

SEE ALSO

mdir(1)

CAVEATS

Allows multiple arguments, which does not follow the MSDOS convention. Wildcards are only applied to filenames, not to directory names.

NAME

mdir – display a MSDOS directory

SYNOPSIS

mdir [**-w**] msdosdirectory

mdir [**-w**] msdosfile [msdosfiles...]

DESCRIPTION

Mdir displays the contents of a MSDOS formatted diskette. Sub directories are supported with either the '/' or '\' separator. The use of the '\' separator or wildcards will require the names to be enclosed in quotes to protect them from the shell.

Mdir will allow the following command line option:

w Wide output. This option will print the file names across the page without displaying the file size or creation date.

An error occurs if a component of the path is not a directory.

The environment variable MCWD may be used to establish a current working directory (relative to the MSDOS diskette).

The environment variable MFLOPPY must be set to the name of the floppy drive you wish to access. For example, to access floppy drive /dev/rdsk/m320_0hd2, type: MFLOPPY=/dev/rdsk/m320_0hd2;export MFLOPPY

SEE ALSO

mtype(1)

CAVEATS

Allows multiple arguments, which does not follow the MSDOS convention. Wildcards are only applied to filenames, not to directory names.

NAME

mesg – permit or deny messages

SYNOPSIS

mesg [**–n**] [**–y**]

DESCRIPTION

mesg with argument **n** forbids messages via *write*(1) by revoking non-user write permission on the user's terminal. *mesg* with argument **y** reinstates permission. All by itself, *mesg* reports the current state without changing it.

FILES

/dev/tty*

SEE ALSO

write(1).

DIAGNOSTICS

Exit status is 0 if messages are receivable, 1 if not, 2 on error.

NAME

mformat – format a MSDOS floppy disk

SYNOPSIS

mformat

DESCRIPTION

The *mformat* utility will format a high or low density floppy diskette. This initialization of the disk will remove all data from the disk, and it will not be able to be recovered.

The environment variable MFLOPPY must be set to the name of the floppy drive you wish to format. For example, to format floppy drive /dev/rdsk/m320_0hd2, type: MFLOPPY=/dev/rdsk/m320_0hd2;export MFLOPPY

MKDIR(1)

NAME

mkdir – make directories

SYNOPSIS

mkdir [**–m** *mode*] [**–p**] *dirname* ...

DESCRIPTION

mkdir creates the named directories in mode 777 (possibly altered by *umask*(1)).

Standard entries in a directory (e.g., the files **.**, for the directory itself, and **..**, for its parent) are made automatically. *mkdir* cannot create these entries by name. Creation of a directory requires write permission in the parent directory.

The owner ID and group ID of the new directories are set to the process's real user ID and group ID, respectively.

Two options apply to *mkdir*:

–m
allows users to specify the mode to be used for new directories. Choices for modes can be found in *chmod*(1).

–p
creates *dirname* by creating all the nonexisting parent directories first.

EXAMPLE

To create the subdirectory structure **ltr/jd/jan**, type:

mkdir –p ltr/jd/jan

SEE ALSO

sh(1), rm(1), umask(1), intro(2), mkdir(2).

DIAGNOSTICS

mkdir returns exit code **0** if all directories given in the command line were made successfully. Otherwise, it prints a diagnostic and returns nonzero. An error code is stored in **errno**.

NAME

mkshlib – create a shared library

SYNOPSIS

mkshlib **–s** specfil **–t** target [**–h** host] [**–n**] [**–L** dir ...] [**–q**] [**–U**]

DESCRIPTION

The *mkshlib* command builds both the host and target shared libraries. A shared library is similar in function to a normal, non-shared library, except that programs that link with a shared library will share the library code during execution whereas programs that link with a non-shared library will get their own copy of each library routine used.

The host shared library is an archive that is used to link-edit user programs with the shared library [see *ar*(4)]. A host shared library can be treated exactly like a non-shared library and should be included on *cc*(1) command lines in the usual way [see *cc*(1)]. Further, all operations that can be performed on an archive can also be performed on the host shared library.

The target shared library is an executable module that is bound into the user's address space during execution of a program using the shared library. The target shared library contains the code for all the routines in the library and must be fully resolved. The target will be brought into memory during execution of a program using the shared library, and subsequent processes that use the shared library will share the copy of code already in memory. The text of the target is always shared, but each process will get its own copy of the data.

The user interface to *mkshlib* consists of command line options and a shared library specification file. The shared library specification file describes the contents of the shared library.

The *mkshlib* command invokes other tools such as the archiver, *ar*(1), the assembler, *as*(1), and the link editor, *ld*(1). Tools are invoked through the use of *execvp* [see *exec*(2)], which searches directories in the user's PATH. Also, prefixes to *mkshlib* are parsed in the same manner as prefixes to the *cc*(1) command, and invoked tools are given the prefix, where appropriate. For example, **pfxmkshlib** will invoke **pfxld**.

The following command line options are recognized by *mkshlib*:

–s *specfil* Specifies the shared library specification file, *specfil*. This file contains the information necessary to build a shared library.

–t *target* Specifies the output filename of the target shared library being created. It is assumed that this file will be installed on the target machine at the location given in the specification file (see the **#target** directive below). If the **–n** option is used, then a new target shared library will not be generated.

–h *host* Specifies the output filename of the host shared library being created. If this option is not given, then the host shared library will not be produced.

–n Do not generate a new target shared library. This option is useful when producing only a new host shared library. The **–t** option must still be supplied since a version of the target shared library is needed to build the host shared library.

–L *dir* ... Change the algorithm of searching for the host shared libraries specified with the **#objects noload** directive to look in *dir* before looking in the default directories. The **–L** option can be specified multiple times on the command line in which case the directories given with the **–L** options are searched in the order given on the command line before the default directories.

–q Quiet warning messages. This option is useful when warning messages are expected but not desired.

–U Unrestricted mode. If this option is given, the branch table is not required, uninitialized common symbols are permitted, and certain warning messages are suppressed.

The shared library specification file contains all the information necessary to build both the host and target shared libraries. The contents and format of the specification file are given by the directives listed below.

All directives that can be followed by multi-line specifications are valid until the next directive or the end of the file.

#address *sectname address*

Specifies the start address, *address*, of section *sectname* for the target. This directive typically is used to specify the start addresses of the **.text** and **.data** sections. One **#address** per section name is valid. A **#address** directive must be given exactly once for the **.text** section and once for the **.data** section.

#target *pathname*

Specifies the absolute pathname, *pathname*, at which the target shared library will be installed on the target machine. The operating system uses this pathname to locate the shared library when executing **a.out** files that use this shared library. If the final character of *pathname* is "/", the target shared library name specified with the **–t** command line option will be appended to complete the pathname. This directive must be specified exactly once per specification file.

#branch

Specifies the start of the branch table specifications. The lines following this directive are taken to be branch table specification lines.

Branch table specification lines have the following format:

funcname < white space> *position*

where *funcname* is the name of the symbol given a branch table entry and *position* specifies the position of *funcname*'s branch table entry. *position* may be a single integer or a range of integers of the form *position1-position2*. Each *position* must be greater than or equal to one, the same position can not be specified more than once, and every position from one to the highest given position must be accounted for.

If a symbol is given more than one branch table entry by associating a range of positions with the symbol or by specifying the same symbol on more than one branch table specification line, then the symbol is defined to have the address of the highest associated branch table entry. All other branch table entries for the symbol can be thought of as "empty" slots and can be replaced by new entries in future versions of the shared library. Only functions should be given branch table entries, and those functions must be **external** symbols.

If the **–U** command line option is specified, the branch table specification may be omitted. Otherwise, this directive must be specified exactly once per shared library specification file.

#objects

The lines following this directive are taken to be the list of input object files in the order they are to be loaded into the target. The list simply consists of each pathname followed by a newline character. This list is also used to determine the input object files for the host shared library, but the order for the host is given by running the list through *lorder*(1) and *tsort*(1).

This directive must be specified exactly once per shared library specification file.

#objects noload

The **#objects noload** is followed by a list of host shared libraries. These libraries are searched in the order listed to resolve undefined symbols from the library being built. During the search it is considered an error if a non-shared version of a symbol is found before a shared version of the symbol.

Each name given is assumed to be a pathname to a host or an argument of the form **–lX**where **libX.a** is the name of a file in LIBDIR or LLIBDIR. This behavior is identical to that of *ld* , and the **–L** option can be used on the command line to specify other directories in which to locate these archives.

Note that if a host shared library is specified using **#objects noload**, any *cc* command that links to the shared library being built will need to specify that host also.

#hide linker [*]

This directive changes symbols that are normally **external** into **static** symbols, local to the library being created. A regular expression may be given [**sh**(1), **find**(1)], in which case all **external** symbols matching the regular expression are hidden; the **#export** directive (see below) can be used to counter this effect for specified symbols.

The optional "*" is equivalent to the directive

```
#hide linker
  *
```

and causes all **external** symbols to be made into **static** symbols.

All symbols specified in **#init** and **#branch** directives are assumed to be **external** symbols, and cannot be changed into **static** symbols using the **#hide** directive.

#export linker [*]

Symbols given in the **#export** directive are **external** symbols (global among files) that, because of a regular expression in a **#hide** directive, would otherwise have been made **static**. For example,

```
#hide linker *
#export linker
  one
  two
```

causes all symbols except *one*, *two*, and those used in **#branch** and **#init** entries to be tagged as **static** .

#init *object*

Specifies that the object file, *object*, requires initialization code. The lines following this directive are taken to be initialization specification lines.

Initialization specification lines have the following format:

ptr < white space> *import*

ptr is a pointer to the associated imported symbol, *import*, and must be defined in the current specified object file, *object*. The initialization code generated for each such line is of the form:

```
ptr = &import;
```

All initializations for a particular object file must be given once and multiple specifications of the same object file are not allowed.

#ident *string*

Specifies a string, *string*, to be included in the .comment section of the target shared library.

##

Specifies a comment. All information on the line beginning with **##** is ignored.

FILES

TEMPDIR/*	temporary files

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

LIBDIR	usually /lib
LLIBDIR	usually /usr/lib

SEE ALSO

ar(1), as(1), cc(1), ld(1), lorder(1), tsort(1), a.out(4), ar(4).

CAVEATS

The **–n** option cannot be used with the **#objects noload** directive.

If *mkshlib* is asked to create a host library and a host of that name already exists, *mkshlib* will update the host using **ar –ru**. This means that you should always remove the host before rebuilding whenever an object file previously included in the library is removed or renamed.

If the address specified with the **#address** directive is outside user space, the library build may look successful, but if you try to use it, it might not work.

NAME

mmd – make a MSDOS sub directory

SYNOPSIS

mmd [**-v**] msdosdirectory

DESCRIPTION

Mmd makes a new directory on a MSDOS formatted diskette. Sub directories are supported with either the '/' or '\' separator. The use of the '\' separator will require the directory name to be enclosed in quotes to protect it from the shell.

Mmd will allow the following command line option:

v Verbose mode. Display the new directory name if the name supplied is invalid.

Reasonable care is taken to create a valid MSDOS directory name. If an invalid name is specified, *mmd* will change the name (and display the new name if the verbose mode is set).

An error occurs if the directory already exists.

The environment variable MCWD may be used to establish a current working directory (relative to the MSDOS diskette).

The environment variable MFLOPPY must be set to the name of the floppy drive you wish to access. For example, to access floppy drive /dev/rdsk/m320_0hd2, type: MFLOPPY=/dev/rdsk/m320_0hd2;export MFLOPPY

SEE ALSO

mdir(1), mrd(1)

NAME

mnt, umnt – mount and dismount file system

SYNOPSIS

mnt [*options*] [*name* [*directory*]]

umnt [*options*] [*name*]

DESCRIPTION

mnt announces to the system that a removable file system is present on the special device. The *directory* must exist already; it becomes the name of the root of the newly mounted file system.

umnt announces to the system that the removable file system previously mounted on a special device is to be removed.

mnt (umnt) has an optional argument, *name*. This argument is used to search the **permissions** file to determine the real device to mount (unmount). The file is searched and when *name* matches either the *slice* or the *alias* entry on a line, the *slice* entry is then used as the special device to be mounted (unmounted).

If no name is specified by the options or by the *name* argument, then the alias **floppy** is used.

By convention *mount*(1M) and *umount*(1M) require root permission to execute. Normal users must use *mnt/umnt* when dealing with mountable media.

The options available are:

–*c*
Mount (unmount) all file systems in the set *c*. Acceptable values for *c* are **A, a, B, b, 1, 2,** or **3**. Set membership is specified in the "perms" field of *permissions* file entries.

–d *dev [dir]*
Mount (unmount) *dev* as *dir* if given, otherwise as *mnt_pt* given in permissions file.

–f *[dir]*
Mount (unmount) the device **floppy** as *dir* if given, otherwise as *mnt_pt* given in permissions file.

—l*list_of_devs*
Mount (unmount) all names listed as the corresponding *mnt_pts* given in permissions file.

–r
Mount the device read-only.

FILES

/etc/mnttab	mount table
/etc/mount	
/etc/umount	
/etc/filesys	permissions file

SEE ALSO

mount(1M), filesys(4), mnttab(4).

DIAGNOSTICS

mnt issues a warning if the file system to be mounted is currently mounted under another name.

umnt complains if the special file is not mounted or if it is busy. The file system is busy if it contains an open file or a user's working directory.

BUGS

Some degree of validation is done on the file system; however, it is generally unwise to mount garbage file systems.

NAME

more, page – file perusal filter for crt viewing

SYNOPSIS

more [**–cdflsu**] [*–n*] [*+linenumber*] [*+/pattern*] [name ...]

page *more options*

DESCRIPTION

More is a filter which allows examination of a continuous text one screenful at a time on a soft-copy terminal. It normally pauses after each screenful, printing --More-- at the bottom of the screen. If the user then types a carriage return, one more line is displayed. If the user hits a space, another screenful is displayed. Other possibilities are enumerated later.

The command line options are:

–n An integer which is the size (in lines) of the window which *more* will use instead of the default.

–c *More* will draw each page by beginning at the top of the screen and erasing each line just before it draws on it. This avoids scrolling the screen, making it easier to read while *more* is writing. This option will be ignored if the terminal does not have the ability to clear to the end of a line.

–d *More* will prompt the user with the message "Press space to continue, 'q' to quit." at the end of each screenful, and will respond to subsequent illegal user input by printing "Press 'h' for instructions." instead of ringing the bell. This is useful if *more* is being used as a filter in some setting, such as a class, where many users may be unsophisticated.

–f This causes *more* to count logical, rather than screen lines. That is, long lines are not folded. This option is recommended if *nroff* output is being piped through *ul*, since the latter may generate escape sequences. These escape sequences contain characters which would ordinarily occupy screen positions, but which do not print when they are sent to the terminal as part of an escape sequence. Thus *more* may think that lines are longer than they actually are, and fold lines erroneously.

–l Do not treat ^L (form feed) specially. If this option is not given, *more* will pause after any line that contains a ^L, as if the end of a screenful had been reached. Also, if a file begins with a form feed, the screen will be cleared before the file is printed.

–s Squeeze multiple blank lines from the output, producing only one blank line. Especially helpful when viewing *nroff* output, this option maximizes the useful information present on the screen.

–u Normally, *more* will handle underlining such as produced by *nroff* in a manner appropriate to the particular terminal: if the terminal can perform underlining or has a stand-out mode, *more* will output appropriate escape sequences to enable underlining or stand-out mode for underlined information in the source file. The *–u* option suppresses this processing.

+linenumber
Start up at *linenumber*.

+/pattern
Start up two lines before the line containing the regular expression *pattern*.

If the program is invoked as *page*, then the screen is cleared before each screenful is printed (but only if a full screenful is being printed), and $k-1$ rather than $k-2$ lines are printed in each screenful, where k is the number of lines the terminal can display.

More looks in the file */etc/termcap* to determine terminal characteristics, and to determine the default window size. On a terminal capable of displaying 24 lines, the default window size is 22 lines.

More looks in the environment variable *MORE* to pre-set any flags desired. For example, if you prefer to view files using the *-c* mode of operation, the *csh* command *setenv MORE -c* or the *sh* command sequence *MORE='-c' ; export MORE* would cause all invocations of *more* , including invocations by programs such as *man* and *msgs* , to use this mode. Normally, the user will place the command sequence which sets up the *MORE* environment variable in the *.cshrc* or *.profile* file.

If *more* is reading from a file, rather than a pipe, then a percentage is displayed along with the --More-- prompt. This gives the fraction of the file (in characters, not lines) that has been read so far.

Other sequences which may be typed when *more* pauses, and their effects, are as follows (*i* is an optional integer argument, defaulting to 1) :

i<space>
: display *i* more lines, (or another screenful if no argument is given)

^D
: display 11 more lines (a "scroll"). If *i* is given, then the scroll size is set to *i*.

d
: same as ^D (control-D)

*i*z
: same as typing a space except that *i*, if present, becomes the new window size.

*i*s
: skip *i* lines and print a screenful of lines

*i*f
: skip *i* screenfuls and print a screenful of lines

*i*b
: skip back *i* screenfuls and print a screenful of lines

i^B
: same as b

q or Q
: Exit from *more*.

=
: Display the current line number.

v
: Start up the editor *vi* at the current line.

h
: Help command; give a description of all the *more* commands.

i/*expr*
: search for the *i*-th occurrence of the regular expression *expr*. If there are less than *i* occurrences of *expr*, and the input is a file (rather than a pipe), then the position in the file remains unchanged. Otherwise, a screenful is displayed, starting two lines before the place where the expression was found. The user's erase and kill characters may be used to edit the regular expression. Erasing back past the first column cancels the search command.

*i*n
: search for the *i*-th occurrence of the last regular expression entered.

'
: (single quote) Go to the point from which the last search started. If no search has been performed in the current file, this command goes back to the beginning of the file.

!*command*
: invoke a shell with *command*. The characters '%' and '!' in "command" are replaced with the current file name and the previous shell command respectively. If there is no current file name, '%' is not expanded. The sequences "\%" and "\!" are replaced by "%" and "!" respectively.

i:n
: skip to the *i*-th next file given in the command line (skips to last file if n doesn't make sense)

i:p skip to the *i*-th previous file given in the command line. If this command is given in the middle of printing out a file, then *more* goes back to the beginning of the file. If *i* doesn't make sense, *more* skips back to the first file. If *more* is not reading from a file, the bell is rung and nothing else happens.

:f display the current file name and line number.

:q or :Q
exit from *more* (same as q or Q).

. (dot) repeat the previous command.

The commands take effect immediately, i.e., it is not necessary to type a carriage return. Up to the time when the command character itself is given, the user may hit the line kill character to cancel the numerical argument being formed. In addition, the user may hit the erase character to redisplay the –More--(xx%) message.

At any time when output is being sent to the terminal, the user can hit the quit key (normally control-\). *More* will stop sending output, and will display the usual --More– prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

The terminal is set to *noecho* mode by this program so that the output can be continuous. What you type will thus not show on your terminal, except for the / and ! commands.

If the standard output is not a teletype, then *more* acts just like *cat*, except that a header is printed before each file (if there is more than one).

A sample usage of *more* in previewing *nroff* output would be

```
nroff –ms +2 doc.n | more -s
```

FILES

/etc/termcap	Terminal data base
/usr/lib/more.help	Help file

SEE ALSO

man(1), pg(1), script(1B), sh(1), environ(7).

BUGS

Skipping backwards is too slow on large files.

NAME

mrd – remove a MSDOS sub directory

SYNOPSIS

mrd msdosdirectory

DESCRIPTION

Mrd removes a directory from a MSDOS formatted diskette. Sub directories are supported with either the '/' or '\' separator. The use of the '\' separator will require the directory name to be enclosed in quotes to protect it from the shell.

An error occurs if the directory is not empty.

Wildcards are not supported.

The environment variable MCWD may be used to establish a current working directory (relative to the MSDOS diskette).

The environment variable MFLOPPY must be set to the name of the floppy drive you wish to access. For example, to access floppy drive /dev/rdsk/m320_0hd2, type: MFLOPPY=/dev/rdsk/m320_0hd2;export MFLOPPY

SEE ALSO

mdir(1), mmd(1)

NAME

mread – read (copy) a MSDOS file to Unix

SYNOPSIS

mread [**-tn**] msdosfile unixfile

mread [**-tn**] msdosfile [msdosfiles...] unixdirectory

DESCRIPTION

In the first form, *mread* copies the specified MSDOS file to the named Unix file. The second form of the command copies multiple MSDOS files to the named Unix directory. MSDOS sub directories are supported with either the '/' or '\' separator. The use of the '\' separator or wildcards will require the names to be enclosed in quotes to protect them from the shell.

Mread will allow the following command line options:

t Text file transfer. *Mread* will translate incoming carriage return/line feeds to line feeds.

n No warning. *Mread* will not warn the user when overwriting an existing file.

If the target file already exists, and the *-n* option is not in effect, *mread* asks whether or not to overwrite the file.

The environment variable MCWD may be used to establish a current working directory (relative to the MSDOS diskette).

The environment variable MFLOPPY must be set to the name of the floppy drive you wish to access. For example, to access floppy drive /dev/rdsk/m320_0hd2, type: MFLOPPY=/dev/rdsk/m320_0hd2;export MFLOPPY

SEE ALSO

mdir(1), mtype(1), mwrite(1)

CAVEATS

Wildcards are only applied to filenames, not to directory names. Unlike MSDOS, the destination directory may not be omitted. This utility ignores all EOF markers.

NAME

mren – rename an existing MSDOS file

SYNOPSIS

mren [**-v**] sourcefile targetfile

DESCRIPTION

Mren renames an existing file on a MSDOS formatted diskette. Sub directories are supported with either the '/' or '\' separator. The use of the '\' separator or wildcards will require the names to be enclosed in quotes to protect them from the shell.

Mren will allow the following command line option:

v Verbose mode. Display the new file name if the name supplied is invalid.

Reasonable care is taken to create a valid MSDOS filename. If an invalid MSDOS target name is specified, *mren* will change the name (and prompt the user to accept or reject the new name if the verbose mode is set).

The path component of the target filename (if supplied) is ignored. In other words, you may not use *mren* to move a file from one sub directory to another.

Mren may also be used to rename directories.

The environment variable MCWD may be used to establish a current working directory (relative to the MSDOS diskette).

The environment variable MFLOPPY must be set to the name of the floppy drive you wish to access. For example, to access floppy drive /dev/rdsk/m320_0hd2, type: MFLOPPY=/dev/rdsk/m320_0hd2;export MFLOPPY

SEE ALSO

mwrite(1)

CAVEATS

Wildcards are only applied to filenames, not to directory names. MSDOS doesn't use ren to rename directories.

NAME

mt – magnetic tape control

SYNOPSIS

mt [**–f** *tapename*] *option* [*count*]

DESCRIPTION

mt sends commands to a magnetic tape drive. If *tapename* is not specified, the environment variable TAPE is used; if TAPE does not exist, *mt* uses the device **/dev/rmt/ctapen**. Note that *tapename* must refer to a raw (not block) tape device.

By default, *mt* performs the requested operation once. Operations may be performed repeatedly by specifying a *count* argument. Some operations may not be supported on QIC media.

mt returns a **0** exit status when the operation(s) were successful, **1** if the command was unrecognized, and **2** if an operation failed.

The options to *mt* are written without dashes (–). The available options are listed below. Only specify as many characters that are required to uniquely identify a command.

eof, weof
write *count* EOF marks at the current position on the tape.

fsf
forward space *count* files.

bsf
backward space *count* files.

fsr
forward space *count* records.

bsr
backward space *count* records.

For the following options, *count* is ignored:

rewind
rewind the tape.

retension
wind the tape to the end of the reel and then rewind it, smoothing out the tape tension.

erase
erase the entire tape.

eod
space to the end of the data. This positions the tape at the end of the data so that additional files may be appended.

offline, rewoffl
rewind the tape and put the device offline.

status
provide status information on the tape drive for this system. The *residual=* field indicates XXXXX; the *ds=* field indicates *drive status*; and the *er=* field indicates contents of the error register. Drive status and error register values are device dependent, but provide information that may be useful when reporting problems with the tape interface.

MT(1)

FILES

/dev/rmt/* raw magnetic tape interface

WARNINGS

In previous versions of the **mt** command, the functionality of the *rewind* and *rewoffl* commands was the same, and, as a result, *mt* would accept *rew* as a valid command. In the newest version of this utility, however, the functionality is different for *rewind* and *rewoffl*, and, as a result, user scripts that call *mt rew* will fail. These scripts should be updated to spell out *rewind* or *rewoffl*, whichever is appropriate.

SEE ALSO

environ(5)

NAME

mtype - display contents of a MSDOS file

SYNOPSIS

mtype [**-ts**] msdosfile [msdosfiles...]

DESCRIPTION

Mtype displays the specified MSDOS file on the screen. MSDOS sub directories are supported with either the '/' or '\' separator. The use of the '\' separator or wildcards will require the names to be enclosed in quotes to protect them from the shell.

Mtype will allow the following command line options:

t Text file viewing. *Mtype* will translate incoming carriage return/line feeds to line feeds.

s Strip high bit. *Mtype* will strip the high bit from the data.

The environment variable MCWD may be used to establish a current working directory (relative to the MSDOS diskette).

The environment variable MFLOPPY must be set to the name of the floppy drive you wish to access. For example, to access floppy drive /dev/rdsk/m320_0hd2, type: MFLOPPY=/dev/rdsk/m320_0hd2;export MFLOPPY

SEE ALSO

mdir(1), mread(1)

CAVEATS

Allows multiple arguments, which does not follow the MSDOS convention. Wildcards are only applied to filenames, not to directory names.

NAME

mv – move or rename files

SYNOPSIS

mv [**–i**] [**–f**] [–] file1 file2

mv [**–i**] [**–f**] [–] file ... directory

mv [**–i**] [**–f**] dir1 dir2

DESCRIPTION

Mv moves (changes the name of) *file1* to *file2*.

If *file2* already exists, it is removed before *file1* is moved. If *file2* has a mode which forbids writing, *mv* prints the mode (see *chmod*(2)) and reads the standard input to obtain a line; if the line begins with **y**, the move takes place; if not, *mv* exits.

In the second form, one or more *regular files* are moved to the *directory* with their original filenames.

Mv refuses to move a file onto itself.

Options:

–i stands for interactive mode. Whenever a move is to supersede an existing file, the user is prompted by the name of the file followed by a question mark. If he answers with a line starting with 'y', the move continues. Any other reply prevents the move from occurring.

–f stands for force. This option overrides any mode restrictions or the –i switch.

– means interpret all the following arguments to *mv* as file names. This allows file names starting with minus.

SEE ALSO

cp(1B), ln(1), mv(1).

BUGS

If *file1* and *file2* lie on different file systems, *mv* must copy the file and delete the original. In this case the owner name becomes that of the copying process and any linking relationship with other files is lost.

NAME

mwrite – write (copy) a Unix file to MSDOS

SYNOPSIS

mwrite [**-tnv**] unixfile msdosfile

mwrite [**-tnv**] unixfile [unixfiles...] msdosdirectory

DESCRIPTION

In the first form, *mwrite* copies the specified Unix file to the named MSDOS file. The second form of the command copies multiple Unix files to the named MSDOS directory. MSDOS sub directories are are supported with either the '/' or '\' separator. The use of the '\' separator will require the names to be enclosed in quotes to protect them from the shell.

Mwrite will allow the following command line options:

t Text file transfer. *Mwrite* will translate incoming line feeds to carriage return/line feeds.

n No warning. *Mwrite* will not warn the user when overwriting an existing file.

v Verbose mode. Display the new file name if the Unix file name requires conversion.

If the target file already exists, and the *-n* option is not in effect, *mwrite* asks whether or not to overwrite the file.

Reasonable care is taken to create a valid MSDOS filename. If an invalid name is specified, *mwrite* will change the name (and display the new name if the verbose mode is set).

The environment variable MCWD may be used to establish a current working directory (relative to the MSDOS diskette).

The environment variable MFLOPPY must be set to the name of the floppy drive you wish to access. For example, to access floppy drive /dev/rdsk/m320_0hd2, type: MFLOPPY=/dev/rdsk/m320_0hd2;export MFLOPPY

SEE ALSO

mdir(1), mread(1)

CAVEATS

Wildcards are only applied to filenames, not to directory names. Unlike MSDOS, the destination directory may not be omitted. This command adds an EOF marker to every file (not just text files).

NAME

newform – change the format of a text file

SYNOPSIS

newform [**–s**] [**–i***tabspec*] [**–o***tabspec*] [**–b***n*] [**–e***n*] [**–p***n*] [**–a***n*] [**–f**] [**–c***char*] [**–l***n*] [files]

DESCRIPTION

newform reads lines from the named *files*, or the standard input if no input file is named, and reproduces the lines on the standard output. Lines are reformatted in accordance with command line options in effect.

Except for **–s**, command line options may appear in any order, may be repeated, and may be intermingled with the optional *files*. Command line options are processed in the order specified. This means that option sequences like "–e15 –l60" will yield results different from "–l60 –e15". Options are applied to all *files* on the command line.

–s

Shears off leading characters on each line up to the first tab and places up to 8 of the sheared characters at the end of the line. If more than 8 characters (not counting the first tab) are sheared, the eighth character is replaced by a * and any characters to the right of it are discarded. The first tab is always discarded.

An error message and program exit will occur if this option is used on a file without a tab on each line. The characters sheared off are saved internally until all other options specified are applied to that line. The characters are then added at the end of the processed line.

For example, to convert a file with leading digits, one or more tabs, and text on each line, to a file beginning with the text, all tabs after the first expanded to spaces, padded with spaces out to column 72 (or truncated to column 72), and the leading digits placed starting at column 73, the command would be:

newform –s –i –l –a –e file-name

–i*tabspec*

Input tab specification: expands tabs to spaces, according to the tab specifications given. *Tabspec* recognizes all tab specification forms described in *tabs*(1). In addition, *tabspec* may be —, in which *newform* assumes that the tab specification is to be found in the first line read from the standard input (see *fspec*(4)). If no *tabspec* is given, *tabspec* defaults to **–8**. A *tabspec* of **–0** expects no tabs; if any are found, they are treated as **–1**.

–o*tabspec*

Output tab specification: replaces spaces by tabs, according to the tab specifications given. The tab specifications are the same as for **–i***tabspec*. If no *tabspec* is given, *tabspec* defaults to **–8**. A *tabspec* of **–0** means that no spaces will be converted to tabs on output.

–b*n*

Truncate *n* characters from the beginning of the line when the line length is greater than the effective line length (see **–l***n*). Default is to truncate the number of characters necessary to obtain the effective line length. The default value is used when **–b** with no *n* is used. This option can be used to delete the sequence numbers from a COBOL program as follows:

newform –l1 –b7 file-name

–e*n*

Same as **–b***n* except that characters are truncated from the end of the line.

–p*n*

Prefix *n* characters (see **–c***k*) to the beginning of a line when the line length is less than the effective line length. Default is to prefix the number of characters necessary to obtain the effective line length.

–a*n*

Same as **–p***n* except characters are appended to the end of a line.

–f

Write the tab specification format line on the standard output before any other lines are output. The tab specification format line which is printed will correspond to the format specified in the *last* **–o** option. If no **–o** option is specified, the line which is printed will contain the default specification of **–8**.

–c*k*

Change the prefix/append character to *k*. Default character for *k* is a space.

–l*n*

Set the effective line length to *n* characters. If *n* is not entered, **–l** defaults to 72. The default line length without the **–l** option is 80 characters. Note that tabs and backspaces are considered to be one character (use **–i** to expand tabs to spaces).

The **–l1** must be used to set the effective line length shorter than any existing line in the file so that the **–b** option is activated.

DIAGNOSTICS

All diagnostics are fatal.

usage: ...

newform was called with a bad option.

not –s format

There was no tab on one line.

can't open file

Self-explanatory.

internal line too long

A line exceeds 512 characters after being expanded in the internal work buffer.

tabspec in error

A tab specification is incorrectly formatted, or specified tab stops are not ascending.

tabspec indirection illegal

A *tabspec* read from a file (or standard input) may not contain a *tabspec* referencing another file (or standard input).

0

normal execution

1

for any error

SEE ALSO

csplit(1), tabs(1), fspec(4).

BUGS

newform normally only keeps track of physical characters; however, for the **–i** and **–o** options, *newform* will keep track of backspaces in order to line up tabs in the appropriate logical columns.

newform will not prompt the user if a *tabspec* is to be read from the standard input (by use of **–i—** or **–o—**).

If the **–f** option is used, and the last **–o** option specified was **–o—**, and was preceded by either a **–o—** or a **–i—**, the tab specification format line will be incorrect.

NAME

news – print news items

SYNOPSIS

news [**–a**] [**–n**] [**–s**] [*items*]

DESCRIPTION

news is used to keep the user informed of current events. By convention, these events are described by files in the directory **/usr/news**.

When invoked without arguments, *news* prints the contents of all current files in **/usr/news**, most recent first, with each preceded by an appropriate header. *news* stores the "currency" time as the modification date of a file named **.news_time** in the user's home directory (the identity of this directory is determined by the environment variable $HOME); only files more recent than this currency time are considered "current."

–a
option causes *news* to print all items, regardless of currency. In this case, the stored time is not changed.

–n
option causes *news* to report the names of the current items without printing their contents, and without changing the stored time.

–s
option causes *news* to report how many current items exist, without printing their names or contents, and without changing the stored time. It is useful to include such an invocation of *news* in your **.profile** file, or in the system's **/etc/profile**.

All other arguments are assumed to be specific news items that are to be printed.

If a *delete* is typed during the printing of a news item, printing stops and the next item is started. Another *delete* within one second of the first causes the program to terminate.

FILES

/etc/profile
/usr/news/*
$HOME/.news_time

SEE ALSO

profile(4), environ(5).

NAME

nice – run a command at low priority

SYNOPSIS

nice [–increment] command [arguments]

DESCRIPTION

nice executes *command* with a lower CPU scheduling priority. If the *increment* argument (in the range 1-19) is given, it is used; if not, an increment of 10 is assumed.

The superuser may run commands with priority higher than normal by using a negative increment, e.g., – –10.

Note that *realtime* processes are unaffected by *nice* values (since realtime scheduling priorities are not recomputed by *realix),* and that *timeshare* processes cannot be promoted to realtime priorites by having a favorable nice value.

SEE ALSO

nohup(1), nice(2).

DIAGNOSTICS

nice returns the exit status of the subject command.

BUGS

An *increment* larger than 19 is equivalent to 19.

NAME

nl - line numbering filter

SYNOPSIS

nl [**-b***type*] [**-h***type*] [**-f***type*] [**-v**start#] [**-i***incr*] [**-p**] [**-l***num*] [**-s***sep*] [**-w***width*] [**-n***format*] [**-d***delim*] *file*

DESCRIPTION

nl reads lines from the named *file* or the standard input if no *file* is named and reproduces the lines on the standard output. Lines are numbered on the left in accordance with the command options in effect. *nl* views the text it reads in terms of logical pages. Line numbering is reset at the start of each logical page. A logical page consists of a header, a body, and a footer section. Empty sections are valid. Different line numbering options are independently available for header, body, and footer (e.g., no numbering of header and footer lines while numbering blank lines only in the body). The start of logical page sections are signaled by input lines containing nothing but the following delimiter character(s):

Line contents	*Start of*
\:\:\:	header
\:\:	body
\:	footer

Unless optioned otherwise, *nl* assumes the text being read is in a single logical page body. Command options may appear in any order and may be intermingled with an optional file name. Only one file may be named. The options are:

-b*type*
Specifies which logical page body lines are to be numbered. Recognized *types* and their meaning are:

a	number all lines
t	number lines with printable text only
n	no line numbering
p*string*	number only lines that contain the regular expression specified in *string*.

Default *type* for logical page body is **t** (text lines numbered).

-h*type*
Same as **-b***type* except for header. Default *type* for logical page header is **n** (no lines numbered).

-f*type*
Same as **-b***type* except for footer. Default for logical page footer is **n** (no lines numbered).

-v*start#*
start# is the initial value used to number logical page lines. Default is **1**.

-i*incr*
incr is the increment value used to number logical page lines. Default is **1**.

-p
Do not restart numbering at logical page delimiters.

-l*num*
num is the number of blank lines to be considered as one. For example, **-l2** results in only the second adjacent blank being numbered (if the appropriate **-ha**, **-ba**, and/or **-fa** option is set). Default is **1**.

–s*sep*
sep is the character(s) used in separating the line number and the corresponding text line. Default *sep* is a tab.

–w*width*
width is the number of characters to be used for the line number. Default *width* is **6**.

–n*format*
format is the line numbering format. Recognized values are: **ln**, left justified, leading zeroes suppressed; **rn**, right justified, leading zeroes supressed; **rz**, right justified, leading zeroes kept. Default *format* is **rn** (right justified).

–d*xx*
The delimiter characters specifying the start of a logical page section may be changed from the default characters (\:) to two user-specified characters. If only one character is entered, the second character remains the default character (:). No space should appear between the **–d** and the delimiter characters. To enter a backslash, use two backslashes.

EXAMPLE

The command:

```
nl –v10 –i10 –d!+ file1
```

will number *file1* starting at line number 10 with an increment of 10. The logical page delimiters are !+.

SEE ALSO

pr(1)

NAME

nm – print name list of common object file

SYNOPSIS

nm [**–oxhvnefurpVT**] filename ...

DESCRIPTION

The *nm* command displays the symbol table of each common object file, *file*name. *file name* may be a relocatable or absolute common object file; or it may be an archive of relocatable or absolute common object files. For each symbol, the following information will be printed:

Name
name of the symbol.

Value
value expressed as an offset or an address depending on its storage class.

Class
storage class.

Type
type and derived type. If the symbol is an instance of a structure or of a union then the structure or union tag will be given following the type (e.g., struct-tag). If the symbol is an array, then the array dimensions will be given following the type (e.g., char[n][m]). Note that the object file must have been compiled with the **–g** option of the *cc*(1) command for this information to appear.

Size
size in bytes, if available. Note that the object file must have been compiled with the **–g** option of the *cc*(1) command for this information to appear.

Line
source line number at which it is defined, if available. Note that the object file must have been compiled with the **–g** option of the *cc*(1) command for this information to appear.

Section
For storage classes static and external, the object file section containing the symbol (e.g., text, data or bss).

The output of *nm* may be controlled using the following options:

–o
Print the value and size of a symbol in octal instead of decimal.

–x
Print the value and size of a symbol in hexadecimal instead of decimal.

–h
Do not display the output header data.

–v
Sort external symbols by value before they are printed.

–n
Sort external symbols by name before they are printed.

–e
Print only external and static symbols.

–f
Produce full output. Print redundant symbols (.text, .data, .lib, and .bss), normally suppressed.

-u
Print undefined symbols only.

-r
Prepend the name of the object file or archive to each output line.

-p
Produce easily parsable, terse output. Each symbol name is preceded by its value (blanks if undefined) and one of the letters **U** (undefined), **A** (absolute), **T** (text segment symbol), **D** (data segment symbol), **S** (user defined segment symbol), **R** (register symbol), **F** (file symbol), or **C** (common symbol). If the symbol is local (non-external), the type letter is in lowercase.

-V
Print the version of the nm command executing on the standard error output.

-T
By default, *nm* prints the entire name of the symbols listed. Since object files can have symbols names with an arbitrary number of characters, a name that is longer than the width of the column set aside for names will overflow its column, forcing every column after the name to be misaligned. The **-T** option causes *nm* to truncate every name which would otherwise overflow its column and place an asterisk as the last character in the displayed name to mark it as truncated.

Options may be used in any order, either singly or in combination, and may appear anywhere in the command line. Therefore, both **nm name -e -v** and **nm -ve name** print the static and external symbols in *name*, with external symbols sorted by value.

FILES

TMPDIR/* temporary files

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

BUGS

When all the symbols are printed, they must be printed in the order they appear in the symbol table in order to preserve scoping information. Therefore, the **-v** and **-n** options should be used only in conjunction with the **-e** option.

SEE ALSO

as(1), cc(1), ld(1), tmpnam(3S), a.out(4), ar(4)

DIAGNOSTICS

"nm: name: cannot open"
if *name* cannot be read.

"nm: name: bad magic"
if *name* is not a common object file.

"nm: name: no symbols"
if the symbols have been stripped from *name*.

NAME

nohup – run a command immune to hangups and quits

SYNOPSIS

nohup *command* [*arguments*]

DESCRIPTION

nohup executes *command* with hangups and quits ignored. If output is not re-directed by the user, both standard output and standard error are sent to **nohup.out**. If **nohup.out** is not writable in the current directory, output is redirected to **$HOME/nohup.out**.

EXAMPLE

It is frequently desirable to apply *nohup* to pipelines or lists of commands. This can be done only by placing pipelines and command lists in a single file, called a shell procedure. You can then issue:

```
nohup sh file
```

and the *nohup* applies to everything in *file*. If the shell procedure *file* is to be executed often, then the need to type *sh* can be eliminated by giving *file* execute permission. Add an ampersand and the contents of *file* are run in the background with interrupts also ignored (see *sh*(1)):

```
nohup file &
```

An example of what the contents of *file* could be is:

```
sort ofile > nfile
```

SEE ALSO

chmod(1), nice(1), sh(1), signal(2).

WARNINGS

In the case of the following command

```
nohup command1; command2
```

nohup applies only to command1. The command

```
nohup (command1; command2)
```

is syntactically incorrect.

NAME

oawk – pattern scanning and processing language

SYNOPSIS

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

DESCRIPTION

oawk scans each input *file* for lines that match any of a set of patterns specified in *prog*. With each pattern in *prog*, there can be an associated action that will be performed when a line of a *file* matches the pattern. The set of patterns may appear literally as *prog*, or in a file specified as **–f** *file*. The *prog* string should be enclosed in single quotes (') to protect it from the shell.

parameters, in the form x=... y=... etc., may be passed to *oawk*.

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

An input line is made up of fields separated by white space. (This default can be changed by using FS; see below). The fields are denoted **$1**, **$2**, ..., and **$0** refers to the entire line.

A pattern-action statement has the form:

pattern { action }

A missing *action* means print the line; a missing *pattern* always matches. An *action* is a sequence of statements. A statement can be one of the following:

```
if ( conditional ) statement [ else statement ]
while ( conditional ) statement
for ( expression ; conditional ; expression ) statement
break
continue
{ [ statement ] ... }
variable = expression
print [ expression-list ] [ >expression ]
printf format [ , expression-list ] [ >expression ]
next    # skip remaining patterns on this input line
exit    # skip the rest of the input
```

Statements are terminated by semicolons, newlines, 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 *>filename* is present), separated by the current output field separator, and terminated by the output record separator. The *printf* statement formats its *expression-list* as described in *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, as in *egrep* (see *grep*(1)). Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions. A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines between an occurrence of the first pattern and the next occurrence of the second.

A relational expression is one of the following:

expression matchop regular-expression
expression relop expression

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

The special patterns BEGIN and END may be used to capture control before the first input line is read and after the last. BEGIN must be the first pattern, END the last.

A single character *c* may be used to separate the fields by starting the program with:

BEGIN { FS = *c* }

or by using the **–F***c* option.

Other variable names with special meanings include NF, the number of fields in the current record; NR, the ordinal number of the current record; FILENAME, the name of the current input file; OFS, the output field separator (default blank); ORS, the output record separator (default new-line); and OFMT, the output format for numbers (default **%.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: oawk –f program n=5 input

SEE ALSO

awk(1), grep(1), lex(1), sed(1), printf(3S).

BUGS

Input white space is not preserved on output if fields are involved.

There are no explicit conversions between numbers and strings. To force an expression to be treated as a number add 0 to it; to force it to be treated as a string concatenate the **null** string ("") to it.

NAME

od – octal dump

SYNOPSIS

od [**–bcdosx**] [*file*] [[**+**]*offset*[**.**][**b**]]

DESCRIPTION

od dumps *file* in one or more formats as selected by the first argument. If the first argument is missing, **–o** is default. The meanings of the format options are:

–b
Interpret bytes in octal.

–c
Interpret bytes in ASCII. Certain non-graphic characters appear as C escapes: null=**\0**, backspace=**\b**, form-feed=**\f**, newline=**\n**, return=**\r**, tab=**\t**; others appear as 3-digit octal numbers.

–d
Interpret words in unsigned decimal.

–o
Interpret words in octal.

–s
Interpret 16-bit words in signed decimal.

–x
Interpret words in hex.

The *file* argument specifies which file is to be dumped. If no file argument is specified, the standard input is used.

The offset argument specifies the offset in the file where dumping is to commence. This argument is normally interpreted as octal bytes. If **.** is appended, the offset is interpreted in decimal. If **b** is appended, the offset is interpreted in blocks of 512 bytes. If the file argument is omitted, the offset argument must be preceded by **+**.

Dumping continues until end-of-file.

NAME

pack, pcat, unpack – compress and expand files

SYNOPSIS

pack [–] [–**f**] *name* ...

pcat *name* ...

unpack *name* ...

DESCRIPTION

pack attempts to store the specified files in a compressed form. Wherever possible (and useful), each input file *name* is replaced by a packed file *name.z* with the same access modes, access and modified dates, and owner as those of *name*. The **-f** option will force packing of *name*. This is useful for causing an entire directory to be packed even if some of the files will not benefit. If ***pack*** is successful, *name* will be removed. Packed files can be restored to their original form using ***unpack*** or ***pcat***.

pack uses Huffman (minimum redundancy) codes on a byte-by-byte basis. If the – argument is used, an internal flag is set that causes the size (in bytes) of the unpacked and packed files, the number of levels in the Huffman tree, the number of distinct bytes in the input file, the dictionary overhead in bytes, and the effective and asymptotic entropy values to be printed on the standard output. Additional occurrences of – in place of *name* will cause the internal flag to be set and reset.

The amount of compression obtained depends on the size of the input file and the character frequency distribution. Because a decoding tree forms the first part of each **.z** file, it is usually not worthwhile to pack files smaller than three blocks, unless the character frequency distribution is very skewed, which may occur with printer plots or pictures.

Typically, text files are reduced to 60-75% of their original size. Load modules, which use a larger character set and have a more uniform distribution of characters, show little compression, the packed versions being about 90% of the original size.

pack returns a value that is the number of files that it failed to compress.

No packing will occur if:

the file appears to be already packed;
the file name has more than 12 characters;
the file has links;
the file is a directory;
the file cannot be opened;
no disk storage blocks will be saved by packing;
a file called *name.z* already exists;
the **.z** file cannot be created;
an I/O error occurred during processing.

The last segment of the file name must contain no more than 12 characters to allow space for the appended **.z** extension. Directories cannot be compressed.

pcat does for packed files what *cat*(1) does for ordinary files, except that *pcat* cannot be used as a filter. The specified files are unpacked and written to the standard output. Thus, to view a packed file named *name*.**z** use:

pcat name.z

or just:

pcat name

To make an unpacked copy, say *nnn*, of a packed file named *name*.**z** (without destroying *name*.**z**) use the command:

pcat name > nnn

pcat returns the number of files it was unable to unpack. Failure may occur if:

the file name (exclusive of the **.z**) has more than 12 characters;
the file cannot be opened;
the file does not appear to be the output of *pack*.

unpack expands files created by *pack*. For each file *name* specified in the command, a search is made for a file called *name*.**z** (or just *name*, if *name* ends in **.z**). If this file appears to be a packed file, it is replaced by its expanded version. The new file has the **.z** suffix stripped from its name, and has the same access modes, access and modification dates, and owner as those of the packed file.

unpack returns a value that is the number of files it was unable to unpack. Failure may occur for the same reasons that it may in *pcat*, as well as for the following:

a file with the "unpacked" name already exists;
if the unpacked file cannot be created.

SEE ALSO

cat(1)

NAME

passwd – change login password and password attributes

SYNOPSIS

passwd [*name*]

passwd [**–l** | **–d**] [**–n** *min*] [**–f**] [**–x** *max*] *name*

passwd **–s** [**–a**]

passwd **–s** [*name*]

DESCRIPTION

The ***passwd*** command changes the password or lists password attributes associated with the user's login *name*. Additionally, superusers may use ***passwd*** to install or change passwords and attributes associated with any login *name*.

When used to change a password, ***passwd*** prompts ordinary users for their old password, if any. It then prompts for the new password twice. When the old password is entered, ***passwd*** checks to see if it has "aged" sufficiently. If "aging" is insufficient, ***passwd*** terminates; see ***passwd***(4).

Assuming aging is sufficient, a check is made to ensure that the new password meets construction requirements. When the new password is entered a second time, the two copies of the new password are compared. If the two copies are not identical the cycle of prompting for the new password is repeated for at most two more times.

Passwords must be constructed to meet the following requirements:

Each password must have at least six characters. Only the first eight characters are significant.

Each password must contain at least two alphabetic characters and at least one numeric or special character. In this case, "alphabetic" refers to all upper- or lowercase letters.

Each password must differ from the user's login *name* and any reverse or circular shift of that login *name*. For comparison purposes, an uppercase letter and its corresponding lowercase letter are equivalent.

New passwords must differ from the old by at least three characters. For comparison purposes, an uppercase letter and its corresponding lowercase letter are equivalent.

Superusers (e.g., real and effective uid equal to zero, see ***id***(1M) and ***su***(1M)) may change any password; hence, ***passwd*** does not prompt superusers for the old password. Superusers are not forced to comply with password aging and password construction requirements. A superuser can create a **NULL** password by entering a carriage return in response to the prompt for a new password. (This differs from **passwd –d** because the "password" prompt will still be displayed.)

Any user may use the **–s** option to show password attributes for his or her own login *name*.

The format of the display will be:

name status mm/dd/yy min max

or, if password aging information is not present,

name status

where

name
The login ID of the user.

status
The password status of *name*: "PS" stands for passworded or locked, "LK" stands for locked, and "NP" stands for no password.

mm/dd/yy
The date password was last changed for *name*. (Note that all password aging dates are determined using Greenwich Mean Time and, therefore, may differ by as much as a day in other time zones.)

min
The minimum number of days required between password changes for *name*.

max
The maximum number of days the password is valid for *name*.

Only a superuser can use the following options:

-l
Locks password entry for *name*.

-d
Deletes password for *name*. The login *name* will not be prompted for password.

-n
Set minimum field for *name*. The *min* field contains the minimum number of days between password changes for *name*. If *min* is greater than *max*, the user may not change the password. Always use this option with the **-x** option, unless *max* is set to -1 (aging turned off). In that case, *min* need not be set.

-x
Set maximum field for *name*. The *max* field contains the number of days that the password is valid for *name*. The aging for *name* will be turned off immediately if *max* is set to -1. If it is set to 0, then the user is forced to change the password at the next login session and aging is turned off.

-a
Show password attributes for all entries. Use only with **-s** option; *name* must not be provided.

-f
Force the user to change password at the next login by expiring the password for *name*.

FILES

/etc/passwd, /etc/shadow, /etc/opasswd, /etc/oshadow

SEE ALSO

login(1) id(1M), passmgmt(1M), pwconv(1M), su(1M), crypt(3C), passwd(4).

WARNINGS

If the optional **/etc/shadow** file feature is used, the *passwd*(1) command uses that instead of the **/etc/passwd** file to obtain password information. Because the way password aging information is stored in the two files is slightly different, the output from *passwd* options that use this information may also be different.

If Network Information Service (NIS) is being used, the file **/etc/passwd.local** is modified, where necessary, instead of **/etc/passwd**. The contents of **/etc/passwd.local** are then combined with the contents of the *passwd.nan* NIS map to form the **/etc/passwd** file. The network **passwd** file maintained by the master NIS server cannot be modified by this command; use *yppasswd*(1N), described in the *Network Services User's Reference Manual.*

DIAGNOSTICS

The *passwd* command exits with one of the following values:

0	SUCCESS.
1	Permission denied.
2	Invalid combination of options.
3	Unexpected failure. Password file unchanged.
4	Unexpected failure. Password file(s) missing.
5	Password file(s) busy. Try again later.
6	Invalid argument to option.

NAME

paste – merge same lines of several files or subsequent lines of one file

SYNOPSIS

paste *file1 file2 ...*
paste **–d** *list file1 file2 ...*
paste **–s** [**–d** *list*] *file1 file2 ...*

DESCRIPTION

In the first two forms, *paste* concatenates corresponding lines of the given input files *file1, file2*. It treats each file as a column or columns of a table and pastes them together horizontally (parallel merging). It is the counterpart of *cat*(1) which concatenates vertically, i.e., one file after the other. In the last form, *paste* replaces the function of an older command with the same name by combining subsequent lines of the input file (serial merging). In all cases, lines are glued together with the *tab* character, or with characters from an optionally specified *list*. Output is to the standard output, so it can be used as the start of a pipe, or as a filter, if – is used in place of a file name.

The meanings of the options are:

–d

Without this option, the newline characters of each but the last file (or last line in the **–s** option) are replaced by a *tab* character. This option allows replacing the *tab* character by one or more alternate characters (see below).

list

One or more characters immediately following **–d** replace the default *tab* as the line concatenation character. The list is used circularly, i.e., when exhausted, it is reused. In parallel merging (i.e., no **–s** option), the lines from the last file are always terminated with a newline character, not from the *list*. The *list* may contain the special escape sequences: **\n** (newline), **\t** (tab), **** (backslash), and **\0** (empty string, not a **NULL** character). Quoting may be necessary, if characters have special meaning to the shell (e.g., to get one backslash, use *–d "\\\\"*).

–s

Merge subsequent lines instead of one from each input file. Use *tab* for concatenation, unless a *list* is specified with **–d** option. Regardless of the *list*, the very last character of the file is forced to be a newline.

–

May be used in place of any file name, to read a line from the standard input. (There is no prompting).

EXAMPLES

ls | paste –d" " –	list directory in one column
ls | paste – – – –	list directory in four columns
paste –s –d"\t\n" file	combine pairs of lines into lines

SEE ALSO

cut(1), grep(1), pr(1)

DIAGNOSTICS

line too long	Output lines are restricted to 511 characters.
too many files	Except for **–s** option, no more than 12 input files may be specified.

NAME

pg – file perusal filter for CRTs

SYNOPSIS

pg [–*number*] [–p *string*] [–cefns] [+*linenumber*] [+/*pattern*/] [*files* ...]

DESCRIPTION

The *pg* command is a filter that allows the examination of *files* one screenful at a time on a CRT. (The file name and/or **NULL** arguments indicate that *pg* should read from the standard input.) Each screenful is followed by a prompt. If the user types a carriage return, another page is displayed; other possibilities are described later.

This command is different from previous paginators because it allows you to back up and review something that has already passed. The method for doing this is explained below.

To determine terminal attributes, *pg* scans the *terminfo*(4) data base for the terminal type specified by the environment variable TERM . If TERM is not defined, the terminal type **dumb** is assumed.

The command line options are:

–*number*
An integer specifying the size (in lines) of the window that *pg* is to use instead of the default. (On a terminal containing 24 lines, the default window size is 23).

–p *string*
Causes *pg* to use *string* as the prompt. If the prompt string contains a "%d", the first occurrence of "%d" in the prompt will be replaced by the current page number when the prompt is issued. The default prompt string is ":".

–c
Homes the cursor and clears the screen before displaying each page. This option is ignored if **clear_screen** is not defined for this terminal type in the *terminfo*(4) data base.

–e
Causes *pg* **not** to pause at the end of each file.

–f
Normally, *pg* splits lines longer than the screen width, but some sequences of characters in the text being displayed (e.g., escape sequences for underlining) generate undesirable results. The –f option inhibits *pg* from splitting lines.

–n
Causes an automatic end of command as soon as a command letter is entered. (Normally, commands must be terminated by a **newline** character.)

–s
Causes *pg* to print all messages and prompts in standout mode (usually inverse video).

+*linenumber*
Start up at *linenumber*.

+/*pattern*/
Start up at the first line containing the regular expression pattern.

The responses that may be typed when *pg* pauses can be divided into three categories: those causing further perusal, those that search, and those that modify the perusal environment.

Commands that cause further perusal normally take a preceding *address*, an optionally signed number indicating the point from which further text should be displayed. This *address* is interpreted in either pages or lines depending on the command. A signed *address* specifies a point relative to the current page or line, and an unsigned *address* specifies an address relative to the

beginning of the file. Each command has a default address that is used if none is provided.

The perusal commands and their defaults are:

(+1) **newline** or **blank**
This causes one page to be displayed. The address is specified in pages.

(+1) **l**
With a relative address this causes *pg* to simulate scrolling the screen, forward or backward, the number of lines specified. With an absolute address this command prints a screenful beginning at the specified line.

(+1) **d** or **^D**
Simulates scrolling half a screen forward or backward.

The following perusal commands take no *address*.

. or **^L**
Typing a single period causes the current page of text to be redisplayed.

$ Displays the last windowful in the file. Use with caution when the input is a pipe.

The following commands are available to search for text patterns in the text. The regular expressions described in *ed*(1) are available. They must always be terminated by a **newline**, even if the *−n* option is specified.

i/*pattern*/
Search forward for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately after the current page and continues to the end of the current file, without wrap-around.

i^*pattern*^
i?*pattern*?
Search backwards for the *i*th (default *i*=1) occurrence of *pattern*. Searching begins immediately before the current page and continues to the beginning of the current file, without wrap-around. The ^ notation is useful for Adds 100 terminals that will not properly handle the ?.

After searching, *pg* will normally display the line found at the top of the screen. This can be modified by appending **m** or **b** to the search command to leave the line found in the middle or at the bottom of the window from now on. The suffix **t** can be used to restore the original situation.

The user of *pg* can modify the environment of perusal with the following commands:

*i***n**
Begin perusing the *i*th next file in the command line. The *i* is an unsigned number, default value is 1.

*i***p**
Begin perusing the *i*th previous file in the command line. *i* is an unsigned number, default is 1.

*i***w**
Display another window of text. If *i* is present, set the window size to *i*.

s *filename*
Save the input in the named file. Only the current file being perused is saved. The white space between the **s** and *filename* is optional. This command must always be terminated by a **newline** even if the *−n* option is specified.

h
Help by displaying an abbreviated summary of available commands.

q or **Q**
Quit *pg*.

!*command*
Command is passed to the shell, whose name is taken from the SHELL environment variable. If this is not available, the default shell is used. This command must always be terminated by a **newline** even if the *−n* option is specified.

At any time when output is being sent to the terminal, the user can press the quit key (normally **CTRL**) or the interrupt (**BREAK**) key. This causes *pg* to stop sending output, and display the prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

If the standard output is not a terminal, then *pg* acts just like *cat*(1), except that a header is printed before each file (if there is more than one).

EXAMPLE

A sample usage of *pg* in reading system news would be

```
news | pg -p (Page %d):
```

NOTES

While waiting for terminal input, *pg* responds to **BREAK DEL**, and ^ by terminating execution. Between prompts, however, these signals interrupt *pgs* current task and place the user in prompt mode. These should be used with caution when input is being read from a pipe, since an interrupt is likely to terminate the other commands in the pipeline.

The *pg* command responds to ^z (**CTRL**–z) by suspending a job on those systems that provide job control. Upon awakening, the screen is refreshed.

Users of Berkeley's *more* will find that the *z* and *f* commands are available, and that the terminal /, ^, or ? may be omitted from the searching commands.

FILES

/usr/lib/terminfo/?/*	terminal information database
/tmp/pg*	temporary file when input is from a pipe

SEE ALSO

ed(1), grep(1), terminfo(4).

BUGS

If terminal tabs are not set every eight positions, undesirable results may occur.

When using *pg* as a filter with another command that changes the terminal I/O options terminal settings may not be restored correctly.

NAME

pr – print files

SYNOPSIS

pr [[**-column**] [**-w***width*] [**-a**]] [**-e**ck] [**-i**ck] [**-drtfp**] [**+***page*] [**-n***ck*] [**-o***offset*] [**-l***length*] [**-s***separator*] [**-h***header*] [*file ...*]

pr [[**-m**] [**-w***width*]] [**-e**ck] [**-i**ck] [**-drtfp**] [**+***page*] [**-n***ck*] [**-o***offset*] [**-l***length*] [**-s***separator*] [**-h***header*] *file1 file2 ...*

DESCRIPTION

pr is used to format and print the contents of a file. If *file* is –, or if no files are specified, *pr* assumes standard input. *pr* prints the named files on standard output.

By default, the listing is separated into pages, each headed by the page number, the date and time that the file was last modified, and the name of the file. Page length is 66 lines which includes 10 lines of header and trailer output. The header is composed of 2 blank lines, 1 line of text (can be altered with **-h**), and 2 blank lines; the trailer is 5 blank lines.

For single column output, line width may not be set and is unlimited. For multicolumn output, line width may be set and the default is 72 columns. Diagnostic reports (failed options) are reported at the end of standard output associated with a terminal, rather than interspersed in the output. Pages are separated by series of line feeds rather than form feed characters.

By default, columns are of equal width, separated by at least one space; lines which do not fit are truncated. If the **-s** option is used, lines are not truncated and columns are separated by the *separator* character.

Either *-column* or **-m** should be used to produce multi-column output. **-a** should only be used with *-column* and not **-m**.

Command line options are

+*page*
Begin printing with page numbered *page* (default is 1).

-*column*
Print *column* columns of output (default is 1). Output appears as if **-e** and **-i** are turned on for multi-column output. May not use with **-m**.

-a
Print multi-column output across the page one line per column. *columns* must be greater than one. If a line is too long to fit in a column, it is truncated.

-m
Merge and print all files simultaneously, one per column. The maximum number of files that may be specified is eight. If a line is too long to fit in a column, it is truncated. May not use with *-column*.

-d
Doublespace the output. Blank lines that result from doublespacing are dropped when they occur at the top of a page.

-e*ck*
Expand input tabs to character positions k+1, 2*k+1, 3*k+1, etc. If k is 0 or is omitted, default tab settings at every eighth position are assumed. Tab characters in the input are expanded into the appropriate number of spaces. If c (any non-digit character) is given, it is treated as the input tab character (default for c is the tab character).

–i*ck*

In output, replace white space wherever possible by inserting tabs to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. If *c* (any non-digit character) is given, it is treated as the output tab character (default for *c* is the tab character).

–n*ck*

Provide *k*-digit line numbering (default for *k* is 5). The number occupies the first *k*+1 character positions of each column of single column output or each line of **–m** output. If *c* (any non-digit character) is given, it is appended to the line number to separate it from whatever follows (default for *c* is a tab).

–w*width*

Set the width of a line to *width* character positions (default is 72). This is effective only for multi-column output (*-column* and **–m**). There is no line limit for single column output.

–o*offset*

Offset each line by *offset* character positions (default is 0). The number of character positions per line is the sum of the width and offset.

–l*length*

Set the length of a page to *length* lines (default is 66). –l0 is reset to –l66. When the value of *length* is 10 or less, **–t** appears to be in effect since headers and trailers are suppressed. By default, output contains 5 lines of header and 5 lines of trailer leaving 56 lines for user-supplied text. When **–l***length* is used and *length* exceeds 10, then *length*-10 lines are left per page for user supplied text. When *length* is 10 or less, header and trailer output is omitted to make room for user supplied text.

–h *header*

Use *header* as the text line of the header to be printed instead of the file name. **–h** is ignored when **–t** is specified or **–l***length* is specified and the value of *length* is 10 or less. (**–h is the only** *pr* option requiring space between the option and argument.)

–p

Pause before beginning each page if the output is directed to a terminal (*pr* will ring the bell at the terminal and wait for a carriage return).

–f

Use single form-feed character for new pages (default is to use a sequence of linefeeds). Pause before beginning the first page if the standard output is associated with a terminal.

–r

Print no diagnostic reports on files that will not open.

–t

Print neither the five-line identifying header nor the five-line trailer normally supplied for each page. Quit printing after the last line of each file without spacing to the end of the page. Use of **–t** overrides the **–h** option.

–s*separator*

Separate columns by the single character *separator* instead of by the appropriate number of spaces (default for *separator* is a tab). Prevents truncation of lines on multicolumn output unless **–w** is specified.

EXAMPLES

Print *file1* and *file2* as a doublespaced, threecolumn listing headed by "file list":

pr −3dh "file list" file1 file2

Copy *file1* to *file2*, expanding tabs to columns 10, 19, 28, 37, . . . :

pr −e9 −t <file1 >file2

Print *file1* and *file2* simultaneously in a twocolumn listing with no header or trailer where both columns have line numbers:

pr −t −n file1 | pr −t −m −n file2 -

FILES

/dev/tty* If standard output is directed to one of the special files **/dev/tty***, then other output directed to this terminal is delayed until standard output is completed. This prevents error messages from being interspersed throughout the output.

SEE ALSO

cat(1), pg(1)

NAME

prof – display profile data

SYNOPSIS

prof [–tcan] [–ox] [–g] [–z] [–h] [–s] [–m*mdata*] [*prog*]

DESCRIPTION

The *prof* command interprets a profile file produced by the *monitor*(3C) function. The symbol table in the object file *prog* (**a.out** by default) is read and correlated with a profile file (**mon.out** by default). For each external text symbol the percentage of time spent executing between the address of that symbol and the address of the next is printed, together with the number of times that function was called and the average number of milliseconds per call.

The mutually exclusive options **t**, **c**, **a**, and **n** determine the type of sorting of the output lines:

–t
Sort by decreasing percentage of total time (default).

–c
Sort by decreasing number of calls.

–a
Sort by increasing symbol address.

–n
Sort lexically by symbol name.

The mutually exclusive options **o** and **x** specify the printing of the address of each symbol monitored:

–o
Print each symbol address (in octal) along with the symbol name.

–x
Print each symbol address (in hexadecimal) along with the symbol name.

The following options may be used in any combination:

–g
Include non-global symbols (static functions).

–z
Include all symbols in the profile range (see *monitor*(3C)), even if associated with zero number of calls and zero time.

–h
Suppress the heading normally printed on the report. (This is useful if the report is to be processed further.)

–s
Print a summary of several of the monitoring parameters and statistics on the standard error output.

–m mdata
Use file *mdata* instead of **mon.out** as the input profile file.

A program creates a profile file if it has been loaded with the **–p** option of *cc*(1). This option to the *cc* command arranges for calls to *monitor*(3C) at the beginning and end of execution. It is the call to *monitor* at the end of execution that causes a profile file to be written. The number of calls to a function is tallied if the **–p** option was used when the file containing the function was compiled.

The name of the file created by a profiled program is controlled by the environment variable PROFDIR. If PROFDIR does not exist, "mon.out" is produced in the directory that is current when the program terminates. If PROFDIR = string, "string/pid.progname" is produced, where *progname* consists of *argv*[0] with any path prefix removed, and *pid* is the program's process id. If PROFDIR is the **NULL** string, no profiling output is produced.

A single function may be split into subfunctions for profiling by means of the MARK macro (see *prof*(5)).

FILES

mon.out	for profile
a.out	for namelist

SEE ALSO

cc(1), exit(2), profil(2), monitor(3C), prof(5)

WARNING

The times reported in successive identical runs may show variances of 20% or more, because of varying cache-hit ratios due to sharing of the cache with other processes. Even if a program seems to be the only one using the machine, hidden background or asynchronous processes may blur the data. In rare cases, the clock ticks initiating recording of the program counter may "beat" with loops in a program, grossly distorting measurements.

Call counts are always recorded precisely.

The times for static functions are attributed to the preceding external text symbol if the -g option is not used. However, the call counts for the preceding function are still correct, i.e., the static function call counts are not added in with the call counts of the external function.

CAVEATS

Only programs that call *exit*(2) or return from *main* will cause a profile file to be produced, unless a final call to monitor is explicitly coded.

The use of the −p option to *cc*(1) to invoke profiling imposes a limit of 600 functions that may have call counters established during program execution. For more counters you must call *monitor*(3C) directly. If this limit is exceeded, other data will be overwritten and the **mon.out** file will be corrupted. The number of call counters used will be reported automatically by the *prof* command whenever the number exceeds 5/6 of the maximum.

NAME

prs – print an SCCS file

SYNOPSIS

prs [–d[*dataspec*]] [–r[*SID*]] [–e] [–l] [–c[*date-time*]] [–a] *files*

DESCRIPTION

prs prints, on the standard output, parts or all of an SCCS file (see *sccsfile*(4)) in a user-supplied format. If a directory is named, *prs* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the 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 or directory to be processed; non-SCCS files and unreadable files are silently ignored.

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

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

–d*[dataspec]*
Used to specify the output data specification. The *dataspec* is a string consisting of SCCS file *data keywords* (see **DATA KEYWORDS**) interspersed with optional user supplied text.

–r*[SID]*
Used to specify the *SCCS* ID*entification* (SID) string of a delta for which information is desired. If no SID is specified, the SID of the most recently created delta is assumed.

–e
Requests information for all deltas created *earlier* than and including the delta designated via the –r keyletter or the date given by the –c option.

–l
Requests information for all deltas created *later* than and including the delta designated via the –r keyletter or the date given by the –c option.

c date-time The cutoff date-time –c[cutoff] is in the form:

YY[MM[DD[HH[MM[SS]]]]]

–c*[date-time]*
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 in the form: **"–c77/2/2 9:22:25"**.

–a
Requests printing of information for both removed, i.e., delta type = *R*, (see *rmdel*(1)) and existing, i.e., delta type = *D*, deltas. If the –a keyletter is not specified, information for existing deltas only is provided.

DATA KEYWORDS

Data keywords specify which parts of an SCCS file are to be retrieved and output. All parts of an SCCS file (see *sccsfile*(4)) have an associated data keyword. There is no limit on the number of times a data keyword may appear in a *dataspec*.

The information printed by *prs* consists of: (1) the user-supplied text; and (2) appropriate values (extracted from the SCCS file) substituted for the recognized data keywords in the order of appearance in the *dataspec*. The format of a data keyword value is either *Simple* (S), in which keyword substitution is direct, or *Multi-line* (M), in which keyword substitution is followed by a carriage return.

User-supplied text is any text other than recognized data keywords.

A tab is specified by **\t** and carriage return/newline is specified by **\n**. The default data keywords are:

":Dt:\t:DL:\nMRs:\n:MR:COMMENTS:\n:C:"

Table 1. SCCS Files Data Keywords

Keyword	Data Item	File Section	Value	Format
:Dt:	Delta information	Delta Table	See below*	S
:DL:	Delta line statistics	"	:Li:/:Ld:/:Lu:	S
:Li:	Lines inserted by Delta	"	nnnnn	S
:Ld:	Lines deleted by Delta	"	nnnnn	S
:Lu:	Lines unchanged by Delta	"	nnnnn	S
:DT:	Delta type	"	*D*" or "*R*	S
:I:	SCCS ID string (SID)	"	:R:.:L:.:B:.:S:	S
:R:	Release number	"	nnnn	S
:L:	Level number	"	nnnn	S
:B:	Branch number	"	nnnn	S
:S:	Sequence number	"	nnnn	S
:D:	Date Delta created	"	:Dy:/:Dm:/:Dd:	S
:Dy:	Year Delta created	"	nn	S
:Dm:	Month Delta created	"	nn	S
:Dd:	Day Delta created	"	nn	S
:T:	Time Delta created	"	:Th:::Tm:::Ts:	S
:Th:	Hour Delta created	"	nn	S
:Tm:	Minutes Delta created	"	nn	S
:Ts:	Seconds Delta created		nn	S
:P:	Programmer who created Delta	"	logname	S
:DS:	Delta sequence number	"	nnnn	S
:DP:	Predecessor Delta seq-no.	"	nnnn	S
:DI:	Seq-no. of deltas incl., excl., ignored	"	:Dn:/:Dx:/:Dg:	S
:Dn:	Deltas included (seq #)	"	:DS:" :DS:...	S
:Dx:	Deltas excluded (seq #)	"	:DS:" :DS:...	S
:Dg:	Deltas ignored (seq #)	"	:DS:" :DS:...	S
:MR:	MR numbers for delta	"	text	M
:C:	Comments for delta	"	text	M
:UN:	User names	User Names	text	M
:FL:	Flag list	Flags	text	M
:Y:	Module type flag	"	text	S
:MF:	MR validation flag	"	*yes*" or "*no*	S

Table 1. SCCS Files Data Keywords (cont'd)

Keyword	Data Item	File Section	Value	Format
:MP:	MR validation pgm name	"	text	S
:KF:	Keyword error/warning flag	"	*yes*˜or˜*no*	S
:KV:	Keyword validation string	"	text	S
:BF:	Branch flag	"	*yes*˜or˜*no*	S
:J:	Joint edit flag	"	*yes*˜or˜*no*	S
:LK:	Locked releases	"	:R:...	S
:Q:	User-defined keyword	"	text	S
:M:	Module name	"	text	S
:FB:	Floor boundary	"	:R:	S
:CB:	Ceiling boundary	"	:R:	S
:Ds:	Default SID	"	:I:	S
:ND:	Null delta flag	"	*yes*˜or˜*no*	S
:FD:	File descriptive text	Comments	text	M
:BD:	Body	Body	text	M
:GB:	Gotten body	"	text	M
:W:	A form of *what*(1) string	N/A	:Z::M:\t:I:	S
:A:	A form of *what*(1) string	N/A	:Z::Y:˜:M:˜:I::Z:	S
:Z:	*what*(1) string delimiter	N/A	@(#)	S
:F:	SCCS file name	N/A	text	S
:PN:	SCCS file pathname	N/A	text	S

* :Dt:˜=˜:DT:˜:I:˜:D:˜:T:˜:P:˜:DS:˜:DP:

EXAMPLES

prs –d"Users and/or user IDs for :F: are:\n:UN:" s.file

may produce on the standard output:

```
Users and/or user IDs for s.file are:
xyz
131
abc
```

prs –d"Newest delta for pgm :M:: :I: Created :D: By :P:" –r s.file

may produce on the standard output:

```
Newest delta for pgm main.c: 3.7 Created 77/12/1 By cas
```

As a *special case:*

prs s.file

may produce on the standard output:

```
D 1.1 77/12/1 00:00:00 cas 1 000000/00000/00000
MRs:
bl78-12345
bl79-54321
COMMENTS:
this is the comment line for s.file initial delta
```

for each delta table entry of the "D" type. The only keyletter argument allowed to be used with the *special case* is the –a keyletter.

FILES

/tmp/pr?????

SEE ALSO

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

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

ps – report process status

SYNOPSIS

ps [options]

DESCRIPTION

ps prints certain information about active processes. Without *options*, information is printed about processes associated with the controlling terminal. The output consists of a short listing containing only the process ID, terminal identifier, cumulative execution time, and the command name. Otherwise, the information that is displayed is controlled by the selection of *options*.

Some *options* accept names or lists as arguments. Arguments can be either separated from one another by commas or enclosed in double quotes and separated from one another by commas or spaces. Values for *proclist* and *grplist* must be numeric.

The *options* are given in descending order according to volume and range of information provided:

–e	Print information about every process now running.
–d	Print information about all processes except process group leaders.
–a	Print information about all processes most frequently requested: all those except process group leaders and processes not associated with a terminal.
–f	Generate a full listing. (See below for significance of columns in a full listing.)
–l	Generate a long listing. (See below.)
–c *corefile*	Use the file *corefile* in place of **/dev/mem**.
–n *name*	Take argument signifying an alternate system *name* in place of **/realix**.
–t *termlist*	List only process data associated with the terminal given in *termlist*. Terminal identifiers may be specified in one of two forms: the device's file name (e.g., **tty04**) or, if the device's file name starts with **tty**, just the digit identifier (e.g., **04**).
–p *proclist*	List only process data whose process ID numbers are given in *proclist*.
–u *uidlist*	List only process data whose user ID number or login name is given in *uidlist*. In the listing, the numerical user ID will be printed unless you give the **–f** option, which prints the login name.
–g *grplist*	List only process data whose process group leader's ID number(s) appears in *grplist*. (A group leader is a process whose process ID number is identical to its process group ID number. A login shell is a common example of a process group leader.)

Under the **–f** option, *ps* tries to determine the command name and arguments given when the process was created by examining the user block. Failing this, the command name is printed, as it would have appeared without the **–f** option, in square brackets.

The column headings and the meaning of the columns in a *ps* listing are given below; the letters **f** and **l** indicate the option (full or long, respectively) that causes the corresponding heading to appear; **all** means that the heading always appears. Note that these two options determine only what information is provided for a process; they do not determine which processes will be listed.

F (l) Flags (hexadecimal and additive) associated with the process:

00	Process has terminated: process table entry now available.
01	A system process: always in primary memory.
02	Parent is tracing process.
04	Tracing parent's signal has stopped process: parent is waiting [*ptrace*(2)].
10	Process currently in primary memory.

S (l) The state of the process:

- O Process is running on a processor. On a multiprocessor system, the process is running on processor zero.
- 1-9 Process is running on a multiprocessor. The number indicates which processor is running the process.
- a-f Process is running on a multiprocessor. The letter indicates which processor is running the process where *a* is processor 10, *b* is processor 11, and so on.
- ? Process was running on a multiprocessor. The process changed state in the middle of determining the processor number.
- S Sleeping: process is waiting for an event to complete. The process can be awakened by signals.
- D Sleeping: process is waiting for an event to complete. The process cannot be awakened by signals.
- R Runnable: process is on run queue.
- I Idle: process is being created.
- Z Zombie state: process terminated and parent not waiting.
- T Traced: process stopped by a signal because parent is tracing it.
- X SXBRK state: process is waiting for more primary memory.

UID (f,l) The user ID number of the process owner (the login name is printed under the **–f** option).

PID (all) The process ID of the process (this datum is necessary in order to kill a process).

PPID (f,l) The process ID of the parent process.

C (f,l) Processor utilization for scheduling.

PRI (l) The priority of the process (higher numbers mean lower priority).

NI (l) Nice value, used in priority computation.

ADDR (l) The memory address of a table in the *proc* structure for the process which contains the page numbers of the process' *u_block*.

SZ (l) The size (in pages or clicks) of the swappable process's image in main memory.

WCHAN (l) The address of an event for which the process is sleeping, or in SXBRK state, (if blank, the process is running).

STIME (f) The starting time of the process, given in hours, minutes, and seconds. (A process begun more than twenty-four hours before the *ps* inquiry is executed is given in months and days.)

TTY (all) The controlling terminal for the process (the message, ?, is printed when there is no controlling terminal).

TIME (all) The cumulative execution time for the process.

COMMAND (all) The command name (the full command name and its arguments are printed under the **–f** option).

A process that has exited and has a parent, but has not yet been waited for by the parent, is marked **< defunct>** .

FILES

/dev	
/dev/sxt/*	
/dev/tty*	
/dev/xt/*	terminal ("tty") names searcher files
/dev/kmem	kernel virtual memory
/dev/swap	the default swap device
/dev/smem	memory
/etc/passwd	UID information supplier
/etc/ps_data	internal data structure
/realix	system namelist

SEE ALSO

kill(1), nice(1), getty(1M).

WARNING

Things can change while *ps* is running; the snapshot it gives is only true for a split second, and it may not be accurate by the time you see it. Some data printed for defunct processes is irrelevant.

If no *termlist*, *proclist*, *uidlist*, or *grplist* is specified, *ps* checks *stdin*, *stdout*, and *stderr* in that order, looking for the controlling terminal and will attempt to report on processes associated with the controlling terminal. In this situation, if *stdin*, *stdout*, and *stderr* are all redirected, *ps* will not find a controlling terminal, so there will be no report.

On a heavily loaded system, *ps* may report an *lseek(2)* error and exit. *ps* may seek to an invalid user area address: having got the address of a process' user area, *ps* may not be able to seek to that address before the process exits and the address becomes invalid.

ps –ef may not report the actual start of a tty login session, but rather an earlier time, when a getty was last respawned on the tty line.

NAME

putbuf - kernel console printf logger

SYNOPSIS

/etc/putbuf [**-a**] [**-A**] [**-s**] [**-d** osmfile] [**-n** namelist] [**-c** corefile]

DESCRIPTION

putbuf operates in one of two modes. If the **-d** *osmfile* option is specified, *putbuf* becomes a daemon. In this mode of operation, *putbuf* continually reads osmfile, typically **/dev/osm**, and writes standard output. Normally the standard output is redirected to **/usr/adm/putbuf**. Console printfs from the kernel appear as data to be read in **/dev/osm**.

In the second form of the command (the **-d** option is not used) **/etc/putbuf** reads console messages from corefile (**/dev/smem** by default) and writes them to standard output. The **-c** option specifies what to use as a corefile and the **-n** option specifies what REAL/IX to use for symbol information.

In this second form of the command, if **-a**, **-A** and **-s** are not set, **/etc/putbuf** will read only those console messages which have not been read via **/dev/osm**. Having read these messages, **/etc/putbuf** marks a location in corefile to show that all console messages have been extracted. Should **/etc/putbuf** be called again for the same corefile it will output an "already extracted" message. The **-s** (silent) option suppresses this message.

The **-a** option causes *putbuf* to print all console messages contained in corefile that have not been read via **/dev/osm**, regardless of any previous calls to putbuf. This option also suppresses the marking of the corefile.

The **-A** option causes *putbuf* to print all console messages that it can find in corefile, even if they have already been read via **/dev/osm** or previous calls to putbuf. Console messages are kept in a cyclic buffer, and so it is possible for early messages to be overwritten: with the **-A** option *putbuf* reports how many bytes of messages, if any, have been lost. The **-A** option also suppresses the marking of the corefile.

putbuf is normally called from one of the initialization scripts held in **/etc/init.d**. See the *System Administrator's Guide* for more details.

NAME

pwd – working directory name

SYNOPSIS

pwd

DESCRIPTION

pwd prints the path name of the working (current) directory.

SEE ALSO

cd(1)

DIAGNOSTICS

"Cannot open .." and "Read error in .." indicate possible file system trouble and should be referred to your local system administrator.

NAME

rcp – remote file copy.

SYNOPSIS

rcp *file1 file2*

rcp [–**r**] file ... directory

DESCRIPTION

rcp copies files between machines. Each file or directory argument is either a remote file name of the form *rhost:path*, or a local file name (containing no ":" characters, or a " / " before any ":"s.)

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

If *path* is not a full path name, it is interpreted relative to your login directory on *rhost*. A path on a remote host may be quoted (using \ , " , or ') so that the metacharacters are interpreted remotely.

Hostnames may also take the form *rhost.rname* to use *rname* rather than the current user name on the remote host. *rcp* does not prompt for passwords; your current local user name (or *rname*) must exist on *rhost* and allow remote command execution via *remsh*(1).

rcp handles third party copies, where neither source nor target files are on the current machine.

FILES

/etc/hosts.equiv (on remote host)

$HOME/.rhosts (on remote host)

SEE ALSO

ftp(1), remsh(1), rlogin(1), cp(1)

BUGS

Does not detect all cases where the target of a copy might be a file in cases where only a directory should be legal.

Is confused by any output generated by commands in a **.profile** file on the remote host.

The –**r** option depends on *cp*(1) in order to work. In particular, SYSTEM V/88 *cp* does not handle –**r**. This will affect **rcp** –**r** only in the case where both the source and target directory are on the local machine.

NAME

real – echo the real device in the **permissions** file for a given alias

SYNOPSIS

real [**-r**] [*name*]

DESCRIPTION

real has an optional argument *name*. This argument is used to search the **permissions** file and determine the real device that would be accessed. The file is searched, and when *name* matches either the *slice* or *alias* field of a **permissions** file entry, the pathname of the real device is echoed. This device pathname is used by the disk access routines *chk*, *mnt* and *umnt*.

The default value for *name* is **floppy**.

The optional **-r** argument indicates that the real device desired is the raw device. This device pathname is used by the routines *dcpy* and *fs*.

FILES

/etc/real
/etc/filesys **permissions** file

SEE ALSO

filesys(4) in the *Programmer's Reference Manual.*

NAME

regcmp – regular expression compile

SYNOPSIS

regcmp [–] *files*

DESCRIPTION

The *regcmp* command performs a function similar to *regcmp*(3X) and, in most cases, precludes the need for calling *regcmp*(3X) from C programs. This saves on both execution time and program size. The command *regcmp* compiles the regular expressions in *file* and places the output in *file*.**i**. If the – option is used, the output will be placed in *file*.**c**. The format of entries in *file* is a name (C variable) followed by one or more blanks followed by a regular expression enclosed in double quotes. The output of *regcmp* is C source code. Compiled regular expressions are represented as **extern char** vectors. *file*.**i** files may thus be *included* in C programs, or *file*.**c** files may be compiled and later loaded. In the C program which uses the *regcmp* output, *regex*(*abc*,*line*) will apply the regular expression named *abc* to *line*. Diagnostics are self-explanatory.

EXAMPLES

```
name    "([A–Za–z][A–Za–z0–9_]*)$0"

telno   "\({0,1}([2–9][01][1–9])$0\){0,1} *"
        "([2–9][0–9]{2})$1[ –]{0,1}"
        "([0–9]{4})$2"
```

In the C program that uses the *regcmp* output,

```
regex(telno, line, area, exch, rest)
```

will apply the regular expression named *telno* to *line*.

SEE ALSO

regcmp(3X)

NAME

remsh – remote shell

SYNOPSIS

remsh rhost [–l *username*] [–ec] [–n] [command]

DESCRIPTION

remsh connects to the specified host, and executes the specified command. *remsh* copies its standard input to the remote command, the standard output of the remote command to its standard output, and the standard error of the remote command to its standard error. Interrupt, quit and terminate signals are propagated to the remote command; *remsh* normally terminates when the remote command does.

The remote username used is the same as your local username, unless you specify a different remote name with the –l option. This remote user named by **-l** must have authorized (in the remote user's **.rhosts** file) *remsh* access for you; there is no provision for specifying a password with a command.

The **-n** option specifies that standard input is not to be copied to the standard input of the remote command, which will act as if its standard input was connected to **/dev/null**.

If you omit command, then instead of executing a single command, you will be logged in on the remote host using *rlogin*(1). In this case you may use the **-e** option to specify an escape character. See the *rlogin*(1) man page for details of this facility. If command is specified, any **-e** parameter is redundant and will be ignored.

Shell metacharacters which are not quoted are interpreted on local machine, while quoted metacharacters are interpreted on the remote machine. Thus the command:

remsh *otherhost* **cat** *remotefile* >> *localfile*

appends the remote file *remotefile* to the local file *localfile*, while:

remsh *otherhost* **cat** *remotefile* ">>" *otherremotefile*

appends *remotefile* to *otherremotefile*.

Host names are given in the file **/etc/hosts**. Each host has one standard name (the first name given in the file), which may be rather long and unambiguous, and optionally one or more nicknames.

FILES

/etc/hosts
/etc/hosts.equiv (on remote host)
$HOME/.rhosts (on remote host)
/usr/bin/rlogin

SEE ALSO

rlogin(1), hosts(4)

NOTES

rlogin must be located in the standard search path (or in **/usr/bin/rlogin**) in order to be found by a command-less invocation of *remsh*.

You cannot run an interactive command (like *vi*(1)); use *rlogin*(1).

NAME

rlogin – remote login

SYNOPSIS

rlogin rhost [**–ex**] [**–l** *username*] [**–8**]

DESCRIPTION

rlogin connects your terminal on the current local host system lhost to the remote host system rhost.

Each host has a file **/etc/hosts.equiv** which contains a list of rhosts with which it shares account names. (The host names must be the standard names as described in *remsh*(1).) When you *rlogin* as the same user on an equivalent host, you may not need to give a password (depending on the restrictions imposed by *login*(1) on the remote host). Each user may also have a private equivalence list in a file **.rhosts** in their login directory. Each line in this file should contain a rhost and a username separated by a space, giving additional cases where logins without passwords may be permitted. If the originating user is not equivalent to the remote user, then a login and password will be prompted for on the remote machine as in *login*(1). To avoid some security problems, the **.rhosts** file must be owned by either the remote user or root and may not be a symbolic link.

Your remote terminal type is the same as your local terminal type (as given in your environment TERM variable). All echoing takes place at the remote site, so that (except for delays) the *rlogin* is transparent. Flow control via ^S and ^Q and flushing of input and output on interrupts are handled properly. A line of the form ~ . disconnects from the remote host, where ~ is the escape character. A different escape character may be specified by the –e option. There is no space separating this option flag and the argument character {x}.

While connected, an input string of the form ~ ! introduces an arbitrary command to be executed locally (a la *cu*(1)). ~ ! alone spawns a local shell. ~ ! must be preceded by a <CR> (i.e., must be the first two characters of a new input line).

The optional argument -8 allows an eight bit input data path at all times; otherwise any parity bits are stripped.

SEE ALSO

remsh(1)

FILES

/etc/hosts.equiv
$HOME/.rhosts

BUGS

More terminal characteristics should be propagated.

NAME

rm, rmdir – remove files or directories

SYNOPSIS

rm [**–f**] [**–i**] *file* ...

rm **–r** [**–f**] [**–i**] *dirname* ... [*file* ...]

rmdir [**–p**] [**–s**] *dirname* ...

DESCRIPTION

rm removes the entries for one or more files from a directory. If an entry was the last link to the file, the file is destroyed. If a file has no write permission and the standard input is a terminal, the full set of permissions (in octal) for the file are printed followed by a question mark. This is a prompt for confirmation. If the answer begins with **y** (for yes), the file is deleted, otherwise the file remains.

When the target is a symbolic link, *rm* removes the symbolic link itself, not the file that the link points to.

Note that if the standard input is not a terminal, the command will operate as if the **–f** option is in effect.

Three options apply to *rm*:

–f

This option causes the removal of all files (whether write-protected or not) in a directory without prompting the user. In a write-protected directory, however, files are never removed (whatever their permissions are), but no messages are displayed. If the removal of a write-protected directory is attempted, this option will not suppress an error message.

–r

This option causes the recursive removal of any directories and subdirectories in the argument list. The directory will be emptied of files and removed. Note that the user is normally prompted for removal of any write-protected files which the directory contains. The write-protected files are removed without prompting, however, if the **–f** option is used, or if the standard input is not a terminal and the **–i** option is not used.

If the removal of a non-empty, write-protected directory is attempted, the command will always fail (even if the **–f** option is used), resulting in an error message.

–i

With this option, confirmation of removal of any write-protected file occurs interactively. It overrides the **–f** option and remains in effect even if the standard input is not a terminal.

Two options apply to *rmdir*:

–p

This option allows users to remove the directory *dirname* and its parent directories which become empty. A message is printed on standard output as to whether the whole path is removed or part of the path remains for some reason.

–s

This option is used to suppress the message printed on standard error when **–p** is in effect.

DIAGNOSTICS

All messages are generally self-explanatory.
It is forbidden to remove the files "." and ".." in order to avoid the consequences of inadvertently doing something like the following:

```
rm -r .*
```

Both *rm* and *rmdir* return exit codes of 0 if all the specified directories are removed successfully. Otherwise, they return a non-zero exit code.

SEE ALSO

unlink(2), rmdir(2).

NAME

rmdel – remove a delta from an SCCS file

SYNOPSIS

rmdel –r*SID files*

DESCRIPTION

rmdel removes the delta specified by the *SID* from each named SCCS file. The delta to be removed must be the newest (most recent) delta in its branch in the delta chain of each named SCCS file. In addition, the specified must *not* be that of a version being edited for the purpose of making a delta (i. e., if a **p-file** (see *get*(1)) exists for the named SCCS file, the specified must *not* appear in any entry of the **p-file**).

The **–r** option is used for specifying the *SID* level of the delta to be removed.

If a directory is named, *rmdel* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the 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; non-SCCS files and unreadable files are silently ignored.

Simply stated, they are either (1) if you make a delta you can remove it; or (2) if you own the file and directory you can remove a delta.

FILES

x.file (see *delta*(1))
z.file (see *delta*(1))

SEE ALSO

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

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

rpcgen – an RPC protocol compiler

SYNOPSIS

rpcgen *infile*
rpcgen **–h** [**-o** *outfile*] [*inputfile*]
rpcgen **–c** [**-o** *outfile*] [*infile*]
rpcgen **–s** *transport* [**-o** *outfile*] [*infile*]
rpcgen **–l** [**-o** *outfile*] [*infile*]
rpcgen **–m** [**-o** *outfile*] [*infile*]

DESCRIPTION

rpcgen is a tool that generates **C** code to implement an RPC protocol. The input to *rpcgen* is a language similar to **C** known as RPC Language (Remote Procedure Call Language). Information about the syntax of RPC Language is available in *RFC1057*.

rpcgen is normally used as in the first synopsis, where it takes an input file and generates four output files. If the *infile* is named **proto.x**, then *rpcgen* generates a header file in **proto.h**, XDR routines in **proto_xdr.c**, server-side stubs in **proto_svc.c**, and client-side stubs in **proto_clnt.c**.

The other synopses shown above are used when you want to generate not all the output files, but only a particular one. Their usage is described in the *OPTIONS* section.

The **C** language preprocessor, *cpp*(1), is run on all input files before they are actually interpreted by *rpcgen*, so all the **cpp** directives are legal within an *rpcgen* input file. For each type of output file, *rpcgen* defines a special *cpp* symbol for use by the *rpcgen* programmer:

RPC_HDR
defined when compiling into header files

RPC_XDR
defined when compiling into XDR routines

RPC_SVC
defined when compiling into server-side stubs

RPC_CLNT
defined when compiling into client-side stubs

In addition, *rpcgen* does a little preprocessing of its own. Any line beginning with the symbol **%** is passed directly into the output file, uninterpreted by *rpcgen*.

You can customize some of your XDR routines by leaving some data types undefined. For every data type that is undefined, *rpcgen* assumes that there exists a routine with the name **xdr_** prepended to the name of the undefined type.

OPTIONS

-c Compile into XDR routines.

-h Compile into **C** data definitions (a header file).

-l Compile into client-side stubs.

-s *transport*
Compile into server-side stubs, using the given transport. The supported transports are **udp** and **tcp**. This option may be invoked more than once so as to compile a server that serves multiple transports.

-m Compile into server-side stubs, but do not produce a *main*() routine. This option is useful if you want to supply your own *main*().

-o *outfile*

Specify the name of the output file. If none is specified, standard output is used (**-c**, **-h**, **-l**, and **-s** modes only).

USAGE

RPCL Syntax Summary

This summary of RPCL syntax, which is used for *rpcgen* input, is intended more for aiding comprehension than as an exact statement of the language.

Primitive Data Types

[unsigned] char
[unsigned] short
[unsigned] int
[unsigned] long
unsigned
float
double
void
bool

Except for the added boolean data-type **bool**, RPCL is identical to **C**. *rpcgen* converts **bool** declarations in the output header file (literally it is converted to a **bool_t** that has been **#defined** to be an **int**). Also, **void** declarations may appear only inside of **union** and **program** definitions. For those averse to typing the prefix **unsigned**, the abbreviations **u_char**, **u_short**, **u_int** and **u_long** are available.

Declarations

RPCL allows only three kinds of declarations:

declaration:

simple-declaration
type-name object-ident

pointer-declaration
*type-name *object ident*

vector-declaration
type-name object ident[size]

where:

size can be either an integer or a symbolic constant.

RPCL declarations contain both limitations and extensions with respect to **C**. The limitations are that you cannot declare multidimensional arrays or pointers-to-pointers in-line (you can still declare them though, using **typedef**).

There are two extensions:

Opaque data is declared as a vector:

opaque *object-ident* [*size*]

In the protocol, this results in an object of *size* bytes. Note that this is *not* the same as a declaration of *size* characters, since XDR characters are 32 bits. Opaque declarations are compiled in the output header file into character array declarations of *size* bytes.

Strings are special and are declared as a vector declaration:

string *object-ident* [*size*]

If **max-size** is unspecified, then there is essentially no limit to the maximum length of the string. String declarations get compiled into the following:

char **object-ident*

Type Definitions

The only way to generate an XDR routine is to define a type. For every type *zetype* you define, there is a corresponding XDR routine named *xdr_zetype*.

There are six different ways to define a type:

type-definition:
typedef
enumeration-def
structure-def
variable-length-array-def
discriminated-union-def
program-def

The first three are very similar to their **C** namesakes. **C** does not have a formal mechanism to define variable-length arrays and XDR unions are quite different from their **C** counterparts. Program definitions are not really type definitions, but they are useful.

You may not nest XDR definitions. For example, the following causes *rpcgen* to choke:

```
struct dontdoit {
        struct ididit {
                int oops;
        } sorry;
        enum ididitagain { OOPS, WHOOPS } iapologize;
};
```

Typedefs

An XDR **typedef** looks as:

typedef:
typedef *declaration* ;

The *object-ident* part of *declaration* is the name of the new type, whereas the type-name is the name of the type from which it is derived.

Enumeration Types

The syntax is:

enumeration-def:
enum *enum-ident* {
enum-list
};
enum-list:
enum-symbol-ident [= *assignment*]
enum-symbol-ident [= *assignment*], *enum-list*

assignment may be either an integer or a symbolic constant.

If there is no explicit assignment, then the implicit assignment is the value of the previous enumeration plus 1. If not explicitly assigned, the first enumeration receives the value 0.

Structures

structure-def:
struct *struct-ident* {
declaration-list
};

declaration-list:
declaration
declarations ; *declarations-list*

Variable-Length Arrays

variable-length-array-def:
array *array-ident* **{**
unsigned *length-identifier;*
vector-declaration;
};

A variable length array definition looks much like a structure definition. For example:

```
array mp_int {
        unsigned len;
        short val[MAX_MP_LENGTH];
};
```

This is compiled into:

```
struct mp_int {
        unsigned len;
        short *val;
};
typedef struct mp_int mp_int;
```

Discriminated Unions

discriminated-union-def:
union *union-ident* **switch (** *discriminant-declaration*) **{**
case-list
[**default** : *declaration* ;]
};

case-list
case *case-ident* : *declaration* ;
case *case-ident* : *declaration* ; *case-list*

discriminant-declaration:
declaration

The union declaration looks like a cross between a C-union and a C-switch. For example:

```
union net_object switch (net_kind kind) {
case MACHINE:
        struct sockaddr_in sin;
case USER:
        int uid;
default:
        string whatisit;
};
```

Compiles into:

```
struct net_object {
        net_kind kind;
        union {
                struct sockaddr_in sin;
                int uid;
                string whatisit;
        } net_object;
```

```
};
typedef struct net_object net_object;
```

Note that the name of the union component of the output struct is the same as the name of the type itself.

Program Definitions

program-def:
program *program-ident* {
version-list
} = *program-number* ;

version-list:
version
version *version-list*

version:
version *version-ident* {
procedure-list
} = *version-number*;

procedure-list:
procedure-declaration
procedure-declaration procedure-list

procedure-declaration:
type-name procedure-ident (type-name)=procedure-number;

Program definitions look like nothing you have ever seen before, so we turn to an example to explain them. Suppose you wanted to create server that could get or set the date. The declaration might look like this:

```
program DATE_PROG {
        version DATE_VERS {
                date DATE_GET(timezone) = 1;
                void DATE_SET(date) = 2;
                /* Greenwich mean time */
        } = 1;
} = 100;
```

In the header file, this compiles into the following:

```
#define DATE_PROG 100
#define DATE_VERS 1
#define DATE_GET 1
#define DATE_SET 2
```

These **defines** are intended for use by the client program to reference the remote procedures.

If you are using *rpcgen* to compile your server, there are some important things for you to know. The server interfaces to your local procedures by expecting a C function with the same name as that in the program definition, but in all lowercase letters and followed by the version number. Here is the local procedure that implements DATE_GET:

```
/* This function returns a pointer to the results */
/* It always takes a pointer to the arguments */
date *
date_get_1(tz)
        timezone *tz;
{
```

```
        static date d;  /* must be static!     */
        /*
         * figure out the date
         * and store it in d
         */
        return(&d);
}
```

The name of the routine is the same as the **#defined** name, but in all lowercase letters and followed by the version number. XDR recursively frees the argument after getting the results from your local procedure, so you should copy from the argument any data that you need between calls. However, XDR neither allocates nor frees your results. You must take care of their storage yourself.

Make Inference Rules For Compiling XDR

It is possible to set up suffix transformation rules in make(1) for compiling XDR routines and header files. The convention is that RPCL protocol files have the extension .x. The make rules to do this are:

```
.SUFFIXES: .x
    .x.c:
        rpcgen -c $< -o $0
.x.h:
        rpcgen -h $< -o $0
```

FILES

/usr/bin/rpcgen

SEE ALSO

The "RPC Programming" chapter in the *ONC User's Guide*

BUGS

Nesting is not supported. As a workaround to achieve the same effect, structures can be declared at top level and their names used inside other structures.

Name clashes can occur when using program definitions, since the apparent scoping does not really apply. Most of these can be avoided by giving unique names for programs, versions, procedures, and types.

NAME

rtgroups – view the realtime privileged groups list

SYNOPSIS

rtgroups [-n system]

DESCRIPTION

rtgroups lists the groups given in the last call to setrtgroups(1M) or setrtgroups(2) by group name. If the group name is not in the **/etc/group** file, the gid is printed.

If the system that was booted on is not */realix*, the optional parameter *system* should be specified.

rtgroups will fail or may give erroneous information if the optional *system* is not the system that was booted.

FILES

/etc/group
/dev/smem

SEE ALSO

rtusers(1), setrt(2), setrtusers(1M), setrtusers(2), setrtgroups(1M), setrtgroups(2).

NAME

rtusers – view the realtime privileged users list

SYNOPSIS

rtusers [-n system]

DESCRIPTION

rtusers lists the users given in the last call to setrtusers(1M) or setrtusers(2) by user name. If the users name is not in the **/etc/passwd** file, the uid is printed.

If the system that was booted on is not */realix*, the optional parameter *system* should be specified.

rtusers will fail or may give erroneous information if the optional *system* is not the system that was booted.

FILES

/etc/passwd
/dev/smem

SEE ALSO

rtgroups(1), setrt(2), setrtusers(1M), setrtusers(2), setrtgroups(1M), setrtgroups(2).

NAME

ruptime – show host status of local machines

SYNOPSIS

ruptime [**–a**] [**–r**] [**–l**] [**–t**] [**–u**]

DESCRIPTION

ruptime gives a status line like *uptime* for each machine on some local network; these are formed from packets broadcast by each host on the network once a minute.

Machines for which no status report has been received for 11 minutes are shown as being down.

Users idle an hour or more are not counted unless the **–a** flag is given.

Normally, the listing is sorted by host name. The **–l**, **–t**, and **–u** flags specify sorting by load average, uptime, and number of users, respectively. The **–r** flag reverses the sort order.

FILES

/usr/bin/ruptime

SEE ALSO

rwho(1N)

NAME

rwho – who's logged in on local machines

SYNOPSIS

rwho [–a]

DESCRIPTION

The *rwho* command produces output similar to **who**, but for all machines on the local network. If no report has been received from a machine for 5 minutes, then *rwho* assumes the machine is down, and does not report users last known to be logged into that machine.

If a user has not typed to the system for a minute or more, then *rwho* reports this idle time. If a user has not typed to the system for an hour or more, then the user is omitted from the output of *rwho* unless the **–a** flag is given.

FILES

/usr/bin/rwho

SEE ALSO

ruptime(1N), rwhod(1M)

BUGS

This is unwieldy when the number of machines on the local net is large.

NAME

sact – print current SCCS file editing activity

SYNOPSIS

sact *files*

DESCRIPTION

sact informs the user of any impending deltas to a named SCCS file. This situation occurs when *get*(1) with the –e option has been previously executed without a subsequent execution of *delta*(1). If a directory is named on the command line, *sact* behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of – is given, the standard input is read with each line being taken as the name of an SCCS file to be processed. The output for each named file consists of five fields separated by spaces.

Field 1
: specifies the SID of a delta that currently exists in the SCCS file to which changes will be made to make the new delta.

Field 2
: specifies the SID for the new delta to be created.

Field 3
: contains the logname of the user who will make the delta (i.e., executed a *get* for editing).

Field 4
: contains the date that **get** –e was executed.

Field 5
: contains the time that **get** –e was executed.

SEE ALSO

delta(1), get(1), unget(1)

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

sar – system activity reporter

SYNOPSIS

sar [**–ubdycwaqvmprDSABCEIT**] [**–o** file] t [n]

sar [**–ubdycwaqvmprDSABCEIT**] [**–s** time] [**–e** time] [**–i** sec] [**–f** file]

DESCRIPTION

sar, in the first instance, samples cumulative activity counters in the operating system at *n* intervals of *t* seconds, where *t* should be 5 or greater. If the **–o** option is specified, it saves the samples in *file* in binary format. The default value of *n* is 1. In the second instance, with no sampling interval specified, **sar** extracts data from a previously recorded *file*, either the one specified by **–f** option or, by default, the standard system activity daily data file **/usr/adm/sa/sa***dd* for the current day *dd*. The starting and ending times of the report can be bounded via the **–s** and **–e** *time* arguments of the form *hh*[:*mm*[:*ss*]]. The **–i** option selects records at *sec* second intervals. Otherwise, all intervals found in the data file are reported.

In either case, subsets of data to be printed are specified by option:

–u Report CPU utilization (the default):
%usr, %sys, %wio, %idle – portion of time running in user mode, running in system mode, idle with some process waiting for block I/O, and otherwise idle. When used with **–D**, %sys is split into percent of time servicing requests from remote machines (%sys remote) and all other system time (%sys local).

–b Report buffer activity:
bread/s, bwrit/s – transfers per second of data between system buffers and disk or other block devices;
lread/s, lwrit/s – accesses of system buffers;
%rcache, %wcache – cache hit ratios, i. e., (1–bread/lread) as a percentage;
pread/s, pwrit/s – transfers via raw (physical) device mechanism.
Note that the real cache hit ratio is likely to be lower than reported, owing to the limitations of the statistics gathered. These limitations are believed to be common to all UNIX systems. They have not been eradicated in REAL/IX for reasons of performance and comparability with other UNIX implementations. As an example, a cache hit is recorded if the process does not have to initiate a physical I/O operation. This means that a hit may be recorded when data is not actually in the cache, a previous I/O operation still being in progress.

–d Report activity for each block device, e. g., disk or tape drive. When data is displayed, the device specification *dsk-* is generally used to represent a disk drive. The device specification used to represent a tape drive is machine dependent. The activity data reported is:
%busy, avque – portion of time device was busy servicing a transfer request, average number of requests outstanding during that time;
r+w/s, blks/s – number of data transfers from or to device, number of bytes transferred in 512-byte units;
avwait, avserv – average time in ms. that transfer requests wait idly on queue, and average time to be serviced (which for disks includes seek, rotational latency and data transfer times).
Note that the number of blocks/sec will often be higher than the number of reads and writes per second. There are two reasons for this. Firstly, the disk may be supporting a file system with a larger block size. Secondly, REAL/IX will concatenate I/O operations for contiguous blocks wherever possible. In particular, there is a multiblock read mode that is normally used when files are accessed sequentially.

-y Report TTY device activity:
rawch/s, canch/s, outch/s – input character rate, input character rate processed by canon, output character rate;
rcvin/s, xmtin/s, mdmin/s – receive, transmit and modem interrupt rates.

-c Report system calls:
scall/s – system calls of all types;
sread/s, swrit/s, fork/s, exec/s – specific system calls;
rchar/s, wchar/s – characters transferred by read and write system calls. When used with **-D**, the system calls are split into incoming, outgoing, and strictly local calls.

-w Report system swapping and switching activity:
swpin/s, swpot/s, bswin/s, bswot/s – number of transfers and number of 512-byte units transferred for swapins and swapouts (including initial loading of some programs);
pswch/s – process switches.

-a Report use of file access system routines:
iget/s, namei/s, dirblk/s.

-q Report average queue length while occupied, and % of time occupied:
runq-sz, %runocc – run queue of processes in memory and runnable;
swpq-sz, %swpocc – swap queue of processes swapped out but ready to run.

-v Report status of process, i-node, file tables, streams:
text-sz, proc-sz, inod-sz, file-sz, lock-sz, stream-sz – entries/size for each table, evaluated once at sampling point;
ov – overflows that occur between sampling points for each table.

-m Report message and semaphore activities:
msg/s, sema/s – primitives per second.

-p Report paging activities:
vflt/s – address translation page faults (valid page not in memory);
pflt/s – page faults from protection errors (illegal access to page) or "copy-on-writes";
pgfil/s – vflt/s satisfied by page-in from file system;
rclm/s – valid pages reclaimed for free list.

-r Report unused memory pages and disk blocks:
freemem – average pages available to user processes;
freeswap – disk blocks available for process swapping.

-D Report Remote File Sharing activity:
When used in combination with **-u** or **-c**, it causes **sar** to produce the remote file sharing version of the corresponding report. **-u** is assumed when neither **-u** or **-c** is specified.

-S Report server and request queue status:
Average number of Remote File Sharing servers on the system (serv/lo-hi), % of time receive descriptors are on the request queue (request %busy), average number of receive descriptors waiting for service when queue is occupied (request avg lgth), % of time there are idle servers (server %avail), average number of idle servers when idle ones exist (server avg avail).

-A Report all data. Equivalent to **-udqbwcayvmprSDBCEIT**.

-B Report use of binary semaphores:
bsemas - number of binary semaphores in use;
bsemaprocs - number of processes using binary semaphores;
maxbsemas/proc - max number of binary semaphores used by any process;
bsgets - number of binary semaphores used by processes (differs from bsemas in that two processes can use the same semaphore in which case bsemas would equal 1 and bsgets would equal 2).

-C Report use of connected interrupts:
cintrpts - number of connected interrupts in use;
ciprocs - number of processes using connected interrupts;
maxcis/proc - max number of connected interrupts used by any process.

-E Report use of events:
events - number of events in use;
evtprocs - number of processes using events;
maxevts/proc - max number of events used by any process.

-I Report use of asynchronous I/O:
aioblks - number of async I/O blocks in use;
aioprocs - number of processes using async I/O;
maxaioblks/proc - max number of async I/O blocks used by any process.

-T Report use of timers:
timers - number of timers in use;
timprocs - number of processes using timers;
maxtims/proc - max number of timers used by any process.

EXAMPLES

To see today's CPU activity so far:

```
sar
```

To watch CPU activity evolve for 10 minutes and save data:

```
sar -o temp 60 10
```

To later review disk and tape activity from that period:

```
sar -d -f temp
```

FILES

/usr/adm/sa/sa*dd*
daily data file, where *dd* are digits representing the day of the month.

SEE ALSO

sar(1M).

NAME

sccsdiff – compare two versions of an SCCS file

SYNOPSIS

sccsdiff **–r***SID*1 **–r***SID2* [**–p**] [**–s***n*] *files*

DESCRIPTION

sccsdiff compares two versions of an SCCS file and generates the differences between the two versions. Any number of SCCS files may be specified, but arguments apply to all files.

–r*SID?*

SID1 and *SID2* specify the deltas of an SCCS file that are to be compared. Versions are passed to *bdiff*(1) in the order given.

–p

pipe output for each file through *pr*(1).

–s*n*

n is the file segment size that *bdiff* will pass to *diff*(1). This is useful when *diff* fails due to a high system load.

FILES

/tmp/get?????

Temporary files

SEE ALSO

bdiff(1), get(1), help(1), pr(1).

DIAGNOSTICS

"*file*: No differences" If the two versions are the same.
Use *help*(1) for explanations.

NAME

script – make typescript of terminal session

SYNOPSIS

script [**–a**] [file]

DESCRIPTION

script makes a typescript of everything printed on your terminal. The typescript is written to *file*, or appended to *file* if the **–a** option is given. It can be sent to the line printer later with *lp*. If no file name is given, the typescript is saved in the file *typescript*.

The script ends when the forked shell exits.

This program is useful when using a crt and a hard-copy record of the dialog is desired, as for a student handing in a program that was developed on a crt when hard-copy terminals are in short supply.

BUGS

script places **everything** in the log file. This is not what the naive user expects.

NAME

sdb – symbolic debugger

SYNOPSIS

sdb [**–u**] [**–w**] [**–W**] [*objfil* [*corfil* [*directory-list*]]]

DESCRIPTION

sdb is a symbolic debugger that can be used with C programs compiled with M88000 OCS-compliant C compilers. It may be used to examine their object files and core files and to provide a controlled environment for their execution.

objfil is an executable program file that has been compiled with the **–g** (debug) option. If it has not been compiled with the **–g** option, the symbolic capabilities of *sdb* is limited, but the file can still be examined and the program debugged. The default for *objfil* is **a.out**. *corfil* is assumed to be a core image file produced after executing *objfil*; the default for *corfil* is **core**. The core file need not be present. A **–** in place of *corfil* will force *sdb* to ignore any core image file. The colon separated list of directories (*directory-list*) is used to locate the source files used to build *objfil*.

It is useful to know that at any time there is a *current line* and *current file*. If *corfil* exists, they are initially set to the line and file containing the source statement at which the process terminated. Otherwise, they are set to the first line in *main*( ). The current line and file may be changed with the source file examination commands.

By default, warnings are provided if the source files used in producing *objfil* cannot be found, or are newer than *objfil*. This checking feature and the accompanying warnings may be disabled by the use of the **–W** flag.

Names of variables are written just as they are in C. *sdb* does not truncate names. Variables local to a procedure may be accessed using the form *procedure*:*variable*. If no procedure name is given, the procedure containing the current line is used by default.

Current compilers prepend an underscore character to all variable names. *sdb* requires the user to reference variables by their original C names, automatically stripping the leading underscore from names as the symbol table is read in. If the **–u** flag is given, leading underscores are not stripped, and *sdb* always requires the leading underscore on variable names and always prints names in the same fashion.

It is also possible to refer to structure members as *variable.member*, pointers to structure members as *variable–>member* and array elements as *variable*[*number*]. Pointers may be dereferenced by using the form *pointer*[0]. Combinations of these forms may also be used.

F77 common variables may be referenced by using the name of the common block instead of the structure name. Blank common variables may be named by the form *.variable*. A number may be used in place of a structure variable name, in which case the number is viewed as the address of the structure, and the template used for the structure is that of the last structure referenced by *sdb*. An unqualified structure variable may also be used with various commands.

Generally, *sdb* interprets a structure as a set of variables. Thus, *sdb* displays the values of all the elements of a structure when it is requested to display a structure. An exception to this interpretation occurs when displaying variable addresses. An entire structure does have an address, and it is this value *sdb* displays, not the addresses of individual elements.

Elements of a multidimensional array may be referenced as *variable* [*number*][*number*]..., or as *variable* [*number*,*number*,...]. In place of *number*, the form *number*;*number* may be used to indicate a range of values, * may be used to indicate all legitimate values for that subscript, or subscripts may be omitted entirely if they are the last subscripts and the full range of values is desired. As with structures, *sdb* displays all the values of an array or of the section of an array if trailing subscripts are omitted. It displays only the address of the array itself or of the section specified by the user if subscripts are omitted. A multidimensional parameter in an F77 program

cannot be displayed as an array, but it is actually a pointer, whose value is the location of the array. The array itself can be accessed symbolically from the calling function.

A particular instance of a variable on the stack may be referenced by using the form *procedure:variable,number*. If no procedure is specified, the procedure currently executing is used by default. All the variations mentioned in naming variables may be used. *number* is the occurrence of the specified procedure on the stack, counting the top, or most current, as the first. If number is 0 or not given, all instances are printed.

It is also possible to specify a variable by its address. All forms of integer constants which are valid in C may be used, so that addresses may be input in decimal, octal, or hexadecimal.

Line numbers in the source program are referred to as *file-name:number* or *procedure:number*. In either case the number is relative to the beginning of the file. If no procedure or file name is given, the current file is used by default. If no number is given, the first line of the named procedure or file is used.

While a process is running under *sdb*, all addresses refer to the executing program; otherwise they refer to *objfil* or *corfil*. An initial argument of −w permits overwriting locations in *objfil*.

Addresses

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

b1<=address<e1

file address=address+f1−b1

otherwise

b2<=address<e2

file address=address+f2−b2,

Otherwise, the requested *address* is not legal. In some cases (e.g., for programs with separated I and D space), the two segments for a file may overlap.

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

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

Commands

The commands for examining data in the program are:

t
Print a stack trace of the terminated or halted program.

T
Print the top line of the stack trace.

variable/*clm*

Print the value of *variable* according to length *l* and format *m*. A numeric count *c* indicates that a region of memory, beginning at the address implied by *variable*, is to be displayed. The length specifiers are:

b	one byte
h	two bytes (half word)
l	four bytes (long word)

Legal values for *m* are:

c	character
d	decimal
u	decimal, unsigned
o	octal
x	hexadecimal
f	32-bit single precision floating point
g	64-bit double precision floating point
s	Assume *variable* is a string pointer and print characters starting at the address pointed to by the variable.
a	Print characters starting at the variable's address. This format may not be used with register variables.
p	pointer to procedure
i	disassemble machine-language instruction with addresses printed numerically and symbolically.
I	disassemble machine-language instruction with addresses just printed numerically.

Length specifiers are only effective with the **c**, **d**, **u**, **o** and **x** formats. Any of the specifiers, *c*, *l*, and *m*, may be omitted. If all are omitted, *sdb* chooses a length and a format suitable for the variable's type as declared in the program. If *m* is specified, this format is used for displaying the variable. A length specifier determines the output length of the value to be displayed, sometimes resulting in truncation. A count specifier *c* tells *sdb* to display that many units of memory, beginning at the address of *variable*. The number of bytes in one such unit of memory is determined by the length specifier *l*, or if no length is given, by the size associated with the *variable*. If a count specifier is used for the **s** or **a** command, then that many characters are printed. Otherwise, successive characters are printed until either a **NULL** byte is reached or 128 characters are printed. The last variable may be redisplayed with the command **./**.

The *sh*(1) metacharacters * and ? may be used within procedure and variable names, providing a limited form of pattern matching. If no procedure name is given, variables local to the current procedure and global variables are matched; if a procedure name is specified then only variables local to that procedure are matched. To match only global variables, the form *:pattern* is used.

linenumber?*lm*
variable:?*lm*

Print the value at the address from **a.out** or I space given by *linenumber* or *variable* (procedure name), according to the format *lm*. The default format is 'i'.

variable=*lm*

linenumber=lm
number=lm
Print the address of *variable* or *linenumber*, or the value of *number*, in the format specified by *lm*. If no format is given, then **lx** is used. The last variant of this command provides a convenient way to convert between decimal, octal and hexadecimal.

variable!value
Set *variable* to the given *value*. The value may be a number, a character constant or a variable. The value must be well defined; expressions which produce more than one value, such as structures, are not allowed. Character constants are denoted *'character*.

Numbers are viewed as integers unless a decimal point or exponent is used. In this case, they are treated as having the type double. Registers are viewed as integers. The *variable* may be an expression that indicates more than one variable, such as an array or structure name. If the address of a variable is given, it is regarded as the address of a variable of type *int*. C conventions are used in any type conversions necessary to perform the indicated assignment.

x
Print the machine registers and the current machine-language instruction.

X
Print the current machine-language instruction.

The commands for examining source files are:

e *procedure*

e *file-name*

e *directory/*

e *directory file-name*
The first two forms set the current file to the file containing *procedure* or to *file-name*. The current line is set to the first line in the named procedure or file. Source files are assumed to be in *directory*. The default is the current working directory. The latter two forms change the value of *directory*. If no procedure, filename, or directory is given, the current procedure name and filename are reported.

/regular expression/
Search forward from the current line for a line containing a string matching *regular expression* as in *ed*(1). The trailing / may be deleted.

?regular expression?
Search backward from the current line for a line containing a string matching *regular expression* as in *ed*(1). The trailing ? is required to resolve ambiguous use (e.g., **?c/d**, could be interpreted to print the value of **ac**, **bc**, **cc**, in decimal, or a reverse search for **c/d**).

p
Print the current line.

z
Print the current line followed by the next 9 lines. Set the current line to the last line printed.

w
Window. Print the 10 lines around the current line.

number
Set the current line to the given line number. Print the new current line.

count+
Advance the current line by *count* lines. Print the new current line.

count–
Retreat the current line by *count* lines. Print the new current line.

The commands for controlling the execution of the source program are:

count **r** *args*
count **R**
Run the program with the given arguments. The *r* command with no arguments reuses the previous arguments to the program while the *R* command runs the program with no arguments. An argument beginning with < or > causes redirection for the standard input or output, respectively. If *count* is given, it specifies the number of breakpoints to be ignored.

linenumber **c** *count*
linenumber **C** *count*
Continue after a breakpoint or interrupt. If *count* is given, the program will stop when *count* breakpoints have been encountered. The signal which caused the program to stop is reactivated with the **C** command and ignored with the **c** command. If a line number is specified then a temporary breakpoint is placed at the line and execution is continued. The breakpoint is deleted when the command finishes.

linenumber **g** *count*
Continue after a breakpoint with execution resumed at the given line. If *count* is given, it specifies the number of breakpoints to be ignored.

s *count*
S *count*
Single step the program through *count* lines. If no count is given then the program is run for one line. **S** is equivalent to **s** except it steps through procedure calls.

i
I Single step by one machine-language instruction. The signal which caused the program to stop is reactivated with the **I** command and ignored with the **i** command.

variable$**m** *count*
address:**m** *count*
Single step (as with **s**) until the specified location is modified with a new value. If *count* is omitted, it is effectively infinity. *Variable* must be accessible from the current procedure. Since this command is done by software, it can be very slow.

level **v**
Toggle verbose mode, for use when single stepping with **S**, **s** or **m**. If *level* is omitted, then just the current source file and/or subroutine name is printed when either changes. If *level* is 1 or greater, each C source line is printed before it is executed; if *level* is 2 or greater, each assembler statement is also printed. A **v** turns verbose mode off if it is on for any level.

k
Kill the program being debugged.

procedure(arg1,arg2,...)
procedure(arg1,arg2,...)/*m*
Execute the named procedure with the given arguments. Arguments can be integer, character or string constants or names of variables accessible from the current procedure. The second form causes the value returned by the procedure to be printed according to format *m*. If no format is given, it defaults to **d**. This facility is only available if the program was loaded with the –**g** option.

linenumber **b** *commands*
Set a breakpoint at the given line. If a procedure name without a line number is given (e.g., "proc:"), a breakpoint is placed at the first line in the procedure even if it was not compiled

with the **-g** option. If no *linenumber* is given, a breakpoint is placed at the current line. If no *commands* are given, execution stops just before the breakpoint and control is returned to *sdb*. Otherwise, the *commands* are executed when the breakpoint is encountered and execution continues. Multiple commands are specified by separating them with semicolons. If *k* is used as a command to execute at a breakpoint, control returns to *sdb*, instead of continuing execution.

B
Print a list of the currently active breakpoints.

linenumber **d**
Delete a breakpoint at the given line. If no *linenumber* is given then the breakpoints are deleted interactively. Each breakpoint location is printed and a line is read from the standard input. If the line begins with a **y** or **d** then the breakpoint is deleted.

D
Delete all breakpoints.

l
Print the last executed line.

linenumber **a**
Announce. If *linenumber* is of the form *proc:number*, the command effectively does a *linenumber* **b l**. If *linenumber* is of the form *proc:*, the command effectively does a *proc:* **b T**.

Miscellaneous commands:

!*command*
The command is interpreted by *sh*(1).

newline
If the previous command printed a source line, then advance the current line by one line and print the new current line. If the previous command displayed a memory location, then display the next memory location.

end-of-file character
Scroll. Print the next 10 lines of instructions, source or data depending on which was printed last. The end-of-file character is usually **CTRL-D**.

< *filename*
Read commands from *file name* until the end of file is reached, and then continue to accept commands from standard input. When *sdb* is told to display a variable by a command in such a file, the variable name is displayed along with the value. This command may not be nested; **<** may not appear as a command in a file.

M
Print the address maps.

M [**?**/] [*****] *b e f*
Record new values for the address map. The arguments **?** and **/** specify the text and data maps, respectively. The first segment (*b1, e1, f1*) is changed unless ***** is specified, in which case the second segment (*b2, e2, f2*) of the mapping is changed. If fewer than three values are given, the remaining map parameters are left unchanged.

" *string*
Print the given string. The C escape sequences of the form *\character* are recognized, where *character* is a nonnumeric character.

q
Exit the debugger.

h
Print help.

SDBPS1
The shell environment variable $SDBPS1 if set, will be used as the *sdb* user prompt in place of the default asterisk prompt.

The following commands also exist and are intended only for debugging the debugger:

V
Print the version number.

Q
Print a list of procedures and files being debugged.

FILES

a.out
core

SEE ALSO

cc(1), sh(1), a.out(4), core(4), syms(4).

WARNINGS

When *sdb* prints the value of an external variable for which there is no debugging information, a warning is printed before the value. The size is assumed to be **int** (integer).

Line number information in optimized functions is unreliable, the compiler always performs some minimal optimization which can re-order line numbers in the final object code.

If a procedure modifies any of its arguments, *sdb* may not be able to locate an unmodified copy to print. If it determines that no unmodified copy exists, it prints ??? in place of the value. It may not always be possible to determine that an existing value is unmodified, thus producing misleading stack back-traces.

BUGS

Current compiler optimization causes problems when printing local variables; the effects may be reduced with the Greenhills compiler by using the **–X159** option which disables some of the register optimization used by the Greenhills compiler. The *cc* front-end automatically passes the **–X159** option to the compiler when it detects the **–g** option on the command line.

Even when the **–X159** option is used, the debugger may still have trouble locating some automatic variables when tracing programs under *sdb*; care should be exercised when attempting to modify local (automatic) variables.

Stack back-tracing on the 88100 causes problems due to the lack of a frame pointer register. The stack back-trace code relies heavily on code examination to perform stack frame construction and register value location during code tracing. Certain code modules, especially those built from assembly code rather than ´C´ source can cause the back-trace to fail. This results in problems ranging from incorrect data symbols in disassembled code to total failure to provide a back-trace.

If a procedure is called when the program is *not* stopped at a breakpoint (such as when a core image is being debugged), all variables are initialized before the procedure is started. This makes it impossible to use a procedure which formats data from a core image.

NAME

sdiff – side-by-side difference program

SYNOPSIS

sdiff [*options* ...] *file1 file2*

DESCRIPTION

sdiff uses the output of *diff*(1) to produce a side-by-side listing of two files indicating those lines that are different. Each line of the two files is printed with a blank gutter between them if the lines are identical, a < in the gutter if the line only exists in *file1*, a > in the gutter if the line only exists in *file2*, and a | for lines that are different.

For example:

```
x       |       y
a               a
b       <
c       <
d               d
        >       c
```

The following options exist:

–w *n*
Use the next argument, *n*, as the width of the output line. The default line length is 130 characters.

–l
Only print the left side of any lines that are identical.

–s
Do not print identical lines.

–o *output*
Use the next argument, *output*, as the name of a third file that is created as a user-controlled merging of *file1* and *file2*. Identical lines of *file1* and *file2* are copied to *output*. Sets of differences, as produced by *diff*(1), are printed; where a set of differences share a common gutter character. After printing each set of differences, *sdiff* prompts the user with a % and waits for one of the following user-typed commands:

l	append the left column to the output file
r	append the right column to the output file
s	turn on silent mode; do not print identical lines
v	turn off silent mode
e l	call the editor with the left column
e r	call the editor with the right column
e b	call the editor with the concatenation of left and right
e	call the editor with a zero length file
q	exit from the program

On exit from the editor, the resulting file is concatenated on the end of the *output* file.

SEE ALSO

diff(1), ed(1)

NAME

sed – stream editor

SYNOPSIS

sed [**–n**] [**–e** *script*] [**–f** *sfilef*] [*ffiles*]

DESCRIPTION

sed copies the named *files* (standard input default) to the standard output, edited according to a script of commands. The **–f** option causes the script to be taken from file *sfile*; these options accumulate. If there is just one **–e** option and no **–f** options, the flag **–e** may be omitted. The **–n** option suppresses the default output. A script consists of editing commands, one per line, of the following form:

[address [, address]] function [arguments]

In normal operation, *sed* cyclically copies a line of input into a *pattern space* (unless there is something left after a **D** command), applies in sequence all commands whose *addresses* select that pattern space, and at the end of the script copies the pattern space to the standard output (except under **–n**) and deletes the pattern space.

Some of the commands use a *hold space* to save all or part of the *pattern space* for subsequent retrieval.

An *address* is either a decimal number that counts input lines cumulatively across files, a **$** that addresses the last line of input, or a context address, i.e., a */regular expression/* in the style of *ed*(1) modified thus:

In a context address, the construction *\?regular expression?*, where *?* is any character, is identical to */regular expression/*. Note that in the context address **\xabc\xdefx**, the second **x** stands for itself, so that the regular expression is **abcxdef**.

The escape sequence **\n** matches a newline *embedded* in the pattern space.

A period **.** matches any character except the *terminal* newline of the pattern space.

A command line with no addresses selects every pattern space.

A command line with one address selects each pattern space that matches the address.

A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line is selected.)Thereafter the process is repeated, looking again for the first address.

Editing commands can be applied only to non-selected pattern spaces by use of the negation function **!** (below).

In the following list of functions the maximum number of permissible addresses for each function is indicated in parentheses.

The *text* argument consists of one or more lines, all but the last of which end with **** to hide the newline. Backslashes in text are treated like backslashes in the replacement string of an **s** command, and may be used to protect initial blanks and tabs against the stripping that is done on every script line. The *rfile* or *wfile* argument must terminate the command line and must be preceded by exactly one blank. Each *wfile* is created before processing begins. There can be at most 10 distinct *wfile* arguments.

(1) a\
text Append. Place *text* on the output before reading the next input line.

(2) b *label* Branch to the : command bearing the *label*. If *label* is empty, branch to the end of the script.

(2) c\
text Change. Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, place *text* on the output. Start the next cycle.

(2) d Delete the pattern space. Start the next cycle.

(2) D Delete the initial segment of the pattern space through the first newline. Start the next cycle.

(2) g Replace the contents of the pattern space by the contents of the hold space.

(2) G Append the contents of the hold space to the pattern space.

(2) h Replace the contents of the hold space by the contents of the pattern space.

(2) H Append the contents of the pattern space to the hold space.

(1) i\
text Insert. Place *text* on the standard output.

(2) l List the pattern space on the standard output in an unambiguous form. Non-printable characters are displayed in octal notation and long lines are folded.

(2) n Copy the pattern space to the standard output. Replace the pattern space with the next line of input.

(2) N Append the next line of input to the pattern space with an embedded newline. (The current line number changes.)

(2) p Print. Copy the pattern space to the standard output.

(2) P Copy the initial segment of the pattern space through the first newline to the standard output.

(1) q Quit. Branch to the end of the script. Do not start a new cycle.

(2) r *rfile* Read the contents of *rfile*. Place them on the output before reading the next input line.

(2) s/*regular expression*/*replacement*/*flags*
Substitute the *replacement* string for instances of the *regular expression* in the pattern space. Any character may be used instead of /. For a fuller description see *ed*(1). *Flags* is zero or more of:

- n n= 1 - 512. Substitute for just the n th occurrence of the *regular expression*.
- g Global. Substitute for all nonoverlapping instances of the *regular expression* rather than just the first one.
- p Print the pattern space if a replacement was made.
- w *wfile* Write. Append the pattern space to *wfile* if a replacement was made.

(2) t *label* Test. Branch to the : command bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of a t. If *label* is empty, branch to the end of the script.

(2) w *wfile* Write. Append the pattern space to *wfile*.

(2) x Exchange the contents of the pattern and hold spaces.

(2) y/*string1*/*string2*/
Transform. Replace all occurrences of characters in *string1* with the corresponding character in *string2*. The lengths of *string1* and *string2* must be equal.

(2)! *function*
Don't. Apply the *function* (or group, if *function* is { } only to lines *not* selected by the address(es).

(0) : *label* This command does nothing; it bears a *label* for b and t commands to branch to.

(1) = Place the current line number on the standard output as a line.

(2) { Execute the following commands through a matching } only when the pattern space is selected.

(0) An empty command is ignored.

(0) # If a # appears as the first character on the first line of a script file, then that entire line is treated as a comment, with one exception. If the character after the # is an 'n', then the default output will be suppressed. The rest of the line after #n is also ignored. A script file must contain at least one non-comment line.

SEE ALSO

awk(1), ed(1), grep(1)

NAME

setpgrp – set process group id and execute command

SYNOPSIS

setpgrp *command* [*arg* ...]

DESCRIPTION

The *setpgrp* command sets its own process group id to its process id and then executes the command line that is the argument list.

SEE ALSO

getpgrp(2), setpgrp(2), getpid(2)

NAME

sh, rsh, ksh, rksh – shell, the standard/restricted command and programming language

SYNOPSIS

sh [**±aefhikmnoprstuvx**] [**±o** option] ... [**−c** string] [arg ...]
ksh [**±aefhikmnoprstuvx**] [**±o** option] ... [**−c** string] [arg ...]
rsh [**±aefhikmnoprstuvx**] [**±o** option] ... [**−c** string] [arg ...]
rksh [**±aefhikmnoprstuvx**] [**±o** option] ... [**−c** string] [arg ...]

DESCRIPTION

Sh (and *ksh*, the names are synonymous under REAL/IX) is a command and programming language that executes commands read from a terminal or a file. *Rsh* (and its synomyn *rksh*) is a restricted version of the standard command interpreter *sh*; it is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. See *Invocation* below for the meaning of arguments to the shell.

Definitions.

A *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 *variables*. 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 variable 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 newlines 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 *words*, 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 *words*, then the value of the variable *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 variable *identifier* is set to *null*. The contents of the line read from standard input is saved in the variable **REPLY**. The *list* is executed for each selection until a **break** or *end-of-file* is encountered. If the **REPLY** variable is set to *null* by the execution of *list*, then the selection list is printed before displaying the **PS3** prompt for the next selection.

case *word* **in** [[(]*pattern* [| *pattern*] ...) *list* ;;] ... **esac**

A **case** command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for file-name generation (see *File Name Generation* 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 occur 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. An **alias** name consists of any number of characters excluding metacharacters, quoting characters, file expansion characters, parameter and command substitution characters, and =. 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 tilde (~). 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 tilde and the matched login name are 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 tilde by itself, or in front of a /, is replaced by **$HOME**. A tilde followed by a + or – is replaced by **$PWD** and **$OLDPWD** respectively.

In addition, *tilde* substitution is attempted when the value of a *variable assignment* begins with a tilde (~).

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.

Process Substitution.

This feature is only available on versions of the UNIX operating system that support the **/dev/fd** directory for naming open files. Each command argument of the form **<** (*list*) or **>** (*list*) will run process *list* asynchronously connected to some file in **/dev/fd**. The name of this file will become the argument to the command. If the form with **>** is selected then writing on this file will provide input for *list*. If **<** is used, then the file passed as an argument will contain the output of the *list* process. For example,

paste < (**cut** –**f1** *file1*) **<** (**cut** –**f3** *file2*) **| tee >** (*process1*) **>** (*process2*)

cuts fields 1 and 3 from the files *file1* and *file2* respectively, *pastes* the results together, and sends it to the processes *process1* and *process2*, as well as putting it onto the standard output. Note that the file, which is passed as an argument to the command, is a UNIX *pipe*(2) so programs that expect to *lseek*(2) on the file will not work.

Parameter Substitution.

A *parameter* is an *identifier*, one or more digits, or any of the characters *, @, #, ?, –, $, and !. A *variable* (a parameter denoted by an identifier) has a *value* and zero or more *attributes*. *Variables* 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 variables pass values and attributes to the environment.

The shell supports a one-dimensional array facility. An element of an array variable 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 variable 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 *variable* may 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 variable is subscripted. If *parameter* is one or more digits then it is a positional parameter. A positional parameter of more than

SII(1)

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 of _ 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 variable 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 variable is set by the **select** statement and by the **read** special command when no arguments are supplied.

SECONDS
Each time this variable is referenced, the number of seconds since shell invocation is returned. If this variable 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 variables are used by the shell:

BCS (REAL/IX only) If set **=YES**, then some obscure differences between the behavior of **ksh** and **bsh** are resolved in favor of **bsh**. This option is required make the Korn shell pass the BCS and other validation suites.

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 variable 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** variable is used to separate arguments for the **"$*"** substitution (See *Quoting* below).

HISTFILE
If this variable 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 variable 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 variable is set to the name of a mail file *and* the **MAILPATH** variable 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** variables. 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 variable 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 variable **$_** defined as the name of the file that has changed. The default message is *you have mail in* **$_**.

PATH The search path for commands (see *Execution* below). The user may not change **PATH** if executing under *rsh* (except in **.profile**).

PS1 The value of this variable 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). Two successive occurrences of **!** will produce a single **!** when the prompt string is printed.

PS2 Secondary prompt string, by default "**>** ".

PS3 Selection prompt string used within a **select** loop, by default "**#?** ".

PS4 The value of this variable 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) will be turned on.

The shell gives default values to **PATH**, **PS1**, **PS2**, **PS3**, **PS4**, **MAILCHECK**, **FCEDIT**, **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.

File Name 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 from 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 removed. 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 occur 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***$2***d*...", where *d* is the first character of the **IFS** variable, whereas **"$@"** 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 then base 10 is used.

An arithmetic expression uses the same syntax, precedence, and associativity of expression as the C language. All the integral operators, other than ++, --, ?:, and , are supported. Variables can be referenced by name within an arithmetic expression without using the parameter substitution syntax. When a variable is referenced, its value is evaluated as an arithmetic expression.

An internal integer representation of a *variable* can be specified with the **-i** option of the **typeset** special command. Arithmetic evaluation is performed on the value of each assignment to a variable with the **-i** attribute. If you do not specify an arithmetic base, the first assignment to the variable 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 parameter expanded 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 is 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 occur 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 occur, **\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 variable 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 variables 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 variable assignments. A variable 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 except for special commands listed below that are preceded with a dagger).

If the **-k** flag is set, *all* variable 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**. Functions 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.

This paragraph and the next require features that are not in all versions of UNIX and may not apply. If you are running a job and wish to do something else you may hit the key **^Z** (control-Z) which sends a STOP signal to the current job. The shell will then normally indicate that the job

has been 'Stopped', and print another prompt. You can then manipulate the state of this job, putting it in the background with the **bg** command, or run some other commands and then eventually bring the job back into the foreground with the foreground command **fg**. A ^Z takes effect immediately and is like an interrupt in that pending output and unread input are discarded when it is typed.

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.

The 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 the **monitor** option is not active. Otherwise, signals have the values inherited by the shell from its parent (but see also the **trap** special 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 variable **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 variables 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 if the file it names 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 variable **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 newline ('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 backward 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 and 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.

% Moves to balancing (,), {, }, [, or]. If cursor is not on one of the above characters, the remainder of the line is searched for the first occurrence of one of the above characters first.

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. Variable assignment lists preceding the command remain in effect when the command completes.
2. I/O redirections are processed after variable 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 variable assignment, are expanded with the same rules as a variable 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, and no alias has been defined for the *name*.

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 variable **HOME** is the default *arg*. The variable **PWD** is set to the current directory. The shell variable **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 *rsh*.

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 *names* 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 –**l** flag 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 variable **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 be 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 *n* 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 variables 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 variable 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 argument can be one of the following option names:

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 as **−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 control mode. A return sends the line.
viraw Each character is processed as it is typed in *vi* mode.
xtrace Same as **−x**.
If no option name is supplied then the current option settings are printed.

−p Disables processing of the **$HOME/.profile** file and uses the file **/etc/suid_profile** instead of the **ENV** file. This mode is on whenever the effective uid (gid) is not equal to the real uid (gid). Turning this off causes the effective uid and gid to be set to the real uid and gid.

−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 variables are printed on the standard output.

† **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 the trap(s) for each *sig* are reset to their original values. If *arg* is the null string then this signal is ignored by the shell and by the commands it invokes. If *sig* is **ERR** then *arg* will be executed whenever a command has a non-zero exit status. If *sig* is **DEBUG** then *arg* will be 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 variables and functions. When invoked inside a function, a new instance of the variables *name* is created. The variables value and type are restored when the function completes. The following list of attributes may be specified:

- **−H** This flag provides UNIX to host-name file mapping on non-UNIX machines.
- **−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 variable is assigned to, 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 variable 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 variable 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 will be searched 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 upper-case characters are converted to lower-case. The upper-case flag, **−u** is turned off.
- **−r** The given *names* are marked readonly and these names cannot be changed by subsequent assignment.
- **−t** Tags the variables. Tags are user definable and have no special meaning to the shell.
- **−u** All lower-case characters are converted to upper-case characters. The lower-case flag, **−l** is turned off.
- **−x** The given *names* 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 *variables* which have these flags set is printed. (Using + rather than − keeps the values from being printed.) If no *names* and flags are given, the *names* and *attributes* of all *variables* are printed.

ulimit [**−HSacdfmnpstv**] [*limit*]

Set or display a resource limit. The available resources 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 specified. 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**

or **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 specified, then 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 plus 1.
- **–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.

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 *aliases* given by the list of *names* are removed from the *alias* list.

unset [**–f**] *name* ...
The variables given by the list of *names* are unassigned, i.e., their values and attributes are erased. Readonly variables cannot be unset. If the **–f**, flag is set, then 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 then 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.
The **–v** flag produces a more verbose report.
The **–p** flag 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 (**$0**) is –, then the shell is assumed to be a *login* shell and commands are read from **/etc/profile** and then from either **.profile** in the current directory or **$HOME/.profile**, if either file exists. Next, commands are read from the file named by performing parameter substitution on the value of the environment variable **ENV** if the file exists. If the **–s** flag is not present and *arg* is, then 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 will be 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 then commands are read from *string*.

-s If the -s flag is present or if no arguments remain then 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)) then 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.

rsh Only.

Rsh is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of *rsh* are identical to those of *sh*, except that the following are disallowed:

changing directory (see *cd*(1)),
setting the value of **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, *rsh* invokes *sh* to execute it. Thus, it is possible to provide to the end-user shell procedures that have access to the full power of the standard shell, while imposing a limited menu of commands; this scheme assumes that the end-user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the **.profile** 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 *rsh*.

EXIT STATUS

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 non-interactively then execution of the shell file is abandoned. Run time 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, then the line number is also printed in square brackets ([]) after the command or function name.

FILES

/etc/passwd
/etc/profile
/etc/suid_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.

CAVEATS

The Korn shell is the default shell under REAL/IX. Hence, it may also be invoked under the names **/bin/sh** and **/bin/rsh**. The Bourne shell, *bsh*(1), which is the traditional default shell for most UNIX-like operating systems, is still available as **/bin/bsh** and as **/bin/rbsh**. Because the Korn shell is the default, all references to the shell in other REAL/IX documentation should be considered as references to the Korn, rather than the Bourne, shell.

The Korn shell does have some incompatibilities with the Bourne shell. If these cause trouble, you may want to try setting **BCS=YES**. This variable resolves those incompatibilities preventing compliance with various test suites in favor of the Bourne shell.

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 will cause 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 foreground job terminates.

NAME

shl – shell layer manager

SYNOPSIS

shl

DESCRIPTION

shl allows a user to interact with more than one shell from a single terminal. The user controls these shells, known as *layers*, using the commands described below.

The *current layer* is the layer which can receive input from the keyboard. Other layers attempting to read from the keyboard are blocked. Output from multiple layers is multiplexed onto the terminal. To have the output of a layer blocked when it is not current, the *stty* option **loblk** may be set within the layer.

The *stty* character **swtch** (set to ^Z if NUL) is used to switch control to *shl* from a layer. *shl* has its own prompt, >>>, to help distinguish it from a layer.

A *layer* is a shell which has been bound to a virtual tty device (**/dev/sxt???**). The virtual device can be manipulated like a real tty device using *stty*(1) and *ioctl*(2). Each layer has its own process group id.

Definitions

A *name* is a sequence of characters delimited by a blank, tab or newline. Only the first eight characters are significant. The *names* **(1)** through **(7)** cannot be used when creating a layer. They are used by *shl* when no name is supplied. They may be abbreviated to just the digit.

Commands

The following commands may be issued from the *shl* prompt level. Any unique prefix is accepted.

create [*name*]
Create a layer called *name* and make it the current layer. If no argument is given, a layer will be created with a name of the form *(#)* where *#* is the last digit of the virtual device bound to the layer. The shell prompt variable PS1 is set to the name of the layer followed by a space. A maximum of seven layers can be created.

block *name* [*name* ...]
For each *name*, block the output of the corresponding layer when it is not the current layer. This is equivalent to setting the *stty* option **loblk** within the layer.

delete *name* [*name* ...]
For each *name*, delete the corresponding layer. All processes in the process group of the layer are sent the SIGHUP signal (see *signal*(2)).

help (or **?**)
Print the syntax of the *shl* commands.

layers [**–l**] [*name* ...]
For each *name*, list the layer name and its process group. The **–l** option produces a *ps*(1)-like listing. If no arguments are given, information is presented for all existing layers.

resume [*name*]
Make the layer referenced by *name* the current layer. If no argument is given, the last existing current layer will be resumed.

toggle
Resume the layer that was current before the last current layer.

unblock *name* [*name* ...]
For each *name*, do not block the output of the corresponding layer when it is not the current layer. This is equivalent to setting the *stty* option **–loblk** within the layer.

quit
Exit *shl*. All layers are sent the SIGHUP signal.

name
Make the layer referenced by *name* the current layer.

FILES

/dev/sxt???	Virtual tty devices
$SHELL	Variable containing pathname of the shell to use (default is /bin/sh).

SEE ALSO

sh(1), stty(1), ioctl(2), signal(2), sxt(7).

NAME

shmget – create shared memory segment

SYNOPSIS

/etc/shmget *path id size shmflg [physaddr]*

DESCRIPTION

shmget will create a shared memory segment if one does not already exist for the unique key generated by *ftok*(3C) using the values of *path* and *id*; see *ftok*(3C) for a description of *path* and *id*. *size* and *shmflg* specify the size and creation flags of the shared segment to be created; see *shmget*(2).

When the shared segment is physical memory then the optional *physaddr* argument must be used to specify the starting physical address of the segment. See *shmget*(2). If 'autosize' is specified instead of *size* then the size of physical memory will be automatically generated.

In the start-up script **/etc/rc2.d/S02SHMGET** a shared segment of physical memory is created when the system changes states from single to multi-user. The key (*path* and *id*) used in this script should be used by all other programs wanting to share physical memory.

EXAMPLES

A shared segment of physical memory may be obtained as follows:

```
/etc/shmget /dev/smem 0 autosize 0x6300 0
```

A 'C' program can obtain a reference id to the segment generated above by providing *shmget* system call with the unique key generated by *ftok*(3C) using the same *path* and *id* values as above.

```
id = shmget(ftok ("/dev/smem",0), 0, 0);
```

SEE ALSO

shmget(2), shmop(2), ftok(3C).

NAME

sifilter - preprocess MC88100 assembly language

SYNOPSIS

sifilter [*-switches*] [*input*] [*output*]

DESCRIPTION

sifilter manipulates MC88100 assembly language source code from *input* to work around known problems in early silicon. *sifilter* is normally invoked transparently by the shell script **/bin/as**, but can be used interactively for testing purposes. The program will no longer be needed when the final revision of the silicon is available.

input and *output* are normally omitted, defaulting to standard input and output paths. File names may be specified for either path, and a dash (-), denoting standard input, may be used as a place holder for *input*.

The translations performed by *sifilter* are controlled by the *switches* listed below. The shell script **/bin/cc** sets the "standard" option switches. In the released software all silicon filter options are disabled. If the user is generating object for old silicon, then sifilter switches can be enabled either on the command line, by using the **-F** option to the assembler, or by editing **/bin/cc**.

Switches

a
Insert a *trap-not-taken* (tb1 0,r0,511) after each **ld** or **ld.d.**

b
Split each **st.d** into an equivalent sequence of two **st** instructions.

c
Do not pass comment lines through to the output.

d
Issue each **st** or **st.d** twice.

e
Enable literal synthesis. See **Literal Synthesis.** (**NOTE**: This option is not compatible with some of the other options available, its use is no longer supported.)

f Signals that input comes from the Absoft FORTRAN compiler.

g
Signals that input comes from the Greenhills C compiler.

See the **Scratch Registers** section for an explanation of the need for the **f** and **g** switches.

j Fix a problem with certain instances of the immediate forms of *div* and *divu*.

k
Insert a *no-op* (or r0,r0,r0) after each *trap-not-taken* inserted by the **a** option. If the **a** option has not been specified, this option has no effect.

l Split each **ld.d** into an equivalent sequence of two **ld** instructions.

n
Insert a *no-op* after each *trap-not-taken* inserted by the **p** option, but only if the store instruction involved uses scaling. Specifying the **n** option automatically forces the **p** option.

p
Insert a *trap-not-taken* before each **st** or **st.d.**

q
Insert a dummy **ld** before each **ld.** A dummy load is a load in which the destination register is r0. The source operands in a dummy load are the same as those in the actual load which

follows.

r
Fix various problems with the *def* and *sdef* directives produced by the Greenhills C compiler. This option has no effect with the Motorola C or Absoft FORTRAN compilers.

s
Produce a statistics dump on the standard error path on termination.

t
Compensate for bugs in floating-point operations which have a double-precision destination. See **WARNINGS**.

u
Adds a *no-op* after an **lda.d** if it is immediately followed by a **ld.bu.**

V
Displays a version identification message and exits immediately.

v
A single **v** enables "verbose" mode, in which various messages detailing actions taken by *sifilter* are output as comment lines. Two or more instances of **v** in the option string generates a comment line containing the current location counter value before each source line.

w
Compensate for a bug in the *fmul* instruction in rev D.5 MC88100 chips.

x
Compensate for a bug in the *fmul* instruction by breaking certain immediate integer divides and multiplies into register-to-register forms. (This fix only works in conjunction with a kernel exception code fix.)

y
Insert a *no-op* after each *trap-not-taken* generated by the **z** option. If **z** has not been specified, this option has no effect.

z
Insert a trap-not-taken after each **st** or **st.d.**

E
Insert a **tcnd eq0** before all divides. This compensates for a bug in all mask revisions up to and including E.2, where divide by zero does not always trigger a trap.

Defaults

All switches default to "off". If neither of the **g** or **f** switches are specified, the input is assumed to have been generated by the Motorola C compiler.

sifilter performs the following transformations regardless of the option switches specified.

addu and **subu** instructions with operands r31,r31,lit32 where "lit32" is a constant whose value is greater than 64K are replaced with an equivalent sequence.

Floating point instructions involving double operands may be moved if they would otherwise fall at the end of a cache line.

Literal Synthesis

Since *sifilter* must maintain an accurate location counter, it must perform the same fixups for "lit16" operands that would normally be done by *ld.*

Instructions with lit16 operands whose value cannot be determined by *sifilter* (for example, a label), or whose value would require more than 16 bits, are replaced with an equivalent sequence. This is called "literal synthesis", since a 32-bit value is "synthesized" in a register from the literal.

There are two forms of literal synthesis. The short form:

```
or.u  r29,r0,hi16(lit16)
op    rd,r29,lo16(lit16)
```

is used for the **add, addu, ld, lda, or, st, xmem,** and **xor** instructions (in all their variations) when the source register is r0. When the source register is other than r0, these instructions are expanded into the long form:

```
or.u  r29,r0,hi16(lit16)
or    r29,r29,lo16(lit16)
op    rd,rs,r29
```

Instructions which always are expanded with the long form are all the variations on **and, cmp, div, divu, mask, mul, sub, subu,** and **tbnd.**

No literal synthesis is done unless the e option has been specified.

Scratch Registers

Some of the fixups performed by *sifilter* require one or two scratch registers (split **ld.d** or **st.d, addu, subu,** and floats).

Fixups which require only one scratch register will always succeed, since all of the supported compilers have at least one available. Fixups requiring two scratch registers will succeed with the Motorola or Greenhills C compilers, but will fail with the Absoft FORTRAN compiler, which has only one available register.

SEE ALSO

/bin/as, /bin/cc

WARNINGS

Code generated when the **-t** switch is in effect requires special kernel support. DO NOT USE the **-t** option unless your operating system kernel is dated 880815 or later. (To examine the kernel date, enter the command **uname -a.** The date will be the fourth field in the output.)

The sifilter cannot deal with labels in branch delay slots if the branch delay slot instruction requires fixup. In these cases the message:

FATAL: delay branch instruction has label

is written to **stderr** and *sifilter* will exit. This is usually only a problem with hand-optimized assembly source files, which should not require filtering in any case.

BUGS

Given the large number of option switches, and the almost infinite number of instruction combinations, bugs would not be surprising.

NAME

sink – canonical "server" process for testing network.

SYNOPSIS

sink

DESCRIPTION

sink spawns a daemon which listens on the port used by the TCP sink service (see *services*(4)). As connections are made to that port, *sink* reads and discards any incoming data for the duration of the connection. Multiple and simultaneous connections are supported.

FILES

/etc/services determines port number

NAME

size – print section sizes in bytes of common object files

SYNOPSIS

size [**–n**] [**–f**] *mail-command*[**–d**] *mail-command*[**–o**] [**–x**] [**–V**] *files*

DESCRIPTION

The *size* command produces section size information in bytes for each loaded section in the common object files. The size of the text, data, and bss (uninitialized data) sections is printed, as well as the sum of the sizes of these sections. If an archive file is input to the *size* command, the information for all archive members is displayed.

The **–n** option includes NOLOAD sections in the size.

The **–f** option produces full output, i.e., it prints the size of every loaded section, followed by the section name in parentheses.

Numbers are printed in decimal unless either the **–o** or the **–x** option is used, in which case they are printed in octal or hexadecimal, respectively. *mail-command*If both options are used, the one appearing last on the command line is applied. The **–d** option (decimal output) is the default.
mail-command.PP The **–V** flag supplies the version information on the *size* command.

SEE ALSO

as(1), cc(1), ld(1), a.out(4), ar(4)

CAVEAT

Since the size of bss sections is not known until link-edit time, the *size* command does not give the true total size of pre-linked objects.

DIAGNOSTICS

size: name: cannot open
if *name* cannot be read.

size: name: bad magic
if *name* is not an appropriate common object file.

*mail-command*If an unknown option is given to *size*, it prints a message that it is ignoring the invalid option and continues. The exit code in this case is zero, unless other errors are encountered.
mail-command.

NAME

sleep – suspend execution for an interval

SYNOPSIS

sleep *time*

DESCRIPTION

sleep suspends execution for *time* seconds. It is used to execute a command after a certain amount of time, as in:

(sleep 105; *command*)&

or to execute a command every so often, as in:

```
while true
do
        command
        sleep 37
done
```

SEE ALSO

alarm(2), sleep(3C).

NAME

sort – sort and/or merge files

SYNOPSIS

sort [**–cmu**] [**–o***output*] [**–y***kmem*] [**–z***recsz*] [**–dfiMnr**] [**–bt***x*]
[**+***pos1* [**–***pos2*]] [*files*]

DESCRIPTION

sort sorts lines of all the named files together and writes the result on the standard output. The standard input is read if – is used as a file name or no input files are named.

Comparisons are based on one or more sort keys extracted from each line of input. By default, there is one sort key, the entire input line, and ordering is lexicographic by bytes in machine collating sequence.

The following options alter the default behavior:

–c
Check that the input file is sorted according to the ordering rules; give no output unless the file is out of sort.

–m
Merge only, the input files are already sorted.

–u
Unique: suppress all but one in each set of lines having equal keys.

–o*output*
The argument given is the name of an output file to use instead of the standard output. This file may be the same as one of the inputs. There may be optional blanks between **–o** and *output*.

–y*kmem*
The amount of main memory used by the sort has a large impact on its performance. Sorting a small file in a large amount of memory is a waste. If this option is omitted, *sort* begins using a system default memory size, and continues to use more space as needed. If this option is presented with a value, *kmem*, *sort* will start using that number of kilobytes of memory, unless the administrative minimum or maximum is violated, in which case the corresponding extremum will be used. Thus, **–y**0 is guaranteed to start with minimum memory. By convention, **–y** (with no argument) starts with maximum memory.

–z*recsz*
The size of the longest line read is recorded in the sort phase so buffers can be allocated during the merge phase. If the sort phase is omitted via the **–c** or **–m** options, a popular system default size will be used. Lines longer than the buffer size will cause *sort* to terminate abnormally. Supplying the actual number of bytes in the longest line to be merged (or some larger value) will prevent abnormal termination.

The following options override the default ordering rules.

–d
"Dictionary" order: only letters, digits, and blanks (spaces and tabs) are significant in comparisons.

–f
Fold lowercase letters into uppercase.

–i
Ignore non-printable characters.

−M

Compare as months. The first three non-blank characters of the field are folded to uppercase and compared. For example, in English the sorting order is "JAN" < "FEB" < ... < "DEC". Invalid fields compare low to "JAN". The **−M** option implies the **−b** option (see below).

−n

An initial numeric string, consisting of optional blanks, optional minus sign, and zero or more digits with optional decimal point, is sorted by arithmetic value. The **−n** option implies the **−b** option (see below). Note that the **−b** option is only effective when restricted sort key specifications are in effect.

−r

Reverse the sense of comparisons.

When ordering options appear before restricted sort key specifications, the requested ordering rules are applied globally to all sort keys. When attached to a specific sort key (described below), the specified ordering options override all global ordering options for that key.

The notation *+pos1 −pos2* restricts a sort key to one beginning at *pos1* and ending just before *pos2*. The characters at position *pos1* and just before *pos2* are included in the sort key (provided that *pos2* does not precede *pos1*). A missing *−pos2* means the end of the line.

Specifying *pos1* and *pos2* involves the notion of a field, a minimal sequence of characters followed by a field separator or a newline. By default, the first blank (space or tab) of a sequence of blanks acts as the field separator. All blanks in a sequence of blanks are considered to be part of the next field; for example, all blanks at the beginning of a line are considered to be part of the first field. The treatment of field separators can be altered using the options:

−b

Ignore leading blanks when determining the starting and ending positions of a restricted sort key. If the **−b** option is specified before the first *+pos1* argument, it will be applied to all *+pos1* arguments. Otherwise, the **b** flag may be attached independently to each *+pos1* or *−pos2* argument (see below).

−t*x*

Use *x* as the field separator character; *x* is not considered to be part of a field (although it may be included in a sort key). Each occurrence of *x* is significant (for example, *xx* delimits an empty field).

pos1 and *pos2* each have the form *m.n* optionally followed by one or more of the flags **bdfinr**. A starting position specified by *+m.n* is interpreted to mean the *n*+1st character in the *m*+1st field. A missing *.n* means .0, indicating the first character of the *m*+1st field. If the **b** flag is in effect *n* is counted from the first non-blank in the *m*+1st field; *+m.*0**b** refers to the first non-blank character in the *m*+1st field.

A last position specified by *−m.n* is interpreted to mean the *n*th character (including separators) after the last character of the *m th* field. A missing *.n* means .0, indicating the last character of the *m*th field. If the **b** flag is in effect *n* is counted from the last leading blank in the *m*+1st field; *−m.*1**b** refers to the first non-blank in the *m*+1st field.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant.

EXAMPLES

Sort the contents of *infile* with the second field as the sort key:

sort +1 −2 infile

Sort, in reverse order, the contents of *infile1* and *infile2*, placing the output in *outfile* and using the first character of the second field as the sort key:

sort −r −o outfile +1.0 −1.2 infile1 infile2

Sort, in reverse order, the contents of *infile1* and *infile2* using the first non-blank character of the second field as the sort key:

sort −r +1.0b −1.1b infile1 infile2

Print the password file (*passwd*(4)) sorted by the numeric user ID (the third colon-separated field):

sort −t: +2n −3 /etc/passwd

Print the lines of the already sorted file *infile*, suppressing all but the first occurrence of lines having the same third field (the options **−um** with just one input file make the choice of a unique representative from a set of equal lines predictable):

sort −um +2 −3 infile

FILES

/usr/tmp/stm???

SEE ALSO

comm(1), join(1), uniq(1)

WARNINGS

Comments and exits with non-zero status for various trouble conditions (for example, when input lines are too long), and for disorder discovered under the −c option. When the last line of an input file is missing a **newline** character, *sort* appends one, prints a warning message, and continues.

sort does not guarantee preservation of relative line ordering on equal keys.

NAME

spell, hashmake, spellin, hashcheck – find spelling errors

SYNOPSIS

spell [**–v**] [**–b**] [**–x**] [**–l**] [**+***local_file*] [*files*]

/usr/lib/spell/hashmake

/usr/lib/spell/spellin n

/usr/lib/spell/hashcheck *spelling_list*

DESCRIPTION

spell collects words from the named *files* and looks them up in a spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes, and/or suffixes) from words in the spelling list are printed on the standard output. If no *files* are named, words are collected from the standard input.

spell ignores most *troff*(1), *tbl*(1), and *eqn*(1) constructions.

Under the **–v** option, all words not literally in the spelling list are printed, and plausible derivations from the words in the spelling list are indicated.

Under the **–b** option, British spelling is checked. Besides preferring *centre*, *colour*, *programme*, *speciality*, *travelled*, this option insists upon *-ise* in words like *standardise*, Fowler and the OED to the contrary notwithstanding.

Under the **–x** option, every plausible stem is printed with = for each word.

By default, *spell* (like *deroff*(1)) follows chains of included files (**.so** and **.nx** *troff*(1) requests), *unless* the names of such included files begin with **/usr/lib**. Under the **–l** option, *spell* will follow the chains of *all* included files.

Under the **+***local_file* option, words found in *local_file* are removed from *spell*'s output. *local_file* is the name of a user-provided file that contains a sorted list of words, one per line. With this option, the user can specify a set of words that are correct spellings (in addition to *spell*'s own spelling list) for each job.

The spelling list is based on many sources, and while more haphazard than an ordinary dictionary, is also more effective with respect to proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine, and chemistry is light.

Pertinent auxiliary files may be specified by name arguments, indicated below with their default settings (see **FILES**). Copies of all output are accumulated in the history file. The stop list filters out misspellings (e.g., thier=thy–y+ier) that would otherwise pass.

Three routines help maintain and check the hash lists used by *spell*:

hashmake
Reads a list of words from the standard input and writes the corresponding nine-digit hash code on the standard output.

spellin
Reads *n* hash codes from the standard input and writes a compressed spelling list on the standard output.

hashcheck
Reads a compressed *spelling_list* and recreates the nine-digit hash codes for all the words in it; it writes these codes on the standard output.

FILES

D_SPELL=/usr/lib/spell/hlist[ab]	hashed spelling lists, American & British
S_SPELL=/usr/lib/spell/hstop	hashed stop list
H_SPELL=/usr/lib/spell/spellhist	history file
/usr/lib/spell/spellprog	program

SEE ALSO

deroff(1), sed(1), sort(1), tee(1)

BUGS

The spelling list's coverage is uneven; new installations will probably wish to monitor the output for several months to gather local additions; typically, these are kept in a separate local file that is added to the hashed *spelling_list* via *spellin*.

With continued use, the history file will grow unchecked.

NAME

split – split a file into pieces

SYNOPSIS

split [*–n*] [*file* [*name*]]

DESCRIPTION

split reads *file* and writes it in *n*-line pieces (default 1000 lines) onto a set of output files. The name of the first output file is *name* with **aa** appended, and so on lexicographically, up to **zz** (a maximum of 676 files). *Name* cannot be longer than 12 characters. If no output name is given, **x** is default.

If no input file is given, or if – is given in its stead, then the standard input file is used.

SEE ALSO

bfs(1), csplit(1)

NAME

starter – information about the system for beginning users

SYNOPSIS

[**help**] **starter**

DESCRIPTION

The Help Facility command *starter* provides five categories of information about the system to assist new users.

The five categories are:

- commands a new user should learn first
- documents important for beginners
- education centers offering courses
- local environment information
- online teaching aids

The user may choose one of the above categories by entering its corresponding letter (given in the menu), or may exit to the shell by typing q (for "quit"). When a category is chosen, the user will receive one or more pages of information pertaining to it.

From any screen in the Help Facility, a user may execute a command via the shell (*sh*(1)) by typing a ! and the command to be executed. The screen will be redrawn if the command that was executed 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

glossary(1), help(1), locate(1), sh(1), usage(1), term(5).

WARNINGS

If the shell variable TERM (see *sh*(1)) 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

strings – display character strings in object files

SYNOPSIS

strings *filename ...*

DESCRIPTION

The *strings* command goes through the .data section(s) of a SYSTEM V/88 object file, searching for strings of one or more printable characters. All strings found are printed to standard output.

DIAGNOSTICS

filename: not object file

This error message indicates that *filename* is not an object file, based on the magic number found in the file header.

filename can't open

This error message indicates that *filename* cannot be opened by the strings program. This occurs if the file does not exist or if it is not readable by the user who is running *strings*.

NOTES

If a character string is found which is more than 140 characters long, it is not displayed.

NAME

strip – strip symbol and line number information from a common object file

SYNOPSIS

strip [**–l**] [**–x**] [**–b**] [**–r**] [**–u** [**–v**]] [**–V**] [**–S**] *filename ...*

DESCRIPTION

The *strip* command strips the symbol table and line number information from common object files, including archives. Once this has been done, no symbolic debugging access will be available for that file; therefore, this command is normally run only on production modules that have been debugged and tested.

The amount of information stripped from the symbol table can be controlled by using any of the following options:

–l
Strip line number information only; do not strip any symbol table information.

–x
Do not strip static or external symbol information.

–b
Same as the **–x** option, but also do not strip scoping information (e.g., beginning and end of block delimiters).

–r
Do not strip static or external symbol information, or relocation information.

–u
Strip out symbols that are unused, in the sense that they are not referenced by any line number entries, relocation entries or other symbol table entries, and are not one of the *special* symbols such as **.file**, **.bf**, etc. If the **–u** option is turned on, it forces **–b**, **–l**, **–r**, and **–x** to be off. If the **–v** is also used with the **–u**, statistics about the symbol table will be printed. The **–v** option is only valid with **–u**.

–S
Tells *strip* to identify local static user symbols that begin with an at sign (**@L** or **@_**), as *referenced.*

–V
Print the version of the strip command executing on the standard error output.

If there are any relocation entries in the object file and any symbol table information is to be stripped, *strip* will complain and terminate without stripping *filename* unless the **–r** option is used.

If the *strip* command is executed on a common archive file (see *ar*(4)) the archive symbol table will be removed. The archive symbol table must be restored by executing the *ar*(1) command with the **s** option before the archive can be link-edited by the *ld*(1) command. *strip* will produce appropriate warning messages when this situation arises.

strip is used to reduce the file storage overhead taken by the object file.

FILES

TMPDIR/strp* temporary files

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

SEE ALSO

ar(1), as(1), cc(1), ld(1), tmpnam(3S), a.out(4), ar(4).

DIAGNOSTICS

strip: name: cannot open
if *name* cannot be read.

strip: name: bad magic
if *name* is not an appropriate common object file.

strip: name: relocation entries present; cannot strip
if *name* contains relocation entries and the **-r** flag is not used, the symbol table information cannot be stripped.

NAME

stty – set the options for a terminal

SYNOPSIS

stty [**–a**] [**–g**] [*options*]

DESCRIPTION

stty sets certain terminal I/O options for the device that is the current standard input; without arguments, it reports the settings of certain options.

In this report, if a character is preceded by a caret (^), then the value of that option is the corresponding **CTRL** character (e.g., "**^h**" is **CTRL-h**; in this case, recall that **CTRL-h** is normally the same as the **BACKSPACE** key.) The sequence "**^ ^**" means that an option has a **NULL** value. For example, normally *stty* **–a** will report that the value of **swtch** is "**^ ^**"; however, if *shl (1)* has been invoked, **swtch** will have the value "**^z**".

–a
reports all the option settings.

–g
reports current settings in a form that can be used as an argument to another *stty* command.

Options in the last group are implemented using options in the previous groups. Note that many combinations of options make no sense, but no sanity checking is performed. The options are selected from the following:

Control Modes

parenb (–parenb) enable (disable) parity generation and detection.

parodd (–parodd) select odd (even) parity.

cs5 cs6 cs7 cs8 select character size (see *termio*(7)).

0 hang up phone line immediately.

110 300 600 1200 1800 2400 4800 9600 19200 38400
Set terminal baud rate to the number given, if possible. (Not all speeds are supported by all hardware interfaces.)

hupcl (–hupcl) hang up (do not hang up) Dataphone connection on last close.

hup (–hup) same as **hupcl (–hupcl)**.

cstopb (–cstopb) use two (one) stop bits per character.

cread (–cread) enable (disable) the receiver.

clocal (–clocal) assume a line without (with) modem control.

loblk (–loblk) block (do not block) output from a non-current layer.

Input Modes

ignbrk (–ignbrk) ignore (do not ignore) break on input.

brkint (–brkint) signal (do not signal) INTR on break.

ignpar (–ignpar) ignore (do not ignore) parity errors.

parmrk (–parmrk) mark (do not mark) parity errors (see *termio*(7)).

inpck (–inpck) enable (disable) input parity checking.

istrip (–istrip) strip (do not strip) input characters to seven bits.

inlcr (–inlcr) map (do not map) NL to CR on input.

igncr (–igncr) ignore (do not ignore) CR on input.

icrnl (–icrnl) map (do not map) CR to NL on input.

iuclc (–iuclc) map (do not map) upper-case alphabetics to lower case on input.

ixon (–ixon) enable (disable) START/STOP output control. Output is stopped by sending an ASCII DC3 and started by sending an ASCII DC1.

ixany (–ixany) allow any character (only DC1) to restart output.

ixoff (–ixoff) request that the system send (not send) START/STOP characters when the input queue is nearly empty/full.

Output Modes

opost (–opost) post-process output (do not post-process output; ignore all other output modes).

olcuc (–olcuc) map (do not map) lowercase alphabetics to uppercase on output.

onlcr (–onlcr) map (do not map) NL to CR-NL on output.

ocrnl (–ocrnl) map (do not map) CR to NL on output.

onocr (–onocr) do not (do) output CRs at column zero.

onlret (–onlret) on the terminal NL performs (does not perform) the CR function.

ofill (–ofill) use fill characters (use timing) for delays.

ofdel (–ofdel) fill characters are DELs (NULs).

cr0 cr1 cr2 cr3 select style of delay for carriage returns (see *termio*(7)).

nl0 nl1 select style of delay for line-feeds (see *termio*(7)).

tab0 tab1 tab2 tab3 select style of delay for horizontal tabs (see *termio*(7)).

bs0 bs1 select style of delay for backspaces (see *termio*(7)).

ff0 ff1 select style of delay for form-feeds (see *termio*(7)).

vt0 vt1 select style of delay for vertical tabs (see *termio*(7)).

Local Modes

isig (–isig) enable (disable) the checking of characters against the special control characters INTR, QUIT, and SWTCH.

icanon (–icanon) enable (disable) canonical input (ERASE and KILL processing).

xcase (–xcase) canonical (unprocessed) upper/lowercase presentation.

echo (–echo) echo back (do not echo back) every character typed.

echoe (–echoe) echo (do not echo) ERASE character as a backspace-space-backspace string. Note that this mode will erase the ERASEed character on many CRT terminals; however, it does *not* keep track of column position and, as a result, may be confusing on escaped characters, tabs, and backspaces.

echok (–echok) echo (do not echo) NL after KILL character.

lfkc (–lfkc) the same as **echok (–echok)**; obsolete.

echonl (–echonl) echo (do not echo) NL.

noflsh (–noflsh) disable (enable) flush after INTR, QUIT, or SWTCH.

stwrap (–stwrap) disable (enable) truncation of lines longer than 79 characters on a synchronous line.

stflush (–stflush) enable (disable) flush on a synchronous line after every *write*(2).

stappl (–stappl) use application mode (use line mode) on a synchronous line.

Control Assignments

control-character c set *control-character* to *c*, where *control-character* is **erase**, **kill**, **intr**, **quit**, **swtch**, **start**, **stop**, **eof**, **eol**, **ctab**, **min**, or **time** (**ctab** is used with **–stappl**; **min** and **time** are used with **–icanon**; see *termio*(7)). If *c* is preceded by a caret (^) (escaped from the shell), then the value used is the corresponding **CTRL** character (e.g., "^**d**" is a **CTRL-d**); "^**?**" is interpreted as **DEL** and "^–" is interpreted as undefined.

line *i* set line discipline to *i* (0 < *i* < 127).

Combination Modes

evenp or **parity** enable **parenb** and **cs7**.

oddp enable **parenb**, **cs7**, and **parodd**.

–parity, **–evenp**, or **–oddp**
disable **parenb**, and set **cs8**.

raw (**–raw** or **cooked**)
enable (disable) raw input and output (no ERASE, KILL, INTR, QUIT, SWTCH, EOT, or output post processing).

nl (**–nl**) unset (set) **icrnl**, **onlcr**. In addition **–nl** unsets **inlcr**, **igncr**, **ocrnl**, and **onlret**.

lcase (**–lcase**) set (unset) **xcase**, **iuclc**, and **olcuc**.

LCASE (**–LCASE**) same as **lcase** (**–lcase**).

tabs (**–tabs** or **tab3**) preserve (expand to spaces) tabs when printing.

ek reset ERASE and KILL characters back to normal # and @.

sane resets all modes to some reasonable values.

term set all modes suitable for the terminal type *term*, where *term* is one of **tty33**, **tty37**, **vt05**, **tn300**, **ti700**, or **tek**.

SEE ALSO

tabs(1), ioctl(2), termio(7).

NAME

sum – print checksum and block count of a file

SYNOPSIS

sum [**–r**] *file*

DESCRIPTION

sum calculates and prints a 16-bit checksum for the named file, and also prints the number of blocks in the file. It is typically used to look for bad spots, or to validate a file communicated over some transmission line. The option **–r** causes an alternate algorithm to be used in computing the checksum.

SEE ALSO

wc(1).

DIAGNOSTICS

"Read error" is indistinguishable from end of file on most devices; check the block count.

NAME

systat – display system statistics on a crt

SYNOPSIS

systat [*–display*] [refresh-interval]

DESCRIPTION

systat displays various system statistics in a screen oriented fashion using the curses screen display library, *curses*(3X).

Input is interpreted at two different levels. A "global" command interpreter processes all keyboard input. If this command interpreter fails to recognize a command, the input line is passed to a per-display command interpreter. This allows each display to have certain display-specific commands.

Certain characters cause immediate action by *systat*. These are

^L Refresh the screen.

^R Refresh the screen.

^G Print the name of the current "display" being shown in the lower window and the refresh interval.

: Move the cursor to the command line and interpret the input line typed as a command. While entering a command, the current character erase and line kill characters may be used. Ctrl-W may be used to erase a word.

The following commands are interpreted by the "global" command interpreter. Precede each command with a colon (:).

help

Display help for the current screen and print the names of the available displays on the command line.

load

Print the load average over the past 1, 5, and 15 minutes on the command line.

ver

Print the program revision and creation date.

pri

The priority may be set at the specified priority (if less than 128). If the priority given is 128 or greater, then the priority is initially set to this value, but normal timesharing priority assignment takes place thereafter. If no priority is given, then toggle the priority between realtime priority 127 and time-sharing priorities.

stop

Stop refreshing the lower part of the screen.

[start] [number]

Start (continue) refreshing the screen. If a second, numeric, argument is provided it is interpreted as a refresh interval (in seconds). Supplying only a number will set the refresh interval to this value.

[interval] [number]

Start (continue) refreshing the screen. If a second, numeric, argument is provided it is interpreted as a refresh interval (in seconds). Supplying only a number will set the refresh interval to this value.

quit

Exit *systat*. (This may be abbreviated to *q*.)

The top part of the screen shows the following: Percentages of cpu usage are displayed for processes in user mode, kernel mode, waiting on I/O, waiting on memory, and idling. The number of system calls, context switches, and preemptions per second are displayed.

Memory information displayed consists of the maximum amount of free memory, current free memory, available resident memory (not swappable), available swappable memory, and current free memory to resident processes. Each amount is in 8K 4K pages. When the vhand process (page stealer) is awakened, an asterisk is displayed to the left of the current free memory label.

At the top of the display is the system name, node name, release, version, and machine name. The current time is also displayed.

The user and kernel cpu utilization is displayed as a horizontal bar graph. The kernel utilization is displayed as 'k' and the user utilization is displayed as 'u'. After 1 minute, an average of the user and kernel cpu utilization is displayed for the past minute. After 5 minutes, another average is displayed for the past 5 minutes. After 15 minutes, another average is displayed for the past 15 minutes. Once displayed, these averages are continually updated each second.

The lower part of the system display shows system performance information. Processes are displayed in order of priority. The default display consists of only those processes on the run queue or having some cpu utilization. Each process displayed consists of four fields.

The first field is a single character indicating the state of the process:

- d process is awaiting an event, ignoring signals (often a daemon)
- i process in intermediate state of creation
- o process is on the cpu (currently executing)
- r process is ready to run
- s process is sleeping (awaiting an event or signal)
- t process stopped since it is being traced
- x process is waiting for more memory
- z zombie process

The second field is the priority. The third field is the cpu usage of the process. The fourth field is the name of the process.

The following commands are specific to the *system* display; the minimum unambiguous prefix may be supplied. Precede each command with a colon (:).

all
: This command causes all processes to be listed. ">>>>>" is displayed in the lower right-hand corner when there are more processes than room to print on the screen. The arrows are informational; there is no way to page through to see all the processes.

noall
: This command restores the display to the default: processes on the run queue or having cpu usage are displayed.

exclude
: This command excludes a process having a specified priority from the processes displayed. If no priority is specified, the priority 0 is used.

include
: This command includes a process having a specified priority to the processes displayed. If no priority is specified, the priority 0 is used.

user
: This command adds user names to the display per process. The number of columns displayed is reduced to accommodate the longer line per process.

nouser
: This command prevents user names from being displayed per process.

FILES

/realix	for the namelist
/dev/kmem	for information in main memory
/etc/ps_data	for user information

BUGS

Takes 2-10 percent of the cpu. Certain displays presume a 24 line by 80 character terminal.

WARNINGS

May conflict with lower priority processes doing terminal I/O such as window applications (xterm).

NAME

tabs - set tabs on a terminal

SYNOPSIS

tabs [*tabspec*] [**-T***type*] [**+m***n*]

DESCRIPTION

tabs sets the tab stops on the user's terminal according to the tab specification *tabspec*, after clearing any previous settings. The user's terminal must have remotely-settable hardware tabs.

tabspec

Four types of tab specification are accepted for *tabspec*. They are described below: canned (*-code*), repetitive (*-n*), arbitrary (*n1,n2,...*), and file (*—file*). If no *tabspec* is given, the default value is **-8**, i.e., "standard" tabs. The lowest column number is 1. Note that for *tabs*, column 1 always refers to the leftmost column on a terminal, even one whose column markers begin at 0, e.g., the DASI 300, DASI 300s, and DASI 450.

-code

Use one of the codes listed below to select a *canned* set of tabs. The legal codes and their meanings are:

-a
1,10,16,36,72
Assembler, IBM S/370, first format

-a2
1,10,16,40,72
Assembler, IBM S/370, second format

-c
1,8,12,16,20,55
COBOL, normal format

-c2
1,6,10,14,49
COBOL compact format (columns 1-6 omitted). Using this code, the first typed character corresponds to card column 7, one space gets you to column 8, and a tab reaches column 12. Files using this tab setup should include a format specification as follows (see *fspec*(4)):

< :t-c2 m6 s66 d:>

-c3 1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67
COBOL compact format (columns 1-6 omitted), with more tabs than **-c2**. This is the recommended format for COBOL. The appropriate format specification is (see *fspec*(4)):

< :t-c3 m6 s66 d:>

-f 1,7,11,15,19,23
FORTRAN

-p 1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61
PL/I

-s 1,10,55
SNOBOL

-u 1,12,20,44
UNIVAC 1100 Assembler

−n

A *repetitive* specification requests tabs at columns 1+*n*, 1+2*n*, etc. Of particular importance is the value **8**: this represents the "standard" tab setting, and is the most likely tab setting to be found at a terminal. Another special case is the value **0**, implying no tabs at all.

n1,n2,...

The *arbitrary* format permits the user to type any chosen set of numbers, separated by commas, in ascending order. Up to 40 numbers are allowed. If any number (except the first one) is preceded by a plus sign, it is taken as an increment to be added to the previous value. Thus, the formats **1,10,20,30**, and **1,10,+10,+10** are considered identical.

—file

If the name of a *file* is given, *tabs* reads the first line of the file, searching for a format specification (see *fspec*(4)). If it finds one there, it sets the tab stops according to it, otherwise it sets them as **−8**. This type of specification may be used to make sure that a tabbed file is printed with correct tab settings, and would be used with the *pr*(1) command:

tabs — file; **pr** file

Any of the following also may be used; if a given flag occurs more than once, the last value given takes effect:

−T*type*

tabs usually needs to know the type of terminal in order to set tabs and always needs to know the type to set margins. *type* is a name listed in *term*(5). If no **−T** flag is supplied, *tabs* uses the value of the environment variable **TERM**. If **TERM** is not defined in the *environment* (see *environ*(5)), *tabs* tries a sequence that will work for many terminals.

+m*n*

The margin argument may be used for some terminals. It causes all tabs to be moved over *n* columns by making column *n+1* the left margin. If **+m** is given without a value of *n*, the value assumed is **10**. For a TermiNet, the first value in the tab list should be **1**, or the margin will move even further to the right. The normal (leftmost) margin on most terminals is obtained by **+m0**. The margin for most terminals is reset only when the **+m** flag is given explicitly.

Tab and margin setting is performed via the standard output.

EXAMPLES

tabs −a

example using *−code* (*canned* specification) to set tabs to the settings required by the IBM assembler: columns 1, 10, 16, 36, 72.

tabs −8

example of using *−n* (*repetitive* specification), where *n* is **8**, causes tabs to be set every eighth position:
1+(1*8), 1+(2*8), ... which evaluate to columns 9, 17, ...

tabs 1,8,36

example of using *n1,n2,...* (*arbitrary* specification) to set tabs at columns 1, 8, and 36.

tabs —$HOME/fspec.list/att4425

example of using *−file* (*file* specification) to indicate that tabs should be set according to the first line of **$HOME/fspec.list/att4425** (see *fspec*(4)).

DIAGNOSTICS

illegal tabs

when arbitrary tabs are ordered incorrectly

illegal increment
when a zero or missing increment is found in an arbitrary specification

unknown tab code
when a *canned* code cannot be found

can't open
if *—file* option used, and file can't be opened

file indirection
if *—file* option used and the specification in that file points to yet another file. Indirection of this form is not permitted

SEE ALSO

newform(1), pr(1), tput(1), fspec(4), terminfo(4), environ(5), term(5).

NOTE

There is no consistency among different terminals regarding ways of clearing tabs and setting the left margin.

tabs clears only 20 tabs (on terminals requiring a long sequence), but is willing to set 64.

WARNING

The *tabspec* used with the *tabs* command is different from the one used with the *newform*(1) command. For example, **tabs –8** sets every eighth position; whereas **newform –i–8** indicates that tabs are set every eighth position.

TAIL(1)

NAME

tail – deliver the last part of a file

SYNOPSIS

tail [± [*number*][**lbc** [**f**]]] [*file*]

DESCRIPTION

tail copies the named file to the standard output beginning at a designated place. If no file is named, the standard input is used.

Copying begins at distance +*number* from the beginning, or −*number* from the end of the input (if *number* is null, the value 10 is assumed). *Number* is counted in units of lines, blocks, or characters, according to the appended option **l**, **b**, or **c**. When no units are specified, counting is by lines.

With the **−f** ("follow") option, if the input file is not a pipe, the program will not terminate after the line of the input file has been copied, but will enter an endless loop, wherein it sleeps for a second and then attempts to read and copy further records from the input file. Thus it may be used to monitor the growth of a file that is being written by some other process. For example, the command:

tail −f fred

will print the last ten lines of the file **fred**, followed by any lines that are appended to **fred** between the time *tail* is initiated and killed. As another example, the command:

tail −15cf fred

will print the last 15 characters of the file **fred**, followed by any lines that are appended to **fred** between the time *tail* is initiated and killed.

SEE ALSO

dd(1M).

BUGS

Tails relative to the end of the file are stored in a buffer, thus, are limited in length. Various kinds of anomalous behavior may happen with character special files.

WARNING

The *tail* command will only tail the last 4096 bytes of a file regardless of its line count.

NAME

tar – tape file archiver

SYNOPSIS

/bin/tar **–c[iLvwfb[#*s*]]** *device block files ...*
/bin/tar **–r[iLvwb[#*s*]]** *device block [files ...]*
/bin/tar **–t[iLvf[#*s*]** *device*
/bin/tar **–u[iLvwb[#*s*]]** *device block [files ...]*
/bin/tar **–x[iLlmovwf[#*s*]]** *device [files ...]*

DESCRIPTION

tar saves and restores files on magnetic tape. Its actions are controlled by the *key* argument. The *key* is a string of characters containing one function letter (**c**, **r**, **t**, **u**, or **x**) and possibly followed by one or more function modifiers (**b**, **f**, **i**, **L**, **v**, **w**, and **#**). Other arguments to the command are *files* (or directory names) specifying which files are to be dumped or restored. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

The function portion of the key is specified by one of the following letters:

r

Replace. The named *files* are written on the end of the tape. The **c** function implies this function.

x

Extract. The named *files* are extracted from the tape. If a named file matches a directory whose contents had been written onto the tape, this directory is (recursively) extracted. Use the file or directory's relative path when appropriate, or *tar* does not find a match. The owner, modification time, and mode are restored (if possible). If no *files* argument is given, the entire content of the tape is extracted. Note that if several files with the same name are on the tape, the last one overwrites all earlier ones.

t

Table. The names and other information for the specified files are listed each time that they occur on the tape. The listing is similar to the format produced by the **ls -l** command. If no *files* argument is given, all names on the tape are listed.

u

Update. The named *files* are added to the tape if they are not already there, or have been modified since last written on that tape. This key implies the **r** key.

c

Create a new tape; writing begins at the beginning of the tape, instead of after the last file. This key implies the **r** key.

The following characters may be used in addition to the letter that selects the desired function. Use them in the order shown in the synopsis:

i

This modifier causes *tar* to ignore symbolic links.

L

This modifier causes *tar* to follow symbolic links. The default is not to follow links. If an archive is made from a tree containing symbolic links, it records the path associated with each link. When it is restored, the symbolic links are remade. If **–L** is specified, the actual file pointed to by the link is archived instead of the symbolic link contents.

#s

This modifier determines the drive on which the tape is mounted (replace # with the drive number) and the speed of the drive (replace **s** with **l**, **m**, or **h** for low, medium or high, respectively). The modifier tells *tar* to use a drive other than the default drive, or the drive

specified with the **–f** option. For example, with the **5h** modifier, *tar* would use **/dev/mt/5h** or **/dev/mt0** instead of the default drives **/dev/mt/0m** or **/dev/mt0**, respectively. However, if "**-f /dev/rmt0 5h**" appeared on the command line, *tar* would use **/dev/rmt5h** or **/devmt0**. The default entry is **0m**.

v

Verbose. Normally, *tar* does its work silently. The **v** (verbose) option causes it to type the name of each file it treats, preceded by the function letter. With the **t** function, **v** gives more information about the tape entries than just the name.

w

What. This causes *tar* to print the action to be taken, followed by the name of the file, and then wait for the user's confirmation. If a word beginning with **y** is given, the action is performed. Any other input means *no*. This is not valid with the **t** key.

f

File. This causes *tar* to use the *device* argument as the name of the archive instead of **/dev/rmt/ctape**. (Note that the cartridge tape in the REAL/IX system is referenced as **/dev/rmt/ctape**.) If the name of the file is –, *tar* writes to the standard output or reads from the standard input, whichever is appropriate. Thus, *tar* can be used as the head or tail of a pipeline. *tar* can also be used to move hierarchies with the command:

cd fromdir; tar cf – . | (cd todir; tar xf –)

b

Blocking Factor. This causes *tar* to use the *block* argument as the blocking factor for tape records. The default is 1, the maximum is 20. This function should not be supplied when operating on regular archives or block special devices. It is mandatory however, when reading archives on raw magnetic tape archives (see **f** above). The block size is determined automatically when reading tapes created on block special devices (key letters **x** and **t**).

l

Link. This tells *tar* to complain if it cannot resolve all of the links to the files being dumped. If **l** is not specified, no error messages are printed.

m

Modify. This tells *tar* to not restore the modification times. The modification time of the file is the time of extraction.

o

Ownership. This causes extracted files to take on the user and group identifier of the user running the program, instead of those on tape. This is only valid with the **x** key.

FILES

/dev/mt/*
/tmp/tar*
/dev/rmt/*

SEE ALSO

ar(1), cpio(1), ls(1).

DIAGNOSTICS

Complaints about bad key characters and tape read/write errors.
Complaints if enough memory is not available to hold the link tables. Complaints about directories which do not have search permissions.

tar(1) will output one of the following messages if it finds an incorrect entry in the **/etc/group** file:

fgetgrent: too many users in group < *group***>**
fgetgrent: line exceeds system limit (*limit***)**

where:

group
is the invalid entry

limit
is the system limit for the length of a line.

The output of one of these messages is a warning only, and does not result in the failure of the command.

BUGS

There is no way to ask for the *n*-th occurrence of a file.
Tape errors are handled ungracefully.
The **−r** and **−u** options are not supported on 88K tape devices.
The **b** option should not be used with archives that are going to be updated. The current magnetic tape driver cannot backspace raw magnetic tape. If the archive is on a disk file, the **b** option should not be used at all, because updating an archive stored on disk can destroy it.
The current limit on filename length is 100 characters.
tar does not copy empty directories or special files.

NAME

tee – pipe fitting

SYNOPSIS

tee [**–i**] [**–a**] [*file*] ...

DESCRIPTION

tee transcribes the standard input to the standard output and makes copies in the *files*. The

–i ignore interrupts;

–a causes the output to be appended to the *files* rather than overwriting them.

NAME

telnet – user interface to the TELNET protocol.

SYNOPSIS

telnet [*host* [*port*]]

DESCRIPTION

telnet is used to communicate with another host using the TELNET protocol. If *telnet* is invoked without arguments, it enters command mode, indicated by its prompt (**telnet >**). In this mode, it accepts and executes the commands listed below. If it is invoked with arguments, it performs an **open** command (see below) with those arguments.

Once a connection has been opened, *telnet* enters input mode. In this mode, text typed is sent to the remote host. To issue *telnet* commands when in input mode, precede them with the *telnet* escape character (initially **ESCAPE (CTRL-]**)). When in command mode, the normal terminal editing conventions are available.

The following commands are available. Only enough of each command to uniquely identify it need be typed.

bye
Synonym for quit

close
Close a TELNET session and return to command mode.

connect *host* [*port*]
Synonym for open

crmod
Toggle carriage return mode. When this mode is enabled, any carriage return characters received from the remote host are mapped into a carriage return and a line feed. This mode does not affect those characters typed by the user, only those received. This mode is not very useful, but is required for some hosts that like to ask the user to do local echoing. Initially, the carriage return characters are not mapped to a carriage return and line feed.

debug
Toggles socket level debugging (useful only to the superuser). The initial value is FALSE.

escape [*escape-char*]
Set the *telnet* escape character.

help [*command*]
Synonym for ?

open *host* [*port*]
Open a connection to the named host. If no port number is specified, *telnet* attempts to contact a TELNET server at the default port. The host specification may be either a host name (see *hosts*(4N)) or an Internet address specified in the dot notation.

options
Toggle viewing of TELNET options processing. When options viewing is enabled, all TELNET option negotiations are displayed. Options sent by *telnet* are displayed as SENT, while options received from the TELNET server are displayed as RCVD. Initially, options processing will not be shown.

quit
Close any open TELNET session and exit *telnet*.

send
Transmit special characters.

status
Show the current status of *telnet*. This includes the peer one is connected to, as well as the state of debugging.

Z
Escape to shell. The environment variable SHELL is used to spawn a shell. When the shell exits, *telnet* returns to input mode.

? [*command*]
Get help. With no arguments, *telnet* prints a help summary. If a command is specified, *telnet* prints the help information available about the command only.

BUGS

This implementation is very simple because *rlogin*(1N) is the standard mechanism used to communicate locally with hosts.

NAME

test – condition evaluation command

SYNOPSIS

test *expr*
[*expr*]

DESCRIPTION

test evaluates the expression *expr* and, if its value is true, sets a zero (true) exit status; otherwise, sets a non-zero (false) exit status. *test* also sets a non-zero exit status if there are no arguments. When permissions are tested, the effective user ID of the process is used.

All operators, flags, and brackets (brackets used as shown in the second **SYNOPSIS** line) must be separate arguments to the *test* command; normally these items are separated by spaces.

The following primitives are used to construct *expr*:

–r *file*
true if *file* exists and is readable.

–w *file*
true if *file* exists and is writable.

–x *file*
true if *file* exists and is executable.

–f *file*
true if *file* exists and is a regular file.

–d *file*
true if *file* exists and is a directory.

–c *file*
true if *file* exists and is a character special file.

–b *file*
true if *file* exists and is a block special file.

–l *file*
true if *file* exists and is a symbolic link.

–p *file*
true if *file* exists and is a named pipe (fifo).

–u *file*
true if *file* exists and its set-user-ID bit is set.

–g *file*
true if *file* exists and its set-group-ID bit is set.

–k *file*
true if *file* exists and its sticky bit is set.

–s *file*
true if *file* exists and has a size greater than zero.

–t [*fildes*]
true if the open file whose file descriptor number is *fildes* (1 by default) is associated with a terminal device.

–z *s1*
true if the length of string *s1* is zero.

−n *s1*
true if the length of the string *s1* is non-zero.

s1 **=** *s2*
true if strings *s1* and *s2* are identical.

s1 **!=** *s2*
true if strings *s1* and *s2* are *not* identical.

s1
true if *s1* is *not* the null string.

n1 **−eq** *n2*
true if the integers *n1* and *n2* are algebraically equal. Any of the comparisons **−ne**, **−gt**, **−ge**, **−lt**, and **−le** may be used in place of **−eq**.

These primaries may be combined with the following operators:

!
unary negation operator

−a
binary *and* operator

−o
binary *or* operator (**−a** has higher precedence than **−o**)

(expr)
parentheses for grouping. Notice also that parentheses are meaningful to the shell, therefore, must be quoted.

SEE ALSO

find(1), sh(1).

WARNING

If you test a file you own (the **-r**, **-w**, or **-x** tests), but the permission tested does not have the *owner* bit set, a non-zero (false) exit status will be returned even though the file may have the *group* or *other* bit set for that permission. The correct exit status will be set if you are superuser.

The **=** and **!=** operators have a higher precedence than the **−r** through **−n** operators, and **=** and **!=** always expect arguments; therefore, they cannot be used with the **−r** through **−n** operators.

If more than one argument follows the **−r** through **−n** operators, only the first argument is examined; the others are ignored, unless an **−a** or an **−o** is the second argument.

NAME

time – time a command

SYNOPSIS

time *command*

DESCRIPTION

The *command* is executed; after it is complete, *time* prints the elapsed time during the command, the time spent in the system, and the time spent in execution of the command. Times are reported in seconds.

The times are printed on standard error.

SEE ALSO

times(2).

NAME

timex – time a command; report process data and system activity

SYNOPSIS

timex [*options*] *command*

DESCRIPTION

The given *command* is executed; the elapsed time, user time and system time spent in execution are reported in seconds. Optionally, process accounting data for the *command* and all its children can be listed or summarized, and total system activity during the execution interval can be reported.

The output of *timex* is written on standard error.

The *options* are:

–p

List process accounting records for *command* and all its children. This option works only if the process accounting software is installed. Suboptions **f** , **h** , **k** , **m** , **r** , and **t** modify the data items reported. The options are:

–f Print the *fork/exec* flag and system exit status columns in the output.

–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).

–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. The number of blocks read or written and the number of characters transferred are always reported.

–o

Report the total number of blocks read or written and total characters transferred by *command* and all its children. This option works only if the process accounting software is installed.

–s

Report total system activity (not just that due to *command*) that occurred during the execution interval of *command*. All the data items listed in *sar*(1) are reported.

SEE ALSO

sar(1)

WARNING

Process records associated with *command* are selected from the accounting file **/usr/adm/pacct** by inference, since process genealogy is not available. Background processes having the same user-id, terminal-id, and execution time window will be spuriously included.

EXAMPLES

A simple example:

timex –ops sleep 60

A terminal session of arbitrary complexity can be measured by timing a sub-shell:

timex –opskmt sh

session commands
EOT

NAME

touch – update access and modification times of a file

SYNOPSIS

touch [**–amc**] [*mmddhhmm* [*yy*]] *files*

DESCRIPTION

touch causes the access and modification times of each argument to be updated. The file name is created if it does not exist. If no time is specified (see *date*(1)) the current time is used. The **–a** and **–m** options cause touch to update only the access or modification times respectively (default is **–am**). The **–c** option silently prevents *touch* from creating the file if it did not previously exist.

The return code from *touch* is the number of files for which the times could not be successfully modified (including files that did not exist and were not created).

SEE ALSO

date(1), utime(2).

NAME

tput – initialize a terminal or query terminfo database

SYNOPSIS

tput [**–T** *type*] *capname* [*parms* ...]

tput [**–T** *type*] **init**

tput [**–T** *type*] **reset**

tput [**–T** *type*] **longname**

tput –S << *file*

DESCRIPTION

tput uses the *terminfo*(4) database to make the values of terminal-dependent capabilities and information available to the shell (see *sh*(1)), to initialize or reset the terminal, or return the long name of the requested terminal type.

tput outputs a string if the attribute (*cap*ability *name*) is of type string, or an integer if the attribute is of type integer. If the attribute is of type boolean, *tput* simply sets the exit code (**0** for TRUE if the terminal has the capability, **1** for FALSE if it does not), and produces no output. Before using a value returned on standard output, the user should test the exit code ($?, see *sh*(1)) to be sure it is **0**. (See **EXIT CODES** and **DIAGNOSTICS**.) For a complete list of capabilities and the *capname* associated with each, see *terminfo*(4).

–T*type*
indicates the *type* of terminal. Normally, this option is unnecessary because the default is taken from the environment variable TERM. If **–T** is specified, the shell variables LINES and COLUMNS and the layer size (see *layers*(1)) will not be referenced.

capname
indicates the attribute from the *terminfo*(4) database.

parms
If the attribute is a string that takes parameters, the arguments *parms* will be instantiated into the string. An all numeric argument will be passed to the attribute as a number.

–S
allows more than one capability per invocation of *tput*. The capabilities must be passed to *tput* from the standard input instead of from the command line (see example). Only one *capname* is allowed per line. The **–S** option changes the meaning of the **0** and **1** boolean and string exit codes (see **EXIT CODES**).

init
If the *terminfo*(4) database is present and an entry for the user's terminal exists (see **–T***type*, above), the following will occur: (1) if present, the terminal's initialization strings will be output (**is1**, **is2**, **is3**, **if**, **iprog**), (2) any delays (e.g., newline) specified in the entry will be set in the TTY driver, (3) tabs expansion will be turned on or off according to the specification in the entry, and (4) if tabs are not expanded, standard tabs will be set (every 8 spaces). If an entry does not contain the information needed for any of the four above activities, that activity will silently be skipped.

reset
Instead of putting out initialization strings, the terminal's reset strings will be output if present (**rs1**, **rs2**, **rs3**, **rf**). If the reset strings are not present, but initialization strings are, the initialization strings will be output. Otherwise, **reset** acts identically to **init**.

longname
If the *terminfo*(4) database is present and an entry for the user's terminal exists (see **–T***type* above), then the long name of the terminal will be put out. The long name is the last name in the first line of the terminal's description in the *terminfo*(4) database (see *term*(5)).

EXAMPLES

tput init
Initialize the terminal according to the type of terminal in the environmental variable TERM. This command should be included in everyone's **.profile** after the environmental variable TERM has been exported, as illustrated on the *profile*(4) manual page.

tput –T5620 reset
Reset an AT&T 5620 terminal, overriding the type of terminal in the environmental variable TERM.

tput cup 0 0
Send the sequence to move the cursor to row **0**, column **0** (upper left corner of the screen, usually known as "home" cursor position).

tput clear
Echo the clear-screen sequence for the current terminal.

tput cols
Print the number of columns for the current terminal.

tput -T450 cols
Print the number of columns for the 450 terminal.

bold=‘tput smso‘
offbold=‘tput rmso‘
Set the shell variables **bold**, to begin stand-out mode sequence, and **offbold**, to end standout mode sequence, for the current terminal. This might be followed by a prompt:

```
echo "${bold}Please type in your name: ${offbold}\c"
```

tput hc
Set exit code to indicate if the current terminal is a hardcopy terminal.

tput cup 23 4
Send the sequence to move the cursor to row 23, column 4.

tput longname
Print the long name from the *terminfo*(4) database for the type of terminal specified in the environmental variable TERM.

```
tput -S <<!
> clear
> cup 10 10
> bold
> !
```

This example shows *tput* processing several capabilities in one invocation. This example clears the screen, moves the cursor to position 10, 10 and turns on bold (extra bright) mode. The list is terminated by an exclamation mark (!) on a line by itself.

FILES

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

/usr/include/curses.h
curses(3X) header file

/usr/include/term.h
terminfo(4) header file

/usr/lib/tabset/* tab settings for some terminals in a format appropriate to be output to the terminal (escape sequences that set margins and tabs); for more information, see the *Tabs and Initialization* section of *terminfo*(4).

SEE ALSO

stty(1), tabs(1), profile(4), terminfo(4).

EXIT CODES

If *capname* is of type boolean, a value of **0** is set for TRUE and **1** for FALSE unless the **–S** option is used.

If *capname* is of type string, a value of **0** is set if the *capname* is defined for this terminal *type* (the value of *capname* is returned on standard output); a value of **1** is set if *capname* is not defined for this terminal *type* (a **NULL** value is returned on standard output).

If *capname* is of type boolean or string and the **–S** option is used, a value of **0** is returned to indicate that all lines were successful. No indication of which line failed can be given so exit code **1** will never appear. Exit codes **2**, **3**, and **4** retain their usual interpretation.

If *capname* is of type integer, a value of **0** is always set, whether or not *capname* is defined for this terminal *type*. To determine if *capname* is defined for this terminal *type*, the user must test the value of standard output. A value of **–1** means that *capname* is not defined for this terminal *type*.

Any other exit code indicates an error; see **DIAGNOSTICS**.

DIAGNOSTICS

tput prints the following error messages and sets the corresponding exit codes.

Exit Code	Error Message
0	**–1 (*capname* is a numeric variable that is not specified in the *terminfo*(4) database for this terminal type, for example: tput –T450 lines and tput –T2621 xmc)**
1	no error message is printed, see **EXIT CODES**, above.
2	usage error
3	unknown terminal *type* or no *terminfo*(4) database
4	unknown *terminfo*(4) capability *capname*

NAME

tr – translate characters

SYNOPSIS

tr [**–cds**] [*string1* [*string2*]]

DESCRIPTION

tr copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. Any combination of the options **–cds** may be used:

–c
Complements the set of characters in *string1* with respect to the universe of characters whose ASCII codes are 001 through 377 octal.

–d
Deletes all input characters in *string1*.

–s
Squeezes all strings of repeated output characters that are in *string2* to single characters.

The following abbreviation conventions may be used to introduce ranges of characters or repeated characters into the strings:

[a–z]
Stands for the string of characters whose ASCII codes run from character **a** to character **z**, inclusive.

[a*n]
Stands for *n* repetitions of **a**. If the first digit of *n* is **0**, *n* is considered octal; otherwise, *n* is taken to be decimal. A zero or missing *n* is taken to be huge; this facility is useful for padding *string2*.

The escape character \ may be used as in the shell to remove special meaning from any character in a string. In addition, \ followed by 1, 2, or 3 octal digits stands for the character whose ASCII code is given by those digits.

EXAMPLE

The following example creates a list of all the words in *file1* one per line in *file2*, where a word is taken to be a maximal string of alphabetics. The strings are quoted to protect the special characters from interpretation by the shell; 012 is the ASCII code for newline.

```
tr –cs "[A–Z][a–z]" "[\012*]" <file1 >file2
```

SEE ALSO

ed(1), sh(1), ascii(5).

BUGS

Will not handle ASCII **NUL** in *string1* or *string2*; always deletes **NUL** from input.

NAME

true, false – provide truth values

SYNOPSIS

true

false

DESCRIPTION

true does nothing, successfully. *False* does nothing, unsuccessfully. They are typically used in input to *sh*(1), for example:

```
while true
do
    command
done
```

SEE ALSO

sh(1).

DIAGNOSTICS

true has exit status zero, *false* nonzero.

NAME

tsort – topological sort

SYNOPSIS

tsort [*file*]

DESCRIPTION

The *tsort* command produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input *file*. If no *file* is specified, the standard input is understood.

The input consists of pairs of items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

SEE ALSO

lorder(1).

DIAGNOSTICS

Odd data: there is an odd number of fields in the input file.

NAME

tt — convert and copy a file

SYNOPSIS

tt if=*file* **of=***file* **bs=***n* **count=***n*

DESCRIPTION

tt copies the specified input file to the specified output; the raw devices will be used by default:

if=*file* input disk alias searched for in the **permissions** file.

of=*file* output disk alias searched for in the **permissions** file.

bs=*n* set both input and output block size.

count=*n* copy only *n* input records.

Where *n* is specified, a number of bytes are 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 a product.

FILES

/bin/dd
/etc/filesys **permissions** file

SEE ALSO

cp(1), dd(1), filesys(4).

DIAGNOSTICS

After completion, *tt* reports the number of whole and partial input and output blocks:

full+partial records in
full+partial records out

BUGS

Only files listed in the **permissions** file may be accessed.

NAME

tty – get the name of the terminal

SYNOPSIS

tty [–l] [–s]

DESCRIPTION

tty prints the pathname of the user's terminal:

–l
prints the synchronous line number to which the user's terminal is connected if it is on an active synchronous line.

–s
inhibits printing of the terminal pathname, allowing you to test just the exit code.

Exit Codes	Description
2	if invalid options were specified
0	if standard input is a terminal
1	otherwise

DIAGNOSTICS

"not on an active synchronous line" if the standard input is not a synchronous terminal and –l is specified.

"not a tty" if the standard input is not a terminal and –s is not specified.

NAME

umask – set file-creation mode mask

SYNOPSIS

umask [ooo]

DESCRIPTION

The user file-creation mode mask is set to *ooo*. The three octal digits refer to read/write/execute permissions for *owner*, *group*, and *others*, respectively (see *chmod*(2) and *umask*(2)). The value of each specified digit is subtracted from the corresponding "digit" specified by the system for the creation of a file (see *creat*(2)). For example, **umask 022** removes *group* and *others* write permission (files normally created with mode **777** become mode **755**; files created with mode **666** become mode **644**).

If *ooo* is omitted, the current value of the mask is printed.

umask is recognized and executed by the shell.

umask can be included in the user's **.profile** (see *profile*(4)) and invoked at login to automatically set the user's permissions on files or directories created.

SEE ALSO

chmod(1), sh(1), chmod(2), creat(2), umask(2), profile(4).

NAME

uname – print name of current system

SYNOPSIS

uname [-snrvma] [-S nodename]

DESCRIPTION

uname prints the name of the current system on standard output. It is mainly useful to determine which system you are using. The options cause selected information returned by *uname*(2) to be printed:

-s print the operating system name.

-n print the nodename. The nodename is the name by which the system is known to a communications network. (default)

-r print the operating system release.

-v print the operating system version.

-m print the machine hardware name.

-a print all the above information.

-S nodename
change the system nodename.

The nodename argument is restricted to 8 characters. Only the superuser is allowed this capability.

SEE ALSO

uname(2).

NAME

unget – undo a previous get of an SCCS file

SYNOPSIS

unget [*–r SID*] [**–s**] [**–n**] *files*

DESCRIPTION

unget undoes the effect of a **get –e** done before creating the intended new delta. If a directory is named, *unget* behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of – is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

Keyletter arguments apply independently to each named file.

–r*SID*

Uniquely identifies which delta is no longer intended. (This would have been specified by *get* as the "new delta"). The use of this keyletter is necessary only if two or more outstanding *get*s for editing on the same SCCS file were done by the same person (login name). A diagnostic results if the specified *SID* is ambiguous, or if it is necessary and omitted on the command line.

–s

Suppresses the printout, on the standard output, of the intended delta's *SID*.

–n

Causes the retention of the gotten file which would normally be removed from the current directory.

SEE ALSO

delta(1), get(1), help(1), sact(1).

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

unifdef – remove ifdef'ed lines

SYNOPSIS

unifdef [**–t** **–l** **–c** **–D***sym* **–U***sym* **–id***sym* **–iu***sym*] ... [file]

DESCRIPTION

Unifdef is useful for removing ifdef'ed lines from a file while otherwise leaving the file alone. *Unifdef* is like a stripped-down C preprocessor: it is smart enough to deal with the nested ifdefs, comments, single and double quotes of C syntax so that it can do its job, but it doesn't do any including or interpretation of macros. Neither does it strip out comments, though it recognizes and ignores them. You specify which symbols you want defined **–D***sym* or undefined **–U***sym* and the lines inside those ifdefs will be copied to the output or removed as appropriate. The ifdef, ifndef, else, and endif lines associated with *sym* will also be removed. Ifdefs involving symbols you don't specify are untouched and copied out along with their associated ifdef, else, and endif lines. If an ifdef X occurs nested inside another ifdef X, then the inside ifdef is treated as if it were an unrecognized symbol. If the same symbol appears in more than one argument, only the first occurrence is significant.

The **–l** option causes *unifdef* to replace removed lines with blank lines instead of deleting them.

If you use ifdefs to delimit non-C lines, such as comments or code which is under construction, then you must tell *unifdef* which symbols are used for that purpose so that it won't try to parse for quotes and comments in those ifdef'ed lines. You specify that you want the lines inside certain ifdefs to be ignored but copied out with **–id***sym* and **–iu***sym* similar to **–D***sym* and **–U***sym* above.

If you want to use *unifdef* for plain text (not C code), use the **–t** option. This makes *unifdef* refrain from attempting to recognize comments and single and double quotes.

Unifdef copies its output to *stdout* and will take its input from *stdin* if no *file* argument is given. If the **–c** argument is specified, then the operation of *unifdef* is complemented, i.e. the lines that would have been removed or blanked are retained and vice versa.

SEE ALSO

diff(1B).

DIAGNOSTICS

Premature EOF, inappropriate else or endif.

Exit status is 0 if output is exact copy of input, 1 if not, 2 if trouble.

BUGS

Does not know how to deal with *cpp* constructs such as

```
#if     defined(X) || defined(Y)
```

AUTHOR

Dave Yost

NAME

uniq – report repeated lines in a file

SYNOPSIS

uniq [**–udc** [**+***n*] [**–***n*]] [*input* [*output*]]

DESCRIPTION

uniq reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. *Input* and *output* should always be different. Note that repeated lines must be adjacent in order to be found; see *sort*(1). If the **–u** flag is used, just the lines that are not repeated in the original file are output. The **–d** option specifies that one copy of just the repeated lines is to be written. The normal mode output is the union of the **–u** and **–d** mode outputs.

The **–c** option supersedes **–u** and **–d** and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

The *n* arguments specify skipping an initial portion of each line in the comparison:

–*n*
The first *n* fields together with any blanks before each are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.

+*n*
The first *n* characters are ignored. Fields are skipped before characters.

SEE ALSO

comm(1), sort(1).

NAME

units – conversion program

SYNOPSIS

units

DESCRIPTION

units converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively in this fashion:

```
You have: inch
You want: cm
        * 2.540000e+00
        / 3.937008e-01
```

A quantity is specified as a multiplicative combination of units optionally preceded by a numeric multiplier. Powers are indicated by suffixed positive integers, division by the usual sign:

```
You have: 15 lbs force/in2
You want: atm
        * 1.020689e+00
        / 9.797299e-01
```

units only does multiplicative scale changes; thus, it can convert Kelvin to Rankine, but not Celsius to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, together with a generous leavening of exotica and a few constants of nature including:

pi	ratio of circumference to diameter,
c	speed of light,
e	charge on an electron,
g	acceleration of gravity,
force	same as **g**,
mole	Avogadro's number,
water	pressure head per unit height of water,
au	astronomical unit.

pound is not recognized as a unit of mass; **lb** is. Compound names are run together, (e.g., **lightyear**). British units that differ from their U.S. counterparts are prefixed: **brgallon**. For a complete list of units, type:

```
cat /usr/lib/unittab
```

FILES

/usr/lib/unittab

NAME

unixsyms - kernel debugger symbol table generator

SYNOPSIS

unixsyms [-w] [-s **sfile**] [-i **ifile**] [-o **ofile**] objfil

DESCRIPTION

unixsyms is used with the ***-w option*** to initialize the kernel symbol table required by the kernel debugger of the REAL/IX operating system. See ***db***(1M) for a description of this debugger.

External (i.e., global) symbols are obtained by reading ***objfil***, which is assumed to not be stripped.

Structure offsets are obtained by reading ***sfile***, which is assumed to be the -g compilation of a file including the header files defining all structures of interest. See options for pcc, *cc*(1).

Additionally, two other options can be used together to permit information of objects (symbols) specified in the first file to appear in the second. These are the ifile and ofile respectively. The information saved in ***ofile*** is the value of the symbol(s) specified in ***ifile***.

The symbol table of the operating system must appear in the .data section of ***objfil***, and must be large enough to accommodate all symbols the options require. The symbol table is ordered by value; structure offsets appear first, and the names of structure fields are prepended with a '.', which must be used in ***db***(1M) in order to reference the offset of the structure field.

Unixsyms is run as one of the last steps in the build procedure of the operating system. If no symbol table is found in ***objfil***, the file is not modified, the -w option notwithstanding.

NAME

usage – retrieve a command description and usage examples

SYNOPSIS

[**help**] **usage** [**–d**] [**–e**] [**–o**] [*command_name*]

DESCRIPTION

The Help Facility command *usage* retrieves information about commands. With no argument, *usage* displays a menu screen prompting the user for the name of a command, or allows the user to retrieve a list of commands supported by *usage*. The user may also exit to the shell by typing **q** (for "quit).

After a command is selected, the user is asked to choose among a description of the command, examples of typical usage of the command, or descriptions of the command's options. Then, based on the user's request, the appropriate information will be printed.

A command name may also be entered at shell level as an argument to *usage*. To receive information on the command's description, examples, or options, the user may use the **–d**, **–e**, or **–o** options respectively. (The default option is **–d**.)

From any screen in the Help Facility, a user may execute a command via the shell (*sh*(1)) by typing a ! and the command to be executed. The screen will be redrawn if the command that was executed 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

glossary(1), help(1), locate(1), sh(1), starter(1), term(5).

WARNINGS

If the shell variable TERM (see *sh*(1)) 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

uucp, uulog, uuname – UNIX-to-UNIX system copy

SYNOPSIS

uucp [*options*] *source-files destination-file*
uulog [*options*] *–ssystem*
uulog [*options*] *system*
uulog [*options*] *–fsystem*
uuname [–l] [–c]

DESCRIPTION

uucp

uucp copies files named by the *source-file* arguments to the *destination-file* argument. A source file name may be a pathname on your machine, or may have the form:

system-name!path-name

where *system-name* is taken from a list of system names that *uucp* knows about. The destination *system-name* may also be a list of names such as:

system-name!system-name!...!system-name!path-name

In this case an attempt is made to send the file via the specified route, to the destination. See **WARNINGS** and **BUGS** for restrictions. Care should be taken to ensure that intermediate nodes in the route are willing to forward information (see **WARNINGS** for restrictions).

The following shell metacharacters are disallowed in *system-name*:

‘ ; & | ^ < > ( ) <CR> <TAB> <SPACE>

Pathnames may be one of the following:

(1) a full pathname.

(2) a pathname preceded by *~user* where *user* is a login name on the specified system and is replaced by that user's login directory.

(3) a pathname preceded by *~/destination* where *destination* is appended to **/usr/spool/uucppublic**. Note that this destination will be treated as a file name unless more than one file is being transferred by this request or the destination is already a directory. To ensure that it is a directory, follow the destination with a '/'. For example, ~/dan/ as the destination will make the directory **/usr/spool/uucppublic/dan** if it does not exist and put the requested file(s) in that directory.

(4) anything else is prefixed by the current directory.

If the result is an erroneous pathname for the remote system the copy will fail. If the *destination-file* is a directory, the last part of the *source-file* name is used.

uucp preserves execute permissions across the transmission and gives 0666 read and write permissions (see *chmod*(2)).

The following options are interpreted by *uucp*:

–c
Do not copy local file to the spool directory for transfer to the remote machine (default).

–C
Force the copy of local files to the spool directory for transfer.

–d
Make all necessary directories for the file copy (default).

-f
Do not make intermediate directories for the file copy.

-g*grade*
grade is a single letter/number; lower ASCII sequence characters will cause the job to be transmitted earlier during a particular conversation.

-j
Output the job identification ASCII string on the standard output. This job identification can be used by *uustat* to obtain the status or terminate a job.

-m
Send mail to the requester when the copy is completed.

-n*user*
Notify *user* on the remote system that a file was sent.

-r
Do not start the file transfer, just queue the job.

-s*file*
Report status of the transfer to *file*. Note that the *file* must be a full pathname.

-x*debug_level*
Produce debugging output on standard output. The *debug_level* is a number between 0 and 9; higher numbers give more detailed information. (Debugging will not be available if **uucp** was compiled with -DSMALL.)

uulog
uulog queries a log file of *uucp* or *uuxqt* transactions in a file **/usr/spool/uucp/.Log/uucico/***system* or **/usr/spool/uucp/.Log/uuxqt/***system*.

The options cause *uulog* to print logging information:

-s*sys*
Print information about file transfer work involving system *sys*.

-f*system*
Does a "tail -f" of the file transfer log for *system*. (You must press **BREAK** to exit this function.) The following are other options used with the above:

-x
Look in the *uuxqt* log file for the given system.

-*number*
Indicates that a "tail" command of *number* lines should be executed.

uuname
uuname lists the names of systems known to *uucp*. The -c option returns the names of systems known to *cu*. (The two lists are the same, unless your machine is using different **Systems** files for *cu* and *uucp*. See the **Sysfiles** file.) The -l option returns the local system name.

FILES

/usr/spool/uucp	spool directories
/usr/spool/uucppublic/*	public directory for receiving and sending (**/usr/spool/uucppublic**)
/usr/lib/uucp/*	other data and program files

SEE ALSO
mail(1), uustat(1C), uux(1C), uuxqt(1M), chmod(2).

WARNINGS

The domain of remotely accessible files can (and for obvious security reasons, usually should) be severely restricted. You will probably not be able to fetch files by pathname; ask a responsible person on the remote system to send them to you. For the same reasons you will probably not be able to send files to arbitrary pathnames. As distributed, the remotely accessible files are those whose names begin **/usr/spool/uucppublic** (equivalent to **~/**).

All files received by *uucp* will be owned by *uucp*.

The **-m** option will only work sending files or receiving a single file. Receiving multiple files specified by special shell characters **? * [...]** will not activate the **-m** option.

The forwarding of files through other systems may not be compatible with the previous version of *uucp*. If forwarding is used, all systems in the route must have the same version of *uucp*.

BUGS

Protected files and files that are in protected directories that are owned by the requester can be sent by *uucp* using the **-C** option. However, if the requestor is **root**, and the directory is not searchable by "other" or the file is not readable by "other", the request will fail.

NAME

uustat – uucp status inquiry and job control

SYNOPSIS

uustat [**–a**]
uustat [**–m**]
uustat [**–p**]
uustat [**–q**]
uustat [**–k***jobid*]
uustat [**–r***jobid*]
uustat [**–s***system*] [**–u***user*]

DESCRIPTION

uustat will display the status of, or cancel, previously specified *uucp* commands, or provide general status on *uucp* connections to other systems. Only one of the following options can be specified with *uustat* per command execution:

–a
Output all jobs in queue.

–m
Report the status of accessibility of all machines.

–p
Execute a "ps –flp" for all the process-ids that are in the lock files.

–q
List the jobs queued for each machine. If a status file exists for the machine, its date, time and status information are reported. In addition, if a number appears in () next to the number of **C** or **X** files, it is the age in days of the oldest **C./X.** file for that system. The Retry field represents the number of hours until the next possible call. The Count is the number of failure attempts. Note that for systems with a moderate number of outstanding jobs, this could take 30 seconds or more of real-time to execute. As an example of the output produced by the **–q** option:

```
eagle      3C    04/07-11:07  NO DEVICES AVAILABLE
mh3bs3     2C    07/07-10:42  SUCCESSFUL
```

This output tells how many command files are waiting for each system. Each command file may have zero or more files to be sent (zero means to call the system and see if work is to be done). The date and time refer to the previous interaction with the system followed by the status of the interaction.

–k*jobid*
Kill the *uucp* request whose job identification is *jobid*. The killed *uucp* request must belong to the person issuing the *uustat* command unless one is the super-user.

–r*jobid*
Rejuvenate *jobid*. The files associated with *jobid* are touched so that their modification time is set to the current time. This prevents the cleanup daemon from deleting the job until the jobs modification time reaches the limit imposed by the deamon.

Either or both of the following options can be specified with *uustat*:

–s*sys*
Report the status of all *uucp* requests for remote system *sys*.

–u*user*
Report the status of all *uucp* requests issued by *user*.

Output for both the –s and –u options has the following format:

```
eaglen0000      4/07-11:01:03(POLL)
eagleN1bd7      4/07-11:07  Seagledan522 /usr/dan/A
eagleC1bd8      4/07-11:07  Seagledan59 D.3b2al2ce4924
                4/07-11:07  Seagledanrmail mike
```

With the above two options, the first field is the *jobid* of the job. This is followed by the date/time. The next field is either an 'S' or 'R' depending on whether the job is to send or request a file. This is followed by the user-id of the user who queued the job. The next field contains the size of the file, or in the case of a remote execution (*rmail* - the command used for remote mail), the name of the command. When the size appears in this field, the file name is also given. This can either be the name given by the user or an internal name (e.g., D.3b2alce4924) that is created for data files associated with remote executions (*rmail* in this example).

When no options are given, *uustat* outputs the status of all *uucp* requests issued by the current user.

FILES

/usr/spool/uucp/* spool directories

SEE ALSO

uucp(1C).

NAME

uuto, uupick – public UNIX-to-UNIX system file copy

SYNOPSIS

uuto [*options*] *source-files destination*
uupick [**–s** *system*]

DESCRIPTION

uuto sends *source-files* to *destination*. *uuto* uses the *uucp*(1C) facility to send files, while it allows the local system to control the file access. A source-file name is a pathname on your machine. Destination has the form:

system!*user*

where *system* is taken from a list of system names that *uucp* knows about (see *uuname*). *user* is the login name of someone on the specified system.

Two *options* are available:

–p
Copy the source file into the spool directory before transmission.

–m
Send mail to the sender when the copy is complete.

The files (or sub-trees if directories are specified) are sent to **PUBDIR** on *system*, where **PUBDIR** is a public directory defined in the *uucp* source. By default, this directory is **/usr/spool/uucppublic**. Specifically, the files are sent to:

PUBDIR/receive/*user***/***mysystem***/files.**

The destined recipient is notified by *mail*(1) of the arrival of files.

uupick accepts or rejects the files transmitted to the user. Specifically, *uupick* searches **PUBDIR** for files destined for the user. For each entry (file or directory) found, the following message is printed on the standard output:

from *system*: [file *file-name*] [dir *dirname*] ?

uupick then reads a line from the standard input to determine the disposition of the file:

<newline>
Go on to next entry.

d
Delete the entry.

m [*dir*]
Move the entry to named directory *dir*. If *dir* is not specified as a complete pathname (in which $HOME is legitimate), a destination relative to the current directory is assumed. If no destination is given, the default is the current directory.

a [*dir*]
Same as **m** except moving all the files sent from *system*.

p
Print the content of the file.

q
Stop.

EOT (**CTRL**)
Same as **q**.

!*command*
Escape to the shell to do *command*.

*
Print a command summary.

uupick invoked with the *-ssystem* option will only search the **PUBDIR** for files sent from *system*.

FILES

PUBDIR	**/usr/spool/uucppublic**	public directory

SEE ALSO

mail(1), uucp(1C), uustat(1C), uux(1C), uucleanup(1M).

WARNINGS

To send files that begin with a dot (e.g., **.profile**), the files must by qualified with a dot. For example: **.profile**, **.prof***, **.profil?** are correct; whereas ***prof***, **?profile** are incorrect.

NAME

uux – UNIX-to-UNIX system command execution

SYNOPSIS

uux [*options*] *command-string*

DESCRIPTION

uux will gather zero or more files from various systems, execute a command on a specified system and then send standard output to a file on a specified system.

NOTE: For security reasons, most installations limit the list of commands executable on behalf of an incoming request from *uux*, permitting only the receipt of mail (see *mail*(1)). (Remote execution permissions are defined in **/usr/lib/uucp/Permissions**.)

The *command-string* is made up of one or more arguments that look like a shell command line, except that the command and file names may be prefixed by *system-name*!. A **NULL** *system-name* is interpreted as the local system.

File names may be one of:

(1) a full pathname;

(2) a pathname preceded by ~*xxx* where *xxx* is a login name on the specified system and is replaced by that user's login directory;

(3) anything else is prefixed by the current directory.

As an example, the command:

uux "!diff usg!/usr/dan/file1 pwba!/a4/dan/file2 !~/dan/file.diff"

will get the *file1* and *file2* files from the "usg" and "pwba" machines, execute a *diff*(1) command and put the results in *file.diff* in the local **PUBDIR/dan/directory**.

Any special shell characters such as <>;| should be quoted either by quoting the entire *command-string*, or quoting the special characters as individual arguments.

uux will attempt to get all files to the execution system. For files that are output files, the file name must be escaped using parentheses. For example, the command:

uux a!cut -f1 b!/usr/file \(c!/usr/file\)

gets **/usr/file** from system "b" and sends it to system "a", performs a *cut* command on that file and sends the result of the *cut* command to system "c".

uux will notify you if the requested command on the remote system was disallowed. This notification can be turned off by the **–n** option. The response comes by remote mail from the remote machine.

The following *options* are interpreted by *uux*:

–
The standard input to *uux* is made the standard input to the *command-string*.

–a*name*
Use *name* as the user identification replacing the initiator user-id. (Notification will be returned to the user.)

–b
Return whatever standard input was provided to the *uux* command if the exit status is non-zero.

–c
Do not copy local file to the spool directory for transfer to the remote machine (default).

-C
Force the copy of local files to the spool directory for transfer.

-g*grade*
grade is a single letter/number; lower ASCII sequence characters will cause the job to be transmitted earlier during a particular conversation.

-j
Output the jobid ASCII string on the standard output which is the job identification. This job identification can be used by *uustat* to obtain the status or terminate a job.

-n
Do not notify the user if the command fails.

-p
Same as -: The standard input to *uux* is made the standard input to the *command-string*.

-r
Do not start the file transfer, just queue the job.

-s*file*
Report status of the transfer in *file*.

-x*debug_level*
Produce debugging output on the standard output. The *debug_level* is a number between 0 and 9; higher numbers give more detailed information.

-z
Send success notification to the user.

FILES

/usr/lib/uucp/spool	spool directories
/usr/lib/uucp/Permissions	remote execution permissions
/usr/lib/uucp/*	other data and programs

SEE ALSO

cut(1), mail(1), uucp(1C), uustat(1C).

WARNINGS

Only the first command of a shell pipeline may have a *system-name*!. All other commands are executed on the system of the first command.

The use of the shell metacharacter * will probably not do what you want it to do. The shell tokens << and >> are not implemented.

The execution of commands on remote systems takes place in an execution directory known to the *uucp* system. All files required for the execution will be put into this directory unless they already reside on that machine. Therefore, the simple file name (without path or machine reference) must be unique within the *uux* request. The following command will not work:

uux "a!diff b!/usr/dan/xyz c!/usr/dan/xyz > !xyz.diff"

but the command:

uux "a!diff a!/usr/dan/xyz c!/usr/dan/xyz > !xyz.diff"

will work (if *diff* is a permitted command).

BUGS

Protected files and files that are in protected directories that are owned by the requestor can be sent in commands using *uux*. However, if the requestor is **root**, and the directory is not searchable by "other", the request will fail.

NAME

val – validate SCCS file

SYNOPSIS

val –
val [**–s**] [**–r***SID*] [**–m***name*] [**–y***type*] *files*

DESCRIPTION

val determines if the specified *file* is an SCCS file meeting the characteristics specified by the optional argument list. Arguments to *val* may appear in any order. The arguments consist of keyletter arguments, which begin with a –, and named files.

val has a special argument, –, which causes reading of the standard input until an EOF condition is detected. Each line read is independently processed as if it were a command line argument list.

val generates diagnostic messages on the standard output for each command line and file processed, and also returns a single 8-bit code upon exit as described below.

The following defines keyletter arguments; the effects of any keyletter argument apply independently to each named file on the command line:

–s
The presence of this argument silences the diagnostic message normally generated on the standard output for any error that is detected while processing each named file on a given command line.

–r*SID*
The argument value *SID* is an SCCS delta number. A check is made to determine if the *SID* is ambiguous (e. g., **r**1 is ambiguous because it physically does not exist but implies 1.1, 1.2, which may exist) or invalid (e. g., **r**1.0 or **r**1.1.0 are invalid because neither case can exist as a valid delta number). If the *SID* is valid and not ambiguous, a check is made to determine if it actually exists.

–mname
The argument value *name* is compared with the s-1SCCS %M% keyword in *file*.

–ytype
The argument value *type* is compared with the SCCS %Y% keyword in *file*.

The 8-bit code returned by *val* is a disjunction of the possible errors, i. e., can be interpreted as a bit string where (moving from left to right) set bits are interpreted as:

bit 0 = missing file argument;
bit 1 = unknown or duplicate keyletter argument;
bit 2 = corrupted SCCS file;
bit 3 = cannot open file or file not SCCS;
bit 4 = *SID* is invalid or ambiguous;
bit 5 = *SID* does not exist;
bit 6 = %Y%, **–y** mismatch;
bit 7 = %M%, **–m** mismatch;

Note that *val* can process two or more files on a given command line and in turn can process multiple command lines (when reading the standard input). In these cases an aggregate code is returned —a logical **OR** of the codes generated for each command line and file processed.

SEE ALSO

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

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

val can process up to 50 files on a single command line. Any number above 50 will produce a core dump.

NAME

vc – version control

SYNOPSIS

vc [**–a**] [**–t**] [**–c***char*] [**–s**] [*keyword=value ... keyword=value*]

DESCRIPTION

The *vc* command copies lines from the standard input to the standard output under control of its *arguments* and *control statements* encountered in the standard input. In the process of performing the copy operation, user declared *keywords* may be replaced by their string *value* when they appear in plain text and/or control statements.

The copying of lines from the standard input to the standard output is conditional, based on tests (in control statements) of keyword values specified in control statements or as *vc* command arguments.

A control statement is a single line beginning with a control character, except as modified by the –t keyletter (see below). The default control character is colon (:), except as modified by the –c keyletter (see below). Input lines beginning with a backslash (\) followed by a control character are not control lines and are copied to the standard output with the backslash removed. Lines beginning with a backslash followed by a non-control character are copied in their entirety.

A keyword is composed of 9 or less alphanumerics; the first must be alphabetic. A value is any ASCII string that can be created with *ed*(1); a numeric value is an unsigned string of digits. Keyword values may not contain blanks or tabs.

Replacement of keywords by values is done whenever a keyword surrounded by control characters is encountered on a version control statement. The **–a** keyletter (see below) forces replacement of keywords in *all* lines of text. An uninterpreted control character may be included in a value by preceding it with \. If a literal \ is desired, then it too must be preceded by \.

Keyletter Arguments

–a
Forces replacement of keywords surrounded by control characters with their assigned value in *all* text lines and not just in *vc* statements.

–t
All characters from the beginning of a line up to and including the first *tab* character are ignored for the purpose of detecting a control statement. If one is found, all characters up to and including the *tab* are discarded.

–cchar
Specifies a control character to be used in place of :.

–s
Silences warning messages (not error) that are normally printed on the diagnostic output.

Version Control Statements

:dcl keyword[, ..., keyword]
Used to declare keywords. All keywords must be declared.

:asg keyword=value
Used to assign values to keywords. An **asg** statement overrides the assignment for the corresponding keyword on the *vc* command line and all previous **asg**'s for that keyword. Keywords declared, but not assigned values have **NULL** values.

```
:if condition
        .
        .
        .
:end
```

Used to skip lines of the standard input. If the condition is true all lines between the *if* statement and the matching *end* statement are copied to the standard output. If the condition is false, all intervening lines are discarded, including control statements. Note that intervening *if* statements and matching *end* statements are recognized solely for the purpose of maintaining the proper *if-end* matching.

The syntax of a condition is:

```
<cond>      ::= [ "not" ] <or>
<or>        ::= <and> | <and> "|" <or>
<and>       ::= <exp> | <exp> "&" <and>
<exp>       ::= "(" <or> ")" | <value> <op> <value>
<op>        ::= "=" | "!=" | "<" | ">"
<value>     ::= <arbitrary ASCII string> | <numeric string>
```

The available operators and their meanings are:

=	equal
!=	not equal
&	and
|	or
>	greater than
<	less than
( )	used for logical groupings
not	may only occur immediately after the *if*, and when present, inverts the value of the entire condition

The > and < operate only on unsigned integer values (e.g., : 012 > 12 is false). All other operators take strings as arguments (e.g., : 012 != 12 is true). The precedence of the operators (from highest to lowest) is:

```
= != > <      all of equal precedence
&
|
```

Parentheses may be used to alter the order of precedence.
Values must be separated from operators or parentheses by at least one blank or tab.

::text
Used for keyword replacement on lines that are copied to the standard output. The two leading control characters are removed; keywords surrounded by control characters in text are replaced by their value before the line is copied to the output file. This action is independent of the –a keyletter.

:on

:off
Turn on or off keyword replacement on all lines.

:ctl char
Change the control character to char.

:msg message
Prints the given message on the diagnostic output.

VC(1)

:err message

Prints the given message followed by:

ERROR: err statement on line ... (915)

on the diagnostic output. *vc* halts execution, and returns an exit code of 1.

SEE ALSO

ed(1), help(1).

DIAGNOSTICS

Use *help*(1) for explanations.

EXIT CODES

0 normal
1 any error

NAME

vi, view, vedit – screen-oriented (visual) display editor based on ex

SYNOPSIS

vi [**–t** *tag*] [**–r** *file*] [**–L**] [**–w***n*] [**–R**] [**–x**] [**–C**]
[**–c** *command*] *file* ...
view [**–t** *tag*] [**–r** *file*] [**–L**] [**–w** *n*] [**–R**] [**–x**] [**–C**]
[**–c** *command*] *file* ...
vedit [**–t** *tag*] [**–r** *file*] [**–L**] [**–w** *in*] [**–R**] [**–x**] [**–C**]
[**–c** *command*] *file* ...

DESCRIPTION

The *vi* (visual) program is a display-oriented text editor based on an underlying line editor *ex*(1). It is possible to use the command mode of **ex** from within *vi* and vice-versa. The visual commands are described on this manual page; how to set options (like automatically numbering lines and automatically starting a new output line when you type carriage return) and all *ex*(1) line editor commands are described on the *ex*(1) manual page.

When using *vi*, changes you make to the file are reflected in what you see on your terminal screen. The position of the cursor on the screen indicates the position within the file.

Invocation Options

The following invocation options are interpreted by *vi* (previously documented options are discussed in the **NOTES** section):

–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 name of all files saved as the result of an editor or system crash.

–w*n*
Set the default window size to *n*. This is useful when using the editor over a slow speed line.

–R
Readonly mode; the **readonly** flag is set, preventing accidental overwriting of the file.

–x
Encryption option; when used, *vi* simulates the *X* command of *ex*(1) 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 or not. 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 **WARNING** section.

–C
Encryption option; same as the **–x** option, except that *vi* simulates the *C* command of *ex*(1). The *C* command is like the *X* command of *ex*(1), 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.

The *view* invocation is the same as *vi* except that the **readonly** flag is set.

The *vedit* invocation is intended for beginners. It is the same as *vi* except that the **report** flag is set to 1, the **showmode** and **novice** flags are set, and **magic** is turned off. These defaults make it easier to learn how to use *vi*.

vi Modes

Command

Normal and initial mode. Other modes return to command mode upon completion. **ESC** (escape) is used to cancel a partial command.

Input

Entered by setting any of the following options: **a A i I o O c C s S R .** Arbitrary text may then be entered. Input mode is normally terminated with **ESC** character, or, abnormally, with an interrupt.

Last line

Reading input for **:** / **?** or **!**; terminate by typing a carriage return; an interrupt cancels termination.

COMMAND SUMMARY

In the descriptions, **CR** stands for carriage return and **ESC** stands for the escape key.

Sample commands

← ↓ ↑ →	arrow keys move the cursor
h j k l	same as arrow keys
i*text***ESC**	insert *text*
cw*new***ESC**	change word to *new*
ea*s***ESC**	pluralize word (end of word; append **s**; escape from input state)
x	delete a character
dw	delete a word
dd	delete a line
3dd	delete 3 lines
u	undo previous change
ZZ	exit *vi*, saving changes
:q!CR	quit, discarding changes
/*text***CR**	search for *text*
^U ^D	scroll up or down
:*cmd***CR**	any *ex* or *ed* command

Counts before vi commands

Numbers may be typed as a prefix to some commands. They are interpreted in one of three ways:

line/column number	**z G |**
scroll amount	**^D ^U**
repeat effect	most of the rest

Interrupting, canceling

ESC	end insert or incomplete command
DEL	(delete or rubout) interrupts

File manipulation

ZZ	if file modified, write and exit; otherwise, exit
:wCR	write back changes
:w!CR	forced write, if permission originally not valid

:qCR	quit
:q!CR	quit, discard changes
:e *name***CR**	edit file *name*
:e!CR	reedit, discard changes
:e + *name***CR**	edit, starting at end
:e +*n***CR**	edit starting at line *n*
:e #CR	edit alternate file
:e! #CR	edit alternate file, discard changes
:w *name***CR**	write file *name*
:w! *name***CR**	overwrite file *name*
:shCR	run shell, then return
:! *cmd***CR**	run *cmd*, then return
:nCR	edit next file in arglist
:n *args***CR**	specify new arglist
^G	show current file and line
:ta *tag***CR**	position cursor to *tag*

In general, any ex or ed command (such as *substitute* or *global*) may be typed, preceded by a colon and followed by a carriage return.

Positioning within file

^F	forward screen
^B	backward screen
^D	scroll down half screen
^U	scroll up half screen
***n*G**	go to the beginning of the specified line (end default), where *n* is a line number
/pat	next line matching *pat*
?*pat*	previous line matching *pat*
n	repeat last / or ? command
N	reverse last / or ? command
/pat/+n	nth line after *pat*
?*pat***?−***n*	nth line before *pat*
]]	next section/function
[[	previous section/function
(	beginning of sentence
)	end of sentence
{	beginning of paragraph
}	end of paragraph
%	find matching ( ) { or }

Adjusting the screen

^L	clear and redraw window
^R	clear and redraw window if ^L is → key
zCR	redraw screen with current line at top of window
z−CR	redraw screen with current line at bottom of window
z.CR	redraw screen with current line at center of window
/pat/z−CR	move *pat* line to bottom of window
z*n***.CR**	use *n*-line window
^E	scroll window down 1 line
^Y	scroll window up 1 line

Marking and returning

``` `` ```	move cursor to previous context
``` '' ```	move cursor to first non-white space in line
m*x*	mark current position with the ASCII lowercase letter *x*
``` `x ```	move cursor to mark *x*
``` 'x ```	move cursor to first non-white space in line marked by *x*

Line positioning

H	top line on screen
L	last line on screen
M	middle line on screen
+	next line, at first non-white
–	previous line, at first non-white
CR	return, same as +
↓ or **j**	next line, same column
↑ or **k**	previous line, same column

Character positioning

^	first non white-space character
0	beginning of line
$	end of line
h or →	forward
l or ←	backward
^H	same as ← (backspace)
space	same as → (space bar)
f*x*	find next *x*
F*x*	find previous **x**
t*x*	move to character prior to next *x*
T*x*	move to character following previous *x*
;	repeat last **f F t** or **T**
,	repeat inverse of last **f F t** or **T**
***n*|**	move to column *n*
%	find matching ({) or }

Words, sentences, paragraphs

w	forward a word
b	back a word
e	end of word
)	to next sentence
}	to next paragraph
(	back a sentence
{	back a paragraph
W	forward a blank-delimited word
B	back a blank-delimited word
E	end of a blank-delimited word

Corrections during insert

^H	erase last character (backspace)
^W	erase last word
erase	your erase character, same as **^H** (backspace)
kill	your kill character, erase this line of input
****	quotes your erase and kill characters
ESC	ends insertion, back to command mode
DEL	interrupt, terminates insert mode
^D	backtab one character; reset left margin of *autoindent*
^^D	caret (^) followed by control-d (^D); backtab to beginning of line; do not reset left margin of *autoindent*
0^D	backtab to beginning of line; reset left margin of *autoindent*
^V	quote non-printable character

Insert and replace

a	append after cursor
A	append at end of line
i	insert before cursor
I	insert before first non-blank
o	open line below
O	open above
r*x*	replace single char with *x*
R*text***ESC**	replace characters

Operators

Operators are followed by a cursor motion, and affect all text that would have been moved over. For example, since **w** moves over a word, **dw** deletes the word that would be moved over. Double the operator, e.g., **dd** to affect whole lines.

d	delete
c	change
y	yank lines to buffer
<	left shift
>	right shift
!	filter through command

Miscellaneous Operations

C	change rest of line (**c$**)
D	delete rest of line (**d$**)
s	substitute chars (**cl**)
S	substitute lines (**cc**)
J	join lines
x	delete characters (**dl**)
X	delete characters before cursor (**dh**)
Y	yank lines (**yy**)

Yank and Put

Put inserts the text most recently deleted or yanked; however, if a buffer is named (using the ASCII lower-case letters **a** - **z**), the text in that buffer is put instead:

3yy	yank 3 lines
3yl	yank 3 characters
p	put back text after cursor
P	put back text before cursor
"*x***p**	put from buffer *x*
"*x***y**	yank to buffer *x*
"*x***d**	delete into buffer *x*

Undo, Redo, Retrieve

u	undo last change
U	restore current line
.	repeat last change
"*d***p**	retrieve *d*'th last delete

AUTHOR

vi and *ex* were developed by the University of California, Berkeley, California, Computer Science Division, Department of Electrical Engineering and Computer Science.

FILES

/tmp default directory where temporary work files are placed; it can be changed using the **directory** option (see the *ex(1) set* command)

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

/usr/lib/.COREterm/?/*
subset of compiled terminal description database

NOTES

Two options, although they continue to be supported, have been replaced in the documentation by options that follow the Command Syntax Standard (see *intro*(1)). A **–r** option that is not followed with an option-argument has been replaced by **–L** and **+***command* has been replaced by **–c** *command*.

SEE ALSO

ed(1), edit(1), ex(1).

WARNINGS

The encryption options are provided as a separate package only to source customers in the United States.

Tampering with entries in **/usr/lib/.COREterm/?/*** or **/usr/lib/terminfo/?/*** (e.g., changing or removing an entry) can affect programs such as *vi*(1) that expect the entry to be present and correct. In particular, removing the "dumb" terminal may cause unexpected problems.

It has been observed that some terminals exhibit problems when the user holds down arrow keys to move around the text. This problem can usually be solved by setting the ixoff flag before running *vi*. The command to do this is **stty ixoff.**

BUGS

Software tabs using **^T** work only immediately after the *autoindent*.

Left and right shifts on intelligent terminals do not make use of insert and delete character operations in the terminal.

NAME

wait – await completion of process

SYNOPSIS

wait [*n*]

DESCRIPTION

Wait for your background process whose process id is *n* and report its termination status. If *n* is omitted, all your shell's currently active background processes are waited for and the return code will be zero.

The shell itself executes *wait*, without creating a new process.

SEE ALSO

sh(1).

CAVEAT

If you get the error message **cannot fork, too many processes**, try using the *wait*(1) command to clean up your background processes. If this does not help, the system process table is probably full or you have too many active foreground processes. (There is a limit to the number of process ids associated with your login, and to the number the system can keep track of.)

BUGS

Not all the processes of a 3- or more-stage pipeline are children of the shell, thus cannot be waited for.

If *n* is not an active process id, all your shell's currently active background processes are waited for and the return code will be zero.

NAME

wall – write to all users

SYNOPSIS

/etc/wall

DESCRIPTION

wall reads its standard input until an EOF. It then sends this message to all currently logged-in users preceded by:

Broadcast Message from ...

It is used to warn all users, typically before shutting down the system.

The sender must be superuser to override any protections the users may have invoked (see *mesg*(1)).

FILES

/dev/tty*

SEE ALSO

mesg(1), write(1).

DIAGNOSTICS

"Cannot send to ..." when the open on a user's TTY file fails.

NAME

wc – word count

SYNOPSIS

wc [**–lwc**] [*names*]

DESCRIPTION

wc counts lines, words, and characters in the named files, or in the standard input if no *names* appear. It also keeps a total count for all named files. A word is a maximal string of characters delimited by spaces, tabs, or newlines.

The options **l**, **w**, and **c** may be used in any combination to specify that a subset of lines, words, and characters are to be reported. The default is **–lwc**.

When *names* are specified on the command line, they will be printed along with the counts.

NAME

what – identify SCCS files

SYNOPSIS

what [–s] *files*

DESCRIPTION

what searches the given files for all occurrences of the pattern that *get*(1) substitutes for %Z% (this is **@(#)** at this printing) and prints out what follows until the first ", >, newline, \, or **NULL** character. For example, if the C program in file **f.c** contains

```
char ident[] = "@(#)identification information";
```

and **f.c** is compiled to yield **f.o** and **a.out**, then the command:

```
what f.c f.o a.out
```

will print:

```
f.c:
        identification information
f.o:
        identification information
a.out:
        identification information
```

what is intended to be used with the command *get*(1), which automatically inserts identifying information, but it can also be used where the information is inserted manually. Only one option exists:

–s Quit after finding the first occurrence of pattern in each file.

SEE ALSO

get(1), help(1).

DIAGNOSTICS

Exit status is 0 if any matches are found, otherwise 1. Use *help*(1) for explanations.

BUGS

It is possible that an unintended occurrence of the pattern **@(#)** could be found just by chance, but this causes no harm in nearly all cases.

NAME

who – who is on the system

SYNOPSIS

who [**–uTlHqpdbrtas**] [**–n** *x*] [*file*]

who am i

who am I

DESCRIPTION

who can list the user's name, terminal line, login time, elapsed time since activity occurred on the line, and the process-ID of the command interpreter (shell) for each current system user. It examines the **/etc/utmp** file at login time to obtain its information. If *file* is given, that file (which must be in *utmp*(4) format) is examined. Usually, *file* will be **/etc/wtmp**, which contains a history of all the logins since the file was last created.

who with the **am i** or **am I** option identifies the invoking user.

The general format for output is:

name [state] line time [idle] [pid] [comment] [exit]

The *name*, *line*, and *time* information is produced by all options except **–q**; the *state* information is produced only by **–T**; the *idle* and *pid* information is produced only by **–u** and **–l**; and the *comment* and *exit* information is produced only by **–a**. The information produced for **–p**, **–d**, and **–r** is explained during the discussion of each option, below.

With options, *who* can list logins, logoffs, reboots, and changes to the system clock, as well as other processes spawned by the *init* process. These options are:

–u
lists only those users who are currently logged in. The *name* is the user's login name. The *line* is the name of the line as found in the directory **/dev**. The *time* is the time that the user logged in. The *idle* column contains the number of hours and minutes since activity last occurred on that particular line. A dot (.) indicates that the terminal has seen activity in the last minute and is therefore "current". If more than 24 hours have elapsed or the line has not been used since boot time, the entry is marked **old**. This field is useful when trying to determine if a person is working at the terminal. The *pid* is the process-ID of the user's shell. The *comment* is the comment field associated with this line as found in **/etc/inittab** (see *inittab*(4)). This can contain information about where the terminal is located, the telephone number of the dataset, type of terminal if hard-wired.

–T
is the same as the **–s** option, except that the *state* of the terminal line is printed. The *state* describes whether someone else can write to that terminal. A **+** appears if the terminal is writable by anyone; a **–** appears if it is not. **root** can write to all lines having a **+** or a **–** in the *state* field. If a bad line is encountered, a **?** is printed.

–l
lists only those lines on which the system is waiting for someone to login. The *name* field is **LOGIN** in such cases. Other fields are the same as for user entries except that the *state* field does not exist.

–H
prints column headings above the regular output.

–q
This is a quick *who*, displaying only the names and the number of users currently logged on. When this option is used, all other options are ignored.

–p
lists any other process which is currently active and has been previously spawned by *init*. The *name* field is the name of the program executed by *init* as found in **/etc/inittab**. The *state*, *line*, and *idle* fields have no meaning. The *comment* field shows the *id* field of the line from **/etc/inittab** that spawned this process. See *inittab*(4).

–d
displays all processes that have expired and not been respawned by *init*. The *exit* field appears for dead processes and contains the termination and exit values (as returned by *wait*(2)), of the dead process. This can be useful in determining why a process terminated.

–b
indicates the time and date of the last reboot.

–r
indicates the current *run-level* of the *init* process. In addition, it produces the process termination status, process id, and process exit status (see *utmp*(4)) under the *idle*, *pid*, and *comment* headings, respectively.

–t
indicates the last change to the system clock (via the *date*(1) command) by **root**. See *su*(1).

–a
processes **/etc/utmp** or the named *file* with all options turned on.

–s
is the default and lists only the *name*, *line*, and *time* fields.

–n *x*
takes a numeric argument, *x*, which specifies the number of users to display per line. *x* must be at least **1**. The –n option must be used with **–q**.

Note to the superuser: after a shutdown to the single-user state, *who* returns a prompt. The reason the prompt displays is because **/etc/utmp** is updated at login time and there is no login in single-user state, therefore, *who* cannot report accurately on this state. *who am i*, however, returns the correct information.

FILES

/etc/utmp
/etc/wtmp
/etc/inittab

SEE ALSO

date(1), login(1), mesg(1), init(1M), su(1M), wait(2), inittab(4), utmp(4).

NAME

whoami – print effective current user id

SYNOPSIS

whoami

DESCRIPTION

Whoami prints who you are. It works even if you are su'd, while 'who am i' does not since it uses /etc/utmp.

FILES

/etc/passwd Name data base

SEE ALSO

who (1).

NAME

write – write to another user

SYNOPSIS

write *user* [*line*]

DESCRIPTION

write copies lines from your terminal to that of another user. When first called, it sends the message:

Message from *yourname* **(tty??)** [*date*]...

to the person you want to talk to. When it has successfully completed the connection, it also sends two bells to your own terminal to indicate that what you are typing is being sent.

The recipient of the message should write back at this point. Communication continues until an end of file is read from the terminal, an interrupt is sent, or the recipient has executed "mesg n". At that point *write* writes **EOT** on the other terminal and exits.

If you want to write to a user who is logged in more than once, the *line* argument may be used to indicate which line or terminal to send to (e.g., **tty00**); otherwise, the first writable instance of the user found in **/etc/utmp** is assumed and the following message posted:

user is logged on more than one place.
You are connected to "*terminal*".
Other locations are:
terminal

Permission to write may be denied or granted by use of the *mesg*(1) command. Writing to others is normally allowed by default. Certain commands, e.g., *pr*(1), disallow messages to prevent interference with their output. However, if the user has superuser permissions, messages can be forced onto a write-inhibited terminal.

If the character **!** is found at the beginning of a line, *write* calls the shell to execute the rest of the line as a command.

The following protocol is suggested for using *write*: when you first *write* to another user, wait for them to *write* back before starting to send. Each person should end a message with a distinctive signal (i.e., **(o)** for "over") so that the other person knows when to reply. The signal **(oo)** (for "over and out") is suggested when conversation is to be terminated.

FILES

/etc/utmp	to find user
/bin/sh	to execute **!**

SEE ALSO

mail(1), mesg(1), pr(1), sh(1), who(1)

DIAGNOSTICS

"*user is not logged on*"
if the person you are trying to *write* to is not logged on.

"*Permission denied*"
if the person you are trying to *write* to denies that permission (with *mesg*).

"*Warning: cannot respond, set mesg -y*"
if your terminal is set to *mesg n* and the recipient cannot respond to you.

"*Can no longer write to user*"
if the recipient has denied permission (*mesg n*) after you had started writing.

NAME

xargs – construct argument list(s) and execute command

SYNOPSIS

xargs [flags] [command [initial-arguments]]

DESCRIPTION

xargs combines the fixed ***initial-arguments*** with arguments read from standard input to execute the specified ***command*** one or more times. The number of arguments read for each ***command*** invocation and the manner in which they are combined are determined by the flags specified.

command, which may be a shell file, is searched for, using one's **$PATH**. If ***command*** is omitted, **/bin/echo** is used.

Arguments read in from standard input are defined to be contiguous strings of characters delimited by one or more blanks, tabs, or newlines; empty lines are always discarded. Blanks and tabs may be embedded as part of an argument if escaped or quoted. Characters enclosed in quotes (single or double) are taken literally, and the delimiting quotes are removed. Outside of quoted strings a backslash (\) will escape the next character.

Each argument list is constructed starting with the ***initial-arguments***, followed by some number of arguments read from standard input (Exception: see **–i** flag). Flags **–i**, **–l**, and **–n** determine how arguments are selected for each command invocation. When none of these flags are coded, the ***initial-arguments*** are followed by arguments read continuously from standard input until an internal buffer is full, and then ***command*** is executed with the accumulated args. This process is repeated until there are no more args. When there are flag conflicts (e.g., **–l** vs. **–n**), the last flag has precedence. *Flag* values are:

–l*number* — ***command*** is executed for each non-empty ***number*** lines of arguments from standard input. The last invocation of ***command*** will be with fewer lines of arguments if fewer than ***number*** remain. A line is considered to end with the first newline ***unless*** the last character of the line is a blank or a tab; a trailing blank/tab signals continuation through the next non-empty line. If ***number*** is omitted, 1 is assumed. Option **–x** is forced.

–i*replstr* — Insert mode: ***command*** is executed for each line from standard input, taking the entire line as a single arg, inserting it in ***initial-arguments*** for each occurrence of ***replstr***. A maximum of 5 arguments in ***initial-arguments*** may each contain one or more instances of ***replstr***. Blanks and tabs at the beginning of each line are thrown away. Constructed arguments may not grow larger than 255 characters, and option **–x** is also forced. **{}** is assumed for ***replstr*** if not specified.

–n*number* — Execute ***command*** using as many standard input arguments as possible, up to ***number*** arguments maximum. Fewer arguments will be used if their total size is greater than ***size*** characters, and for the last invocation if there are fewer than ***number*** arguments remaining. If option **–x** is also coded, each ***number*** arguments must fit in the ***size*** limitation, else ***xargs*** terminates execution.

–t — Trace mode: The ***command*** and each constructed argument list are echoed to file descriptor 2 just prior to their execution.

–p — Prompt mode: The user is asked whether to execute ***command*** each invocation. Trace mode (**–t**) is turned on to print the command instance to be executed, followed by a **?...** prompt. A reply of **y** (optionally followed by anything) will execute the command; anything else, including just a carriage return, skips that particular invocation of ***command***.

−x Causes *xargs* to terminate if any argument list would be greater than *size* characters; **−x** is forced by the options **−i** and **−l**. When neither of the options **−i**, **−l**, or **−n** are coded, the total length of all arguments must be within the *size* limit.

−s*size* The maximum total size of each argument list is set to *size* characters; *size* must be a positive integer less than or equal to 3996. If **−s** is not coded, 1000 is taken as the default. Note that the character count for *size* includes one extra character for each argument and the count of characters in the command name.

−e*eofstr* *eofstr* is taken as the logical end-of-file string. Underbar (_) is assumed for the logical **EOF** string if **−e** is not coded. The value **−e** with no *eofstr* coded turns off the logical **EOF** string capability (underbar is taken literally). *xargs* reads standard input until either end-of-file or the logical **EOF** string is encountered.

−P*lim* Permits *xargs* to spin off its children in parallel. The number of outstanding children is limited by *lim* or by the system load, if no limit was specified.

−C Collate option. Valid only if **−P** is also specified. Causes the output of children running in parallel not to be intermixed.

Xargs will terminate if either it receives a return code of **−1** from, or if it cannot execute, *command*. When *command* is a shell program, it should explicitly *exit* (see *sh*(1)) with an appropriate value to avoid accidentally returning with **−1**.

CAVEATS

There may be a substantial delay when the collate option is used, as *xargs* must spool the standard error and standard output of each child to temporary files, and not print those files until the child exits.

EXAMPLES

The following will move all files from directory $1 to directory $2, and echo each move command just before doing it:

```
ls $1 | xargs −i −t mv $1/{} $2/{}
```

The following will combine the output of the parenthesized commands onto one line, which is then echoed to the end of file *log*:

```
(logname; date; echo $0 $*) | xargs >> log
```

The user is asked which files in the current directory are to be archived and archives them into *arch* (1.) one at a time, or (2.) many at a time.

```
1. ls | xargs −p −l ar r arch
2. ls | xargs −p −l | xargs ar r arch
```

The following will execute *diff*(1) with successive pairs of arguments originally typed as shell arguments:

```
echo $* | xargs −n2 diff
```

SEE ALSO

sh(1).

NAME

yacc – yet another compiler-compiler

SYNOPSIS

yacc [**–vdlt**] *grammar*

DESCRIPTION

The *yacc* command converts a context-free grammar into a set of tables for a simple automaton which executes an **LR**(1) parsing algorithm. The grammar may be ambiguous; specified precedence rules are used to break ambiguities.

The output file, **y.tab.c**, must be compiled by the C compiler to produce a program *yyparse*. This program must be loaded with the lexical analyzer program, *yylex*, as well as *main* and *yyerror*, an error handling routine. These routines must be supplied by the user; *lex*(1) is useful for creating lexical analyzers usable by *yacc*.

If the **–v** flag is given, the file **y.output** is prepared, which contains a description of the parsing tables and a report on conflicts generated by ambiguities in the grammar.

If the **–d** flag is used, the file **y.tab.h** is generated with the **#define** statements that associate the *yacc*-assigned "token codes" with the user-declared "token names". This allows source files other than **y.tab.c** to access the token codes.

If the **–l** flag is given, the code produced in **y.tab.c** will *not* contain any **#line** constructs. This should only be used after the grammar and the associated actions are fully debugged.

Runtime debugging code is always generated in **y.tab.c** under conditional compilation control. By default, this code is not included when **y.tab.c** is compiled. However, when *yacc*'s **–t** option is used, this debugging code will be compiled by default. Independent of whether the **–t** option was used, the runtime debugging code is under the control of **YYDEBUG**, a preprocessor symbol. If **YYDEBUG** has a non-zero value, then the debugging code is included. If its value is zero, then the code will not be included. The size and execution time of a program produced without the runtime debugging code will be smaller and slightly faster.

FILES

y.output	
y.tab.c	
y.tab.h	defines for token names
yacc.tmp,	
yacc.debug, yacc.acts	temporary files
/usr/lib/yaccpar	parser prototype for C programs

SEE ALSO

lex(1).

DIAGNOSTICS

The number of reduce-reduce and shift-reduce conflicts is reported on the standard error output; a more detailed report is found in the **y.output** file. Similarly, if some rules are not reachable from the start symbol, this is also reported.

CAVEAT

Because file names are fixed, at most one *yacc* process can be active in a given directory at a given time.

NAME

ypcat – print values in a NIS database

SYNOPSIS

ypcat [**–k**] [**–t**] [**–d** *domainname*] *mname*
ypcat **–x**

DESCRIPTION

ypcat prints out values in NIS map specified by *mname*, which may be either a *mapname* or a map "nickname." Since *ypcat* uses the NIS network services, no NIS server is specified.

For the REAL/IX implementation of ONC, map information from the NIS master is rapidly incorporated into local system files. Despite this, there may be occasions when you wish to check on the contents of a map. For example, to look at the system group map, which will be held on the NIS master server and any slave servers,

ypcat group

The maps typically have both a long name, an abbreviated name to comply with the 14 character filename limitation of the REAL/IX Operating System, and a nickname. Here is a list of abbreviated mapnames, full mapnames, and corresponding system files:

long name	short name	system file
passwd.byname	passwd.nam	/etc/passwd
group.byname	group.nam	/etc/group
hosts.byaddr	hosts.adr	/etc/hosts
networks.byaddr	netwks.adr	/etc/networks
services.byname	srvcs.nam	/etc/services
protocols.bynumber	proto.no	/etc/protocols
netgroup	netgroup	/etc/netgroup

See *ypfiles*(4N) and *ypserv*(1M) for an overview of NIS.

FLAG OPTIONS

–k

Display the keys for those maps in which the values are null or the key is not part of the value. (None of the maps derived from files that have an ASCII version in **/etc** fall into this class.)

–t

Inhibit translation of *mname* to *mapname*. For example, **ypcat –t passwd** will fail because there is no map named **passwd**, whereas **ypcat passwd** will be translated to **ypcat passwd.byname**.

–d

Specify a domain other that the default domain. The default domain is returned by *domainname*.

–x

Display the map nickname table. This lists the nicknames (*mnames*) the command knows of, and indicates the *mapname* associated with each nickname.

FILES

/usr/bin/ypcat

SEE ALSO

domainname(1N), *ypmatch*(1N), *ypserv*(1M), *ypfiles*(4N).
ONC User's Guide.

NAME

ypmatch – print the value of one or more keys from a NIS map

SYNOPSIS

ypmatch [-d*domain*] [-k] [-t] *key* ... *mname*

ypmatch -x

DESCRIPTION

ypmatch prints the values associated with one or more keys from the NIS map (database) specified by a *mname*, which may be either a mapname or a map nickname.

Multiple keys can be specified; the same map will be searched for all. The keys must be exact values insofar as capitalization and length are concerned. No pattern matching is available. If a key is not matched, a diagnostic message is produced.

FLAG OPTIONS

–d

Specify a domain other than the default domain.

–k

Before printing the value of a key, print the key itself, followed by a colon (:). This is useful only if the keys are not duplicated in the values, or you've specified so many keys that the output could be confusing.

–t

Inhibit translation of nickname to mapname. For example, **ypmatch -t zippy passwd** fails because there is no map named **passwd**, while **ypmatch zippy passwd** is translated to **ypmatch zippy passwd.byname**.

–x

Display the map nickname table. This lists the nicknames (*mnames*) the command knows of, and indicates the *mapname* associated with each nickname.

FILES

/usr/bin/ypmatch

SEE ALSO

ypcat(1N), ypfiles(4N)

NAME

yppasswd – change login password in NIS

SYNOPSIS

yppasswd [*name*]

DESCRIPTION

yppasswd changes (or installs) a password associated with the user *name* (your own name by default) in NIS. The NIS password may be different from the one on your own machine.

yppasswd prompts for the old NIS password and then for the new one. The caller must supply both. The new password must be typed twice, to forestall mistakes.

New passwords must be at least four characters long if they use a sufficiently rich alphabet and at least six characters long if monocase. These rules are relaxed if you are insistent enough.

Only the owner of the name or the superuser may change a password; in either case you must prove you know the old password.

FILES

/usr/bin/yppasswd

SEE ALSO

passwd(1), yppasswdd(1M), ypfiles(4N)

BUGS

The update protocol passes all the information to the server in one rpc call, without ever looking at it. Thus, if you type in your old password incorrectly, you will not be notified until after you have entered and verified your new password.

NAME

ypwhich – which host is the NIS server or map master?

SYNOPSIS

ypwhich [**-d** *domain*] [**-V1** | **-V2**] [*hostname*]
ypwhich [**-t** *mapname*] [**-d** *domain*] **-m** *mname*
ypwhich -x

DESCRIPTION

ypwhich tells which NIS server supplies NIS services to a NIS client, or which is the master for a map. If invoked without arguments, it gives the NIS server for the local machine. If *hostname* is specified, that machine is queried to find out which NIS master it is using.

Refer to *ypfiles*(4N) and *ypserv*(1M) for an overview of the NIS.

OPTIONS

–d
Use *domain* instead of the default domain.

–V1
Which server is serving v.1 NIS protocol-speaking client processes?

–V2
Which server is serving v.2 NIS protocol client processes?

If neither version is specified, *ypwhich* attempts to locate the server that supplies the (current) v.2 services. If there is no v.2 server currently bound, *ypwhich* then attempts to locate the server supplying the v.1 services. Since NIS servers and NIS clients are both backward compatible, the user need seldom be concerned about which version is currently in use.

–t *mapname*
Inhibit nickname translation; useful if there is a mapname identical to a nickname.

–m
Find the master NIS server for a map. No *hostname* can be specified with **–m**. *mname* can be a mapname, or a nickname for a map.

–x
Display the map nickname table. This lists the nicknames (*mnames*) the command knows of, and indicates the *mapname* associated with each nickname.

SEE ALSO

ypfiles(4N), rpcinfo(1M), ypset(1M), ypserv(1M)