Dansk Data Elektronik A/S

SUPERMAX BASIC UTILITIES

System V Reference Manual

Section 1, Essential Utilities

Release 3.1, Version 2.6

Permuted Index

This is a permuted index of all the articles found in the *Supermax System V, Reference Manuals*. Each line in the index consists of three fields:

1) A possibly empty 'head' field.

2) A 'key' field, followed by a number of periods.

3) A 'reference' field.

The index is sorted alphabetically by the key field.

Most lines in the index are taken directly from the 'NAME' section of each article. Each word of that short description of the article is used as a key in the key field.

The head field contains the part of the description preceding the key.

The reference field tells the reader where to find the article.

As an example consider the article about the *ls* in Section 1 of the Reference Manuals. The purpose of *ls* is to 'list contents of directory'. Therefore *ls* may be found in the permuted index in four places, namely under *ls*, under *list*, under *contents*, and under *directory*, thus:

ls: list	contents of directory.................	ls(1)
ls: list of contents	directory	ls(1)
ls:	list contents of directory...........	ls(1)
	ls: list contents of directory......	ls(1)

The most common words, such as 'a', 'the', 'of', etc., are not used as keys.

This page is intentionally left blank

Permuted Index

l3tol, ltol3: convert between	3-byte integers and long integers	l3tol(3C)
diff3	3-way differential file comparison	diff3(1)
	a64l, l64a: convert between	a64l(3C)
	abort: generate an IOT fault	abort(3C)
	abs: integer absolute value	abs(3C)
abs: integer	absolute value	abs(3C)
ceil, fmod, fabs: floor, ceiling, remainder	absolute value functions	floor(3M)
	accept, reject: allow/prevent LP requests	accept(1M)
touch: update	access and modification times of a file	touch(1)
utime: set file	access and modification times	utime(2)
	access: determine accessibility of a file	access(2)
par_chm: change	access rights to partition	par_chm(2)
ldfcn: common object file	access routines	ldfcn(4)
dcopy: copy file systems for optimal	access time	dcopy(1M)
locking: provide exclusive	access to a byte range	locking(2)
getutent, getutid, getutline	access utmp file entry	getut(3C)
pututline, setutent:	access utmp file entry	getut(3C)
endutent, utmpname:	access utmp file entry	getut(3C)
access: determine	accessibility of a file	access(2)
acct: enable or disable process	accounting	acct(2)
acctcon1, acctcon2: connect-time	accounting	acctcon(1M)
ckpacct, dodisk: shell procedure for	accounting	acctsh(1M)
monacct, nulladm: shell procedure for	accounting	acctsh(1M)
prdaily: shell procedure for	accounting	acctsh(1M)
prctmp, prtacct: shell procedures for	accounting	acctsh(1M)
startup, turnacct: shell procedures for	accounting	acctsh(1M)
acctdusg, accton, acctwtmp: overview of	accounting and misc. accounting	acct(1M)
overview of accounting and misc.	accounting commands	acct(1M)
diskusg: generate disk	accounting data by user ID	diskusg(1M)
acct: per-process	accounting file format	acct(4)
acctcom: search and print process	accounting files	acctcom(1)
acctmerg: merge or add total	accounting files	acctmerg(1M)
acctcms: summary from per-process	accounting records	acctcms(1M)
fwtmp, wtmpfix: manipulate connect	accounting records	fwtmp(1M)
runacct: run daily	accounting	runacct(1M)

	acct: enable/disable process accounting	acct(2)
	acct: pre-process accounting file format	acct(4)
	acctcms: summary from per-process	acctcms(1M)
	acctcom: search and print process acct	acctcom(1)
	acctcon1, acctcon2: connect-time accnt	acctcon(1M)
acctcon1	acctcon2: connect-time accounting	acctcon(1M)
	acctdis, acctdusg, accton: overview of	acct(1M)
	acctwtmp: overview of	acct(1M)
acctdis	acctdusg, accton, acctwtmp: overview of	acct(1M)
	acctmerg: merge/add total accounting files	acctmerg(1M)
acctdis, acctdusg	accton, acctwtmp: overview of acct and	acct(1M)
	acctprc1, acctprc2: process accounting	acctprc(1M)
acctprc1	acctprc2: process accounting	acctprc(1M)
acctdis, acctdusg, accton	acctwtmp: overview of acct and	acct(1M)
sin, cos, tan, asin	acos, atan, atan2: trigonometric functions	trig(3M)
killall: kill all	active processes	killall(1M)
time a command:	activity	timex(1)
report process data and system	activity	timex(1)
acctmerg: merge or	add total accounting files	acctmerg(1M)
putenv: change or	add value to environment	putenv(3C)
set_parm: define	additional system call params	set_parm(2)
udamind:	administrative control	uadmin(1M)
uadmin:	administrative control	uadmin(2)
alarm: set a process	alarm clock	alarm(2)
	alarm: set a process alarm clock	alarm(2)
aliens: the	alien invaders attack the earth	aliens(6)
	aliens: the alien invaders attack the earth	aliens(6)
killall: kill	all active processes	killall(1M)
wall: write to	all users	wall(1)
brk: change data segment space	allocation	brk(2)
sbrk: change data segment space	allocation	brk(2)
malloc, free: main memory	allocator	malloc(3C)
realloc, calloc: main memory	allocator	malloc(3C)
calloc, mallopt: fast main memory	allocator	malloc(3X)
mallinfo: fast main memory	allocator	malloc(3X)
accept, reject:	allow or prevent LP requests	accept(1M)
	amsgop: asynchronous message operations	amsgop(2)

yacc: yet	another compiler-compiler	yacc(1)
su: become super-user or	another user	su(1M)
write: write to	another user	write(1)
	a.out: assembler and link editor output	a.out(4)
	ar: archive and library maintainer	ar(1)
	ar: common archive file format	ar(4)
bc:	arbitrary precision arithmetic language	bc(1)
ar:	archive and library maintainer	ar(1)
cpio: format of cpio	archive	cpio(4)
ar: common	archive file format	ar(4)
read the archive header of a member	archive file	ldahread(3X)
ldahread: read the	archive header of a member	ldahread(3X)
tar: tape file	archiver	tar(1)
cpio: copy file	archives in and out	cpio(1)
	aread: asynchronous read	aread(2)
varargs: handle variable	argument list	varargs(5)
vsprintf: print formatted output	argument list	vprintf(3S)
xargs: construct	argument lists and execute cmd	xargs(1)
getopt: get option letter from	argument vector	getopt(3C)
expr: evaluate	arguments as an expression	expr(1)
echo: echo	arguments	echo(1)
bc: arbitrary precision	arithmetic language	bc(1)
	arithmetic: provide drill in number facts	arithmetic(6)
	as, as20: assembler	as(1)
as	as20: assembler	as(1)
l64a: conv.btw.long integer and base-64	ASCII	a64l(3C)
ascii: map of	ASCII character set	ascii(5)
	ascii: map of ASCII character set	ascii(5)
ctime, localtime, gmtime	asctime, tzset: convert date and time	ctime(3C)
sin, cos, tan	asin, acos, atan, atan2: trigonometric	trig(3M)
a.out: common	assembler and link editor output	a.out(4)
as, as20:	assembler	as(1)
	assert: verify program assertion	assert(3X)
assert: verify program	assertion	assert(3X)
setbuf, setvbuf:	assign buffering to a stream	setbuf(3S)
	asuspend: asynchronous suspend	asuspend(2)
amsgop:	asynchronous message operations	amsgop(2)

aread:	asynchronous read	aread(2)
asuspend:	asynchronous suspend	asuspend(2)
awrite:	asynchronous write	awrite(2)
	at, batch: execute cmd at a later time	at(1)
sin, cos, tan, asin, acos	atan, atan2: trigonometric functions	trig(3M)
sin, cos, tan, asin, acos, atan	atan2: trigonometric functions	trig(3M)
strtod	atof: conv. string to dbl-precision number	strtod(3C)
strtol, atol	atoi: convert string to integer	strtol(3C)
strtol	atol, atoi: convert string to integer	strtol(3C)
par_att:	attach a memory partition	par_att(2)
alien: the alien invaders	attack the earth	aliens(6)
wait:	await completion of process	wait(1)
	awk: pattern scan and process language	awk(1)
	awrite: asynchronous write	awrite(2)
ungetc: push character	back into input stream	ungetc(3S)
	back: the game of backgammon	back(6)
back: the game of	backgammon	back(6)
finc: fast incremental	backup	finc(1M)
frec: recover files from a	backup tape	frec(1M)
	banner: make posters	banner(1)
terminfo: terminal capability data	base	terminfo(4)
a64l, l64a: conv.btw.long integer and	base-64 ASCII	a64l(3C)
vedit: scrn.oriented (visual) display ed	based on ex	vi(1)
	basename,dirname: deliver portions of	basename(1)
at	batch: execute cmd at a later time	at(1)
	bc: arbitrary precision arithmetic language	bc(1)
brc	bcheckrc: system initialization procedures	brf(1M)
	bdiff: big diff	bdiff(1)
cb: C program	beautifier	cb(1)
su:	become super-user or another user	su(1M)
j0, j1, jn, y0, y1, yn:	Bessel functions	bessel(3M)
l3tol, ltol3: convert	btw. 3-byte integers and long integers	l3tol(3C)
a64l, l64a: convert	btw. long integer and base-64 ASCII	a64l(3C)
bdiff:	big diff	bdiff(1)
bsf:	big file scanner	bfs(1)
fread, fwrite:	binary input and output	fread(3S)
bsearch:	binary search a sorted table	bsearch(3C)

tsearch, tfind, tdelete, twalk: manage	binary search trees	tsearch(3C)
	bj: the game of black jack	bj(6)
bj: the game of	black jack	bj(6)
sum: print checksum and	block count of a file	sum(1)
sync: update the super	block	sync(1M)
df: report number of free disk	blocks and i-nodes	df(1)
	brc, bcheckrc: system init. procedures	brf(1M)
	brk, sbrk: chg data segment space alloc	brk(2)
	bsearch: binary search a sorted table	bsearch(3C)
	bsf: big file scanner	bfs(1)
stdio: standard	buffered input-output package	stdio(3S)
streamdrv: copy with	buffering	streamdrv(1)
setbuf, setvbuf: assign	buffering to a stream	setbuf(3S)
mknod:	build special file	mknod(1M)
locking: provide excl.access to a	byte range	locking(2)
size: print section size in	bytes of common object files	size(1)
swab: swap	bytes	swab(3C)
cc:	C compiler	cc(1)
cflow: generate	C flowgraph	cflow(1)
cpp: the	C language preprocessor	cpp(1)
cb:	c program beautifier	cb(1)
lint: a	C program checker	lint(1)
cxref: generate	C program cross-reference	cxref(1)
ctrace:	C program debugger	ctrace(1)
clist: list	C programs	clist(1)
list: procedure	C source listing from a common obj.file	list(1)
sync: write disk	cache to disk	sync(2)
	cal: print calendar	cal(1)
dc: desk	calculator	dc(1)
cal: print	calendar	cal(1)
	calendar: reminder service	calendar(1)
set_parm: define additional system	call parameters	set_parm(2)
stat: data returned by stat system	call	stat(5)
malloc, free, realloc	calloc: main memory allocator	malloc(3C)
malloc, free, realloc	calloc, mallopt, mallinfo: fast main	malloc(3X)
lp, cancel: send or	cancel requests to LP line printer	lp(1)
lp	cancel: send/cancel req. to LP line printer	lp(1)

10

	clock: report CPU time used	clock(3C)
ldclose, ldaclose:	close a common object file	ldclose(3X)
close:	close a file-descriptor	close(2)
	close: close a file-descriptor	close(2)
fclose, fflush:	close or flush a stream	fclose(3S)
	clri: clear i-node	clri(1M)
	cmp: compare two files	cmp(1)
seek to line-no of a section	common object file	ldlseek(3X)
	comm: select/reject lines comm to two	comm(1)
nice: run a	command at low priority	nice(1)
chroot: chg root directory for a	command	chroot(1M)
env: set environment for	command execution	env(1)
nohup: run a	command immune to hangups and quits	nohup(1)
getopt: parse	command options	getopt(1)
sh, rsh: shell, the std.and restricted	command programming language	sh(1)
timex: time a	command: report process data and system	timex(1)
acctcms:	command summ.from pre-proc.accounting	acctcms(1M)
system: issue a shell	command	system(3S)
test: condition evaluation	command	test(1)
time: time a	command	time(1)
xargs: construct arg.lists and execute	command	xargs(1)
of accounting and misc.accounting	commands	acct(1M)
at, batch: execute	commands at a later time	at(1)
install: install	commands	install(1M)
rc2: run	cmd's performed for multi-user envimnt.	rc2(1M)
rc0: run	cmd's performed to stop opr.system	rc0(1M)
mcs: manipulate the object file	comment section	mcs(1)
ar:	common archive file format	ar(4)
a.out:	common assmbl.and link editor output	a.out(4)
ldfcn:	common object file access routines	ldfcn(4)
cprs: compress a	common object file	cprs(1)
ldopen, ldsopen: open a	common object file for reading	ldopen(3X)
lditem: manip.line-no entries for a	common object file function	ldlread(3X)
ldclose, ldaclose: close a	common object file	ldclose(3X)
ldfhread: read file header for a	common object file	ldfhread(3X)
ldohseek: seek opt.file header for a	common object file	ldohseek(3X)
seek reloc. entries of a section for a	common object file	ldrseek(3X)

13

test:	condition evaluation command	test(1)
	config: print system configuration	config(1M)
chhw: change system	configuration	chhw(1M)
config: print system	configuration	config(1M)
lpadmin:	configure the LP spooling system	lpadmin(1M)
fwtmp, wtmpfix: manipulate	connect accounting records	fwtmp(1M)
dial: establish outgoing terminal line	connection	dial(3C)
acctcon1, acctcon2:	connect-time accounting	acctcon(1M)
fsck, fsck512, dfsck: file system	consistency check and interactive repair	fsck(1M)
math: math functions and	constants	math(5)
mkfs, mkfs512:	construct a file system	mkfs(1M)
xargs:	construct arg. lists and execute cmd	xargs(1)
deroff: remove nroff, troff, tbl, and eqn	constructs	deroff(1)
ls: list	contents of directory	ls(1)
csplit:	context split	csplit(1)
fcntl: file	control	fcntl(2)
init, telinit: process	control initialization	init(1M)
ioctl: device	control	ioctl(2)
msgctl: message	control operations	msgctl(2)
semctl: semaphore	control operations	semctl(2)
shmctl: shared memory	control operations	shmctl(2)
fcntl: file	control operations	fcntl(5)
udamind: administrative	control	uadmin(1M)
uadmin: administrative	control	uadmin(2)
tty:	controlling terminal	tty(7)
term:	conventional names for terminals	term(5)
units:	conversion program	units(1)
dd:	convert and copy a file	dd(1)
l3tol, ltol3:	convert btw. 3-byte integer and long	l3tol(3C)
a64l, l64a:	conv.btw. long integer and base-64 ASCII	a64l(3C)
ctime, localtime, gmtime, asctime, tzset:	convert date and time	ctime(3C)
ecvt, fcvt, gcvt:	conv. floating-point number to string	ecvt(3C)
scanf, fscanf, sscanf:	convert formatted input	scanf(3S)
strtod, atof:	convert string to dbl-precision number	strtod(3C)
strtol, atol, atoi:	convert string to integer	strtol(3C)
dd: convert and	copy a file	dd(1)
cpio:	copy file archives in and out	cpio(1)

dcopy:	copy file systems for opt. access time	dcopy(1M)
cp, ln, mv:	copy, link or move files	cp(1)
make literal:	copy of file system	volcopy(1M)
streamdrv:	copy with buffering	streamdrv(1)
sin:	cos, tan, asin, acos, atan, atan2	trig(3M)
sinh	cosh, tanh: hyperbolic functions	sinh(3M)
sumdir: sum and	count characters in the file in given	sumdir(1)
sum: print checksum and block	count of a file	sum(1)
wc:	word count	wc(1)
millisec: get millisecond	counter	millisec(2)
	cp, ln, mv: copy, link or move files	cp(1)
cpio: format of	cpio archive	cpio(4)
	cpio: copy file archives in and out	cpio(1)
	cpio: format of cpio archive	cpio(4)
	cpp: the C language pre-processor	cpp(1)
	cprs: compress a common object file	cprs(1)
clock: report	CPU time used	clock(3C
craps: the game of	craps	craps(6)
	craps: the game of craps	craps(6)
crash: provoke system	crash	crash(1M)
	crash: provoke system crash	crash(1M)
	creat: create new file or rewrite existing	creat(2)
par_cre:	create a memory partition	par_cre(2)
tmpnam, tempnam:	create a name for a temporary file	tmpnam(3S)
creat:	create a new file or rewrite an existing	creat(2)
fork:	create a new process	fork(2)
tmpfile:	create a temporary file	tmpfile(3S)
pipe:	create an inter-process channel	pipe(2)
umask: set and get file	creation mask	umask(2)
	cref: make cross-reference listing	cref(1)
	cron: clock daemons	cron(1M)
crontab: user	crontab file	crontab(1)
	crontab: user crontab file	crontab(1)
cxref: generate C program	cross-reference	cxref(1)
cref: make	cross-reference listing	cref(1)
curses:	CRT screen handling and opt. package	curses(3X)
pg: file perusal filter for	CRT's	pg(1)

	crypt, setkey, encrypt: generate hashing	crypt(3C)
	csh: a shell with C-like syntax	csh(1)
	csh: C-shell	csh(1)
csh:	C-shell	csh(1)
	csplit: contest split	csplit(1)
	ctermid: generate file name for terminal	ctermid(3S)
	ctime, localtime, gmtime, asctime, tzset	ctime(3C)
	ctrace: C program debugger	ctrace(1)
uname: get and set name of	current operating system	uname(2)
print name of	current UNIX system	uname(1)
ttyslot: find slot in utmp file of	current user	ttyslot(3C)
getcwd: get pathname of	current working directory	getcwd(3C)
	curses: CRT screen handl.and optimization.	curses(3X)
	cuserid: get char. login name of user	cuserid(3S)
	cut: cut out selected fields of each line	cut(1)
cut:	cut out selected fields of each line of a file	cut(1)
	cxref: generate C program cross-reference	cxref(1)
cron: clock	daemon	cron(1M)
runacct: run	daily accounting	runacct(1M)
timex:time a command:report process	data and system activity	timex(1)
terminfo: terminal capability	data base	terminfo(4)
diskusg: generate disk accounting	data by user ID	diskusg(1M)
fedit, flook: inspect and edit	data file or name partitions	fedit(1)
plock: lock process, text, or	data in memory	plock(2)
prof: display profile	data	prof(1)
stat:	data returned by stat system call	stat(5)
brk, sbrk: change	data segment space allocation	brk(2)
type: primitive system	data types	types(5)
join: relational	database operator	join(1)
tput: init.a terminal or query terminfo	database	tput(1)
localtime, gmtime, asctime, tzset: convert	date and time	ctime(3C)
date: print and set the	date	date(1)
	date: print and set the	date(1)
	dc: desk calculator	dc(1)
	dcopy: copy file systems for opt. access	dcopy(1M)
	dd: convert and copy a file	dd(1)
ctrace: C program	debugger	ctrace(1)

fsdb: file system	debugger	fsdb(1M)
sdb: symbolic	debugger	sdb(1)
timezone: set	default system time zone	timezone(4)
set_parm:	define additional system call param.	set_parm(2)
par_del:	delete a named partition	par_del(2)
basename, dirname:	deliver portions of path names	basename(1)
tail:	deliver the last part of a file	tail(1)
mesg: permit or	deny messages	mesg(1)
	deroff: remove nroff, troff, tbl and eqn	deroff(1)
dc:	desk calculator	dc(1)
par_det:	detach a memory partition	par_det(2)
access:	determine accessibility of a file	access(2)
file:	determine file type	file(1)
ioctl:	device control	ioctl(2)
error: the Operating system error	device	error(7)
devnm:	device name	devnm(1M)
null: the null	device	null(7)
gendev: generate	device numbers	gendev(1M)
cioc: the CIOC	devices	cioc(7)
kmem: the kernel memory	devices	kmem(7)
	devnm: device name	devnm(1M)
	df: report number of free disk blocks and	df(1)
fsck, fsck512	dfsck: file system consistency check and	fsck(1M)
	dial: establish an out-going terminal line	dial(3C)
bdiff: big	diff	bdiff(1)
	diff: differential file comparator	diff(1)
	diff3: 3-way differential file comparison	diff3(1)
sdiff: side-by-side	difference program	sdiff(1)
diff:	differential file comparator	diff(1)
diff3: 3-way	differential file comparison	diff3(1)
	dir: format of directories	dir(4)
	dircmp: directory comparison	dircmp(1)
dir: format of	directories	dir(4)
link, unlink: link, unlink files and	directories	link(1M)
rm, rmdir: remove files or	directories	rm(1)
sum and count char.in files in given	directories	sumdir(1)
cd: change working	directory	cd(1)

chdir: change working	directory	chdir(2)
chroot: change root	directory	chroot(2)
dircmp:	directory comparison	dircmp(1)
unlink: remove a	directory entry	unlink(2)
chroot: change root	directory for a command	chroot(1M)
getcwd: get path-name of curr.working	directory	getcwd(3C)
ls: list contents of	directory	ls(1)
mkdir: make a	directory	mkdir(1)
mvdir: move a	directory	mvdir(1M)
pwd: print working	directory name	pwd(1)
mknod: make a	directory, or special or ordinary file	mknod(2)
basename:	dirname: deliver portion of path names	basename(2)
	dis, dis20: disassembler	dis(1)
dis	dis20: disassembler	dis(1)
enable	disable: enable or disable LP printers	enable(1)
enable, disable: enable or	disable LP printers	enable(1)
acct: enable or	disable process accounting	acct(2)
dis, dis20:	disassembler	dis(1)
set term.type, modes, speed, and line	discipline	getty(1M)
diskusg: generate	disk accounting data by user ID	diskusg(1M)
df: report number of free	disk blocks and i-nodes	df(1)
sync: write	disk cache to disk	sync(2)
	disk: disks	disk(7)
setdioc: display or set	disk operation modes	setdioc(1M)
chlds: change logical	disk size	chlds(1M)
dsize: display	disk size	dsize(1)
sync: write disk cache to	disk	sync(2)
du: summarize	disk usage	du(1)
disk:	disks	disk(7)
	diskusg: generate disk accnt.data by user	diskusg(1M)
dsize:	display disk size	dsize(1)
vi, view, vedit: scr.oriented (visual)	display editor based on ex	vi(1)
sysvers:	display operating system versions	sysvers(1M)
setdioc:	display or set disk operating modes	setdioc(1M)
prof:	display profile data	prof(1)
led: flash hyphens in MCU	display	led(1M)
hypot: Euclidean	distance function	hypot(3M)

drand48, erand48: generate uniformly	distributed pseudo-random numbers	drand48(3C)
lrand48: generate uniformly	distributed pseudo-random numbers	drand48(3C)
nrand48, mrand48: generate uniformly	distributed pseudo-random numbers	drand48(3C)
jrand48: generate uniformly	distributed pseudo-random numbers	drand48(3C)
signal: specify what to	do upon receipt of a signal	signal(2)
chargefee, ckpacct	dodisk: shell proced. for accounting	acctsh(1M)
whodo: who is	doing what	whodo(1M)
hack: exploring the dungeon of	doom	hack(6)
strtod, atof: convert string to	double-precision number	strtod(3C)
shutdown: shut	down system, change system state	shutdown(1M)
	drand48, erand48, lrand48: generate unif.	drand48(3C)
arithmetic: provide	drill in number facts	arithmetic(6)
	dsh: shell with history facility	dsh(1)
	dsize: display disk size	dsize(1)
	du: summarize disk usage	du(1)
	dump: dump selected part of object file	dump(1)
od: octal	dump	od(1)
dump:	dump selected parts of object file	dump(1)
hack: exploring the	dungeons of doom	hack(6)
	dup: duplicate an open file-descriptor	dup(2)
dup:	duplicate an open file-descriptor	dup(2)
cut: cut out selected fields of	each line of a file	cut(1)
aliens: the aliens invaders attack	earth	aliens(6)
echo:	echo arguments	echo(1)
	echo: echo arguments	echo(1)
	ecvt, fcvt, gcvt: conv.floating-point number	ecvt(3C)
	ed, red: text editor	ed(1)
end, etext	edata: last locations in program	end(3C)
fedit, flook: inspect and	edit data file or named partitions	fedit(1)
	edit, medit: update a line of text from a	edit(2)
	edit: text editor (variant of ex for casual)	edit(1)
editor:	edit text files	editor(1)
view, vedit: screen oriented display	editor based on ex	vi(1)
ed, red: text	editor	ed(1)
	editor: edit text files	editor(1)
ex: text	editor	ex(1)
ld: link	editor for common object files	ld(1)

a.out: common assembler and link	editor output	a.out(4)
sed: stream	editor	sed(1)
edit: text	editor (variant of ex for casual user)	edit(1)
effective-user-ID, real group-ID,and	effective-group-ID	getuid(2)
getgid: get real-user-ID	effective user-ID, real-group-ID	getuid(2)
geteuid, getegid: get rel-user-iD	effective user-ID, real-group-ID	getuid(2)
	egrep: search file for pattern using full	egrep(1)
times: get process and child process	elapsed times	times(2)
	enable, disable: enable/disable LP prts	enable(1)
enable, disable:	enable or disable LP printers	enable(1)
acct:	enable or disable process accounting	acct(2)
crypt, setkey	encrypt: generate hashing encryption	crypt(3C)
crypt, setkey, encrypt: generate hashing	encryption	crypt(3C)
	end, etext, edata: last locations in prg.	end(3C)
setgrent	endgrent, fgetgrent: get group file entry	getgrent(3C)
seteof: set	end-of-file	seteof(2)
setpwent	endpwent, fgetpwent: get password file	getpwent(3C)
pututline, setutent	endutent, utmpname: access utmp file	getut(3C)
nlist: get	entries from name list	nlist(3C)
linenum: line number	entries in common object file	linenum(4)
ldlinit, ldlitem: manipulate line number	entries of common object file function	ldlread(3x)
ldlseek, ldnlseek: seek to line number	entries of section of common object file	ldlseek(3X)
ldrseek, ldnrseek: seek to relocation	entries of section of common object file	ldrseek(3X)
utmp, wtmp: utmp and wtmp	entry formats	utmp(4)
getgrent, getgrgid, getgrnam: get group file	entry	getgrent(3C)
setgrent, endgrent, fgetgrent: get group file	entry	getgrent(3C)
getpwent getpwnam: get password file	entry	getpwent(3C)
getpwuid: get password file	entry	getpwent(3C)
endpwent, fgetpwent: get password file	entry	getpwent(3C)
getutent, getutid, getutline: access utmp file	entry	getut(3C)
setutent, endutent, utmpname: access utmp file	entry	getut(3C)
symbol name for comm.obj.file symbol table	entry	ldgetname(3X)
ldtbindex: compute index of symbol table	entry of a common object file	ldtbindex(3X)
ldtbread: read indexed symbol table	entry of a common object file	ldtbread(3X)
putpwent: write password file	entry	putpwent(3C)
unlink: remove a directory	entry	unlink(2)
	env: set environment for cmd executions	env(1)

	environ: user environment	environ(5)
environ: user	environment	environ(5)
env: set	environment for command executions	env(1)
getenv: return value for	environment name	getenv(3C)
putenv: change or add value to	environment	putenv(3C)
rc2: run cmd performed for multi-user	environment	rc2(1M)
deroff: remove nroff, troff, tbl and	eqn constructs	deroff(1)
drand48	erand48, lrand48: generate uniformly	drand48(3C)
	erf, erfc: error function and	erf(3M)
erf	erfc: error function and complementary	erf(3M)
	errlog: log system errors	errlog(1M)
perror	errno, sys_errlist, sys_nerr: system error	perror(3C)
error: the Operating system	error device	error(7)
erf, erfc:	error function and compl. error function	erf(3M)
erf, erfc: error function and complementary	error function	erf(3M)
errout: write an	error message	errout(3C)
perror, errno, sys_errlist, sys_nerr: system	error messages	perror(3C)
	error: the Operating system error device	error(7)
matherr:	error-handling function	matherr(3M)
errlog: log system	errors	errlog(1M)
	errout: write an error message	errout(3C)
dial:	establish out-going term-line connection	dial(3C)
setmnt:	establish mount table	setmnt(1M)
end	etext, edata: last locations in program	end(3C)
hypot:	Euclidean distance function	hypot(3M)
expr:	evaluate arguments as an expression	expr(1)
test: condition	evaluation command	test(1)
edit: text editor (variant of	ex for casual user)	edit(1)
	ex: text editor	ex(1)
oriented (visual) display editor based on	ex	vi(1)
robots:	escape from the robots	robots(6)
locking: provide	exclusive access to a byte range	locking(2)
	execl, execv, execle: execute a file	exec(2)
	execve, execlp, execvp: execute a file	exec(2)
execl, execv	execle, execve, execlp, execvp: execute	exec(2)
execl, execv, execle, execve	execlp, execvp: execute a file	exec(2)
execl, execv, execle, execve, execlp, execvp	execute a file	exec(2)

	fcntl: file control	fcntl(2)
	fcntl: file control options	fcntl(5)
ecvt	fcvt, gcvt: convert floating-point number	ecvt(3C)
fopen, freopen	fdopen: open a stream	fopen(3S)
	fedit, flook: inspect and edit data file or	fedit(1)
ferror	feof, clearerr, fileno: stream status	ferror(3S)
	ferror, feof, clearerr, fileno: stream status	ferror(3S)
	ff: list file names and statistics for a file	ff(1M)
fclose	fflush: close or flush a stream	fclose(3S)
getc, getchar	fgetc, getw: get char. or word from stream	getc(3S)
setgrent, endgrent	fgetgrent: get group file entry	getgrent(3C)
setpwent, endpwent	fgetpwent: get password file entry	getpwent(3C)
gets	fgets: get a string from a stream	gets(3S)
	fgrep: search a file for a char. string	fgrep(1)
cut: cut out selected	fields of each line of a file	cut(1)
utime: set	file access and modification times	utime(2)
ldfcn: common object	file access routines	ldfcn(4)
access: determine accessibility of a	file	access(2)
tar: tape	file archiver	tar(1)
cpio: copy	file archives in and out	cpio(1)
pwck, grpck: password and group	file checkers	pwck(1M)
chmod: change mode of a	file	chmod(1)
chmod: change mode of a	file	chmod(2)
chown, chgrp: change owner/group ID of a	file	chown(1)
chown: change owner or group of a	file	chown(2)
mcs: manipulate the object	file comment section	mcs(1)
diff: differential	file comparator	diff(1)
diff3: 3-way differential	file comparison	diff3(1)
fcntl:	file control	fcntl(2)
fcntl:	file control options	fcntl(5)
cprs: compress a common object	file	cprs(1)
umask: set and get	file creation mask	umask(2)
crontab: user crontab	file	crontab(1)
cut: cut out selected fields of a	file	cut(1)
dd: convert and copy a	file	dd(1)
	file: determine file types	file(1)
dump: dump selected parts of object	file	dump(1)

getgrent, getgrgid, getgrnam: get group	file entry	getgrent(3C)
setgrent, endgrent, fgetgrent: get group	file entry	getgrent(3C)
getpwent, getpwnam: get password	file entry	getpwent(3C)
getpwuid: get password	file entry	getpwent(3C)
setpwent, endpwent: get password	file entry	getpwent(3C)
fgetpwent: get password	file entry	getpwent(3C)
getutent, getutid, getutline: access utmp	file entry	getut(3C)
setutent, endutent, utmpname: access utmp	file entry	getut(3C)
pupwent: write password	file entry	putpwent(3C)
execle, execve: execute a	file	exec(2)
execlp, execvp: execute a	file	exec(2)
fgrep: search a	file for a character string	fgrep(1)
grep: search a	file for a pattern	grep(1)
egrep: search a	file for a pattern using full regular	egrep(1)
ldopen, ldaopen: open a common object	file for reading	ldopen(3X)
acct: per-process accounting	file format	acct(4)
ar: common archive	file format	ar(4)
line number entries of a comm.object	file function	ldlread(3X)
group: group	file	group(4)
filehdr:	file header for common object files	filehdr(4)
ldfhread: read the	file header of a common object file	ldfhread(3X)
ldohseek: seek to the optional	file header of a common object file	ldohseek(3X)
split: split a	file into pieces	split(1)
issue: issue identification	file	issue(4)
archive header of member of an archive	file	ldahread(3X)
ldclose, ldaclose: close a common object	file	ldclose(3X)
read file header of a common object	file	ldfhread(3X)
number entries of section of comm obj	file	ldlseek(3X)
to the optional file header of comm obj	file	ldohseek(3X)
entries of a section of comm object	file	ldrseek(3X)
or named section header of comm obj	file	ldshread(3X)
an indexed/named section of comm obj	file	ldsseek(3X)
of a symbol table entry of comm obj	file	ldtbindex(3X)
indexed symbol table entry of comm obj	file	ldtbread(3X)
seek to the symbol table of comm obj	file	ldtbseek(3X)
linenum: linenum entries of comm obj	file	linenum(4)
link: link to a	file	link(2)

produce C source listing from comm obj	file	list(1)
mknod: build special	file	mknod(1M)
make directory, or special, or ordinary	file	mknod(2)
ctermid: generate	file name for terminal	ctermid(3S)
mktemp: make a unique	file name	mktemp(3C)
ff: list	file names and statistics for file system	ff(1M)
newform: change the format of a text	file	newform(1)
nm: print name list of common object	file	nm(1)
ttyslot: find the slot in the utmp	file of the current user	ttyslot(3C)
open: open a	file	open(2)
fuser: identify processes using a	file or file structure	fuser(1M)
fedit, flook: inspect and edit data	file or name partition	fedit(1)
creat: creat a new	file or rewrite an existing one	creat(2)
passwd: password	file	passwd(4)
of several files or subs. lines of one	file	paste(1)
pg:	file perusal filter for CRT's	pg(1)
fseek, ftell, rewind: reposition a	file pointer in a stream	fseek(3S)
read: read from	file	read(2)
relocation info for a common object	file	reloc(4)
bsf: big	file scanner	bfs(1)
sccsfile: format of an SCCS	file	sccsfile(4)
scnhdr: sect.header for a common object	file	scnhdr(4)
fsstat: report	file status	fsstat(1M)
stat, fstat: get	file status	stat(2)
line number info from a comm obj.	file	strip(1)
fuser: identify processes using a file or	file structure	fuser(1M)
sum: print checksum and block count of a	file	sum(1)
retrieve symbol name for common object	file symbol table entry	ldgetname(3X)
syms: common object	file symbol table format	syms(4)
fsck, fsck512, dfsck:	file syst.consistency check and interactive ...	fsck(1M)
fsdb:	file system debugger	fsdb(1M)
ff: list file names and statistics for a	file system	ff(1M)
	file system: format of system volume	fs(4)
mkfs, mkfs512: construct a	file system	mkfs(1M)
mount, umount: mount and unmount	file system	mount(1M)
mount: mount	file system	mount(2)
ustat: get	file system statistics	ustat(2)

25

lockf: record locking on	files	lockf(3C)
rm, rmdir: remove	files or directories	rm(1)
paste: merge same lines of several	files or subsequent lines of one file	paste(1)
pack, pcat, unpack: compress and expand	files	pack(1)
pr: print	files	pr(1)
print section sizes in bytes of common obj	files	size(1)
sort: sort or merge	files	sort(1)
pg: file perusal	filter for CRT's	pg(1)
nl: line numbering	filter	nl(1)
	finc: fast incremental backup	finc(1M)
find:	find files	find(1)
	find: find files	find(1)
ttyname, isatty:	find name of a terminal	ttyname(3C)
lorder:	find ordering relation for an object library	lorder(1)
ttyslot:	find the slot in the utmp file of current	ttyslot(3C)
fish: the game Go	Fish	fish(6)
	fish: the game Go Fish	fish(6)
tee: pipe	fitting	tee(1)
led:	flash hyphens in MCU displays	led(1M)
isnand, isnanf: test for	floating point NaN (Not-a-Number)	isnan(3C)
ecvt, fcvt, gcvt: convert	floating point number to string	ecvt(3C)
frexp, ldexp, modf: manipulate parts of	floating point numbers	frexp(3C)
fedit	flook: inspect and edit data file or named	fedit(1)
	floor, ceil, fmod, fabs: floor, ceiling	floor(3M)
floor, ceil, fmod, fabs:	floor, ceiling remainder, absolute value	floor(3M)
cflow: generate C	flowgraph	cflow(1)
fclose, fflush: close or	flush a stream	fclose(3S)
floor, ceil	fmod, fabs: floor, ceiling remainder, abs	floor(3M)
	fopen, freopen, fdopen: open a stream	fopen(3S)
	fork: create a new process	fork(2)
acct: pre-process accounting file	format	acct(4)
ar: common archive file	format	ar(4)
newform: change the	format of a text file	newform(1)
inode:	format of an i-node	inode(4)
sccsfile:	format of an SCCS file	sccsfile(4)
term:	format of compiled term file	term(4)
cpio:	format of cpio archive	cpio(4)

27

dir:	format of directories	dir(4)
file system:	format of system volume	fs(4)
fspec:	format specification in text files	fspec(4)
syms: common object file symbol table	format	syms(4)
utmp, wtmp: utmp and wtmp entry	formats	utmp(4)
scanf, fscanf, sscanf: convert	formatted input	scanf(3S)
vprintf, vfprintf, vsprintf: print	formatted output of varargs arg. list	vprintf(3S)
printf, fprintf, sprintf: print	formatted output	printf(3S)
printf	fprintf, sprintf: print formatted output	printf(3S)
putc, putchar	fputc, putw: put char/word on a stream	putc(3S)
puts	fputs: put a string on a stream	puts(3S)
	fread, fwrite: binary input and output	fread(3S)
	frec: recover files from a backup tape	frec(1M)
df: report number of	free disk blocks and i-nodes	df(1)
malloc	free, realloc, calloc: main memory allocator	malloc(3C)
malloc	free, realloc, calloc, mallopt, mallinfo: fast	talloc(3X)
fopen	freopen, fdopen: open a stream	fopen(3S)
	frexp, ldexp, modf: manipulate parts of	frexp(3C)
scanf	fscanf, sscanf: convert formatted input	scanf(3S)
checklist: list of file syst.processed by	fsck	checklist(4)
	fsck, fsck512, dfsck: file system consistency	fsck(1M)
fsck	fsck512, dfsck: file sys.consistency check	fsck(1M)
	fsdb: file system debugger	fsdb(1M)
	fseek, ftell, rewind: reposition a file pointer	fseek(3S)
	fspec: format specification in text files	fspec(4)
	fsstat: report file status	fsstat(1M)
stat	fstat: get file status	stat(2)
fseek	ftell, rewind: reposition a file pointer in a	fseek(3S)
stdipc	ftoc: standard inter-proc. communication	stdipc(3C)
	ftw: walk a file tree	ftw(3C)
egrep: search a file for pattern using	full regular expressions	egrep(1)
erf, erfc: error	function and complementary error funct.	erf(3M)
erfc: error funct. and complmt.error	function	erf(3M)
gamma: log gamma	function	gamma(3M)
hypot: Euclidean distance	function	hypot(3M)
line number entries of common obj file	function	ldlread(3X)
matherr: error-handling	function	matherr(3M)

prof: profile within a	function	prof(5)
math: math	functions and constants	math(5)
j0, j1, jn, y0, y1, yn: Bessel	functions	bessel(3M)
exponential, logarithm, power, square root	functions	exp(3M)
floor, ceiling remainder, abs.value	functions	floor(3M)
sinh, cosh, tanh: hyperbolic	functions	sinh(3M)
smsys: machine specific	functions	smsys(2)
tan, asin, acos, atan, atan2: trigonometric	functions	trig(3M)
	fuser: identify processes using a file or	fuser(1M)
fread	fwrite: binary input and output	fread(3S)
	fwtmp, wtmpfix: manipulate connect	fwtmp(1M)
fish: the	game Go Fish	fish(6)
back: the	game of Backgammon	back(6)
bj: the	game of Black Jack	bj(6)
craps: the	game of Craps	craps(6)
life: the	game of Life	life(6)
gamma: log	gamma function	gamma(3M)
	gamma: log gamma function	gamma(3M)
ecvt, fcvt	gcvt: convert floating-point number to	ecvt(3C)
	gendev: generate device numbers	gendev(1M)
termio:	general terminal interface	termio(7)
abort:	generate an IOT fault	abort(3C)
cflow:	generate C flowgraph	cflow(1)
cxref:	generate C program cross-reference	cxref(1)
gendev:	generate device numbers	gendev(1M)
diskusg:	generate disk accounting data by user ID	diskusg(1M)
ctermid:	generate file name for terminal	ctermid(3S)
crypt, setkey, encrypt:	generate hashing encryption	crypt(3C)
ncheck:	generate path names from i-numbers	ncheck(1M)
lex:	generate programs for simple lexical tasks	lex(1)
drand48, erand48, lrand48:	generate uniformly distrb.pseudo-random	drand48(3C)
nrand48, mrand48, jrand48:	generate uniformly distrb.pseudo-random	drand48(3C)
srand48, seed48, lcong48:	generate uniformly distrb.pseudo-random	drand48(3C)
rand, srand: simple random number	generator	rand(3C)
	getc, getchar, fgetc, getw: get character or	getc(3S)
getc	getchar, fgetc, getw: get character or word	getc(3S)
	getcwd: get path-name of current working	getcwd(3C)

getuid, geteuid, getgid	getegid: get real-user-ID, effective-user-ID...	getuid(2)
	getenv: return value for environment..........	getenv(3C)
getuid	geteuid, getgid, getegid: get real-user-ID......	getuid(2)
getuid, geteuid	getgid, getegid: get real-user-ID	getuid(2)
	getgrent, getgrgid, getgrnam: get group file.	getgrent(3C)
getgrent	getgrgid, getgnam: get group file entry.........	getgrent(3C)
getgrent, getgrgid	getgnam: getgroup file entry........................	getgrent(3C)
	getlogin: get login name	getlogin(3C)
	getopt: get option letter from arg. vector	getopt(3C)
	getopt: get parse command options..............	getopt(1)
	getpass: read a password	getpass(3C)
getpid	getpgrp, getppid: get process-ID	getpid(2)
	getpid, getpgrp, getppid: get process-ID........	getpid(2)
getpid, getpgrp	getppid: get proc-ID, proc-group-ID, ...	getpid(2)
	getpw: get name from user-ID......................	getpw(3C)
	getpwent, getpwuid: get password file	getpwent(3C)
	getpwnam: get password file.........................	getpwent(3C)
getpwent, getpwuid	getpwnam: get password file entry...............	getpwent(3C)
getpwent	getpwuid, getpwnam: get password file	getpwent(3C)
	gets, fgets: get a string from a stream.........	gets(3S)
gettydefs: speed and term.settings used by	getty..	gettydefs(4)
	getty: set terminal type, modes, speed,.........	getty(1M)
	gettydefs: speed and term. settings used......	gettydefs(4)
	getuid, geteuid, getgid, getegid: get	getuid(2)
	getutent, getutid: access utmp file................	getut(3C)
	getutline: access utmp file............................	getut(3C)
getutent	getutid, getutline: access utmp file entry	getut(3C)
getutent, getutid	getutline: access utmp file entry..................	getut(3C)
getc, getchar, fgetc	getw: get char/word from stream	getc(3S)
sum and count characters in the files in	given directories	sumdir(1)
ctime, localtime	gmtime, asctime, tzset: conv. date+time	ctime(3C)
fish: the game	Go Fish..	fish(6)
setjmp, longjmp: non-local	goto..	setjmp(3C)
	grep: search a file for a pattern	grep(1)
pwck, grpck: password and	group file checkers	pwck(1M)
getgrent, getgrgid, getgrnam: get	group file entry...	getgrent(3C)
setgrent, endgrent, fgetgrent: get	group file entry...	getgrent(3C)

fuser:	identify processes using a file or	fuser(1M)
id: print user and group	IDs and names	id(1)
nohup: run a command	immune to hangups and quits	nohup(1)
finc: fast	incremental backup	finc(1M)
ldtbindex: compute the	index of a symbol table entry of	ldtbindex(3X)
ldshread, ldnshread: read an	indexed or named section header of	ldshread(3X)
ldsseek, ldnseek: seek to an	indexed or named section of a common	ldsseek(3X)
ldtbread: read an	indexed symbol table entry of a common	ldtbread(3X)
reloc: relocation	information for a common object file	reloc(4)
strip: strip symbol and line number	information from a common object file	strip(1)
lpstat: print LP status	information	lpstat(1)
smos_var: get Supermax Operating System	information	smos_var(2)
inittab: script for the	init process	inittab(4)
	init, telinit: process control initialization	init(1M)
init, telinit: process control	initialization	init(1M)
rsetsioc:	initialize terminal or printer	rsetsioc(1)
popen, pclose:	initiate pipe to or from a process	popen(3S)
	inittab: script for the init process	inittab(4)
clri: clear	i-node	clri(1M)
	inode: format of an i-node	inode(4)
inode: format of an	i-node	inode(4)
df: report number of free disk blocks and	i-nodes	df(1)
fread, fwrite: binary	input and output	fread(3S)
scanf, fscanf, sscanf: convert formatted	input	scanf(3S)
ungetc: push character back into	input stream	ungetc(3S)
stdio: standard buffered	input-output package	stdio(3S)
ferror, feof, clearerr, fileno: stream status	inquiries	ferror(3S)
fedit, flook:	inspect and edit data or named	fedit(1)
install:	install commands	install(1M)
	install: install commands	install(1M)
abs:	integer absolute value	abs(3C)
a64l, l64a: convert between long	integer and base-64 ASCII	a64l(3C)
strtol, atol, atoi: convert string to	integer	strtol(3C)
l3tol, ltol3: convert between 3-byte	integers and long integers	l3tol(3C)
ltol3: conv btw 3-byte integers and long	integers	l3tol(3C)
mailx	interactive message processing system	mailx(1)
dfsck: file system consistency check	interactive repair	fsck(1M)

32

system:	issue a shell command	system(3S)
issue:	issue identification file	issue(4)
	issue: issue identification file	issue(4)
isalpha	isupper, islower, isdigit: classify char.	ctype(3C)
	isxdigit, isalnum: classify characters	ctype(3C)
	isspace, ispunct: classify characters	ctype(3C)
news: print news	items	news(1)
	j0, j1, jn, y0, y1, yn: Bessel functions	bessel(3M)
j0	j1, jn, y0, y1, yn: Bessel functions	bessel(3M)
bj:	the game of black jack	bj(6)
j0, j1	jn, y0, y1, yn: Bessel functions	bessel(3M)
	join: relational database operator	join(1)
nrand48, mrand48	jrand48: generate uniformly distrb.	drand48(3C)
kmem: the	kernel memory device	kmem(7)
killall:	kill all active processes	killall(1M)
	kill: send a signal to a process or a	kill(2)
	kill: terminate a process	kill(1)
	killall: kill all active processes	killall(1M)
	kmem: the kernel memory devices	kmem(7)
	l3tol, ltol3: conv. btw. 3-byte integers	l3tol(3C)
a64l	l64a: conv. btw. long integer and base-64	a64l(3C)
	labelit: provide labels for file systems	labelit(1M)
labelit: provide	labels for file systems	labelit(1M)
awk: pattern scanning and process.	language	awk(1)
bc: arbitrary precision arithmetic	language	bc(1)
cpp: the C	language pre-processor	cpp(1)
the std. and restricted cmd programming	language	sh(1)
end, etext, edata:	last locations in program	end(3C)
tail: deliver the	last part of a file	tail(1)
	lastlogin, monacct, nulladm: shell proced.	acctsh(1M)
at, batch: execute commands at a	later time	at(1)
shl: shell	layer manager	shl(1)
srand48, seed48	lcong48: generate uniformly distributed	drand48(3C)
	ld: link editor for common object files	ld(1)
ldclose	ldaclose: close a common object file	ldclose(3X)
	ldahread: read the archive header of a	ldahread(3X)
ldopen	ldaopen: open a common object file for	ldopen(3X)

34

	ldclose, ldaclose: close a common obj file	ldclose(3X)
frexp	ldexp, modf: manipulate parts of	frexp(3C)
	ldfcn: common object file access routines	ldfcn(4)
	ldfhread: read the file header of a	ldfhread(3X)
	ldgetname: retrieve symbol name for	ldgetname(3X)
ldlread	ldlinit, ldlitem: manipulate line number	ldlread(3X)
ldlread, ldlinit	ldlitem: manipulate line number entries	ldlread(3X)
	ldlread, ldlinit: manipulate line number	ldlread(3X)
	ldlitem: manipulate line number entries	ldlread(3X)
	ldlseek, ldnseek: seek to line number	ldlseek(3X)
ldlseek	ldnlseek: seek to line number entries of	ldlseek(3X)
ldrseek	ldnrseek: seek to relocation entries of	ldrseek(3X)
ldsseek	ldnseek: seek to an indexed/named section	ldsseek(3X)
ldshread	ldnshread: read an indexed/named section	ldshread(3X)
	ldohseek: seek to the optional file header	ldohseek(3X)
	ldopen, ldaopen: open a common obj file	ldopen(3X)
	ldrseek, ldnrseek: seek to relocation	ldrseek(3X)
	ldshread, ldnshread: read an indexed or	ldshread(3X)
	ldsseek, ldnseek: seek to an indexed or	ldsseek(3X)
	ldtbindex: compute the index of a symbol	ldtbindex(3X)
	ldtbread: read an indexed symbol table	ldtbread(3X)
	ldtbseek: seek to the symbol table of a	ldtbseek(3X)
	led: flash hyphens in MCU displays	led(1M)
getopt: get option	letter from argument vector	getopt(3C)
	lex: generate programs for simple lexical	lex(1)
lex: generate programs for simple	lexical tasks	lex(1)
lsearch	lfind: linear search and update	lsearch(3C)
lorder: find ordering relation for an obj	library	lorder(1)
ar: archive and	library maintainer	ar(1)
life: the game of	life	life(6)
	life: the game of life	life(6)
ulimit: get and set user	limits	ulimit(2)
dial: establish and out-going term	line connections	dial(3C)
getty: set term type, modes, speed and	line discipline	getty(1M)
line: read one	line	line(1)
linenum:	line number entries in a common object	linenum(4)
ldlread, ldlinit, ldlitem: manipulate	line number entries of a common object	ldlread(3X)

ldlseek, ldnlseek: seek to	line number entries of a section of	ldlseek(3X)
strip: strip symbol and	line number info from common object	strip(1)
nl:	line numbering filter	nl(1)
cut: cut out selected fields of each	line of a file	cut(1)
edit, medit: update a	line of text from a terminal	edit(2)
lp, cancel: send/cancel req. to an LP	line printer	lp(1)
	line: read one line	line(1)
lsearch, lfind:	linear search and update	lsearch(3C)
	linenum: line number entries in a	linenum(4)
comm: select or reject	lines common to two sorted	comm(1)
uniq: report repeated	lines in a file	uniq(1)
merge same lines of several files or	lines of one file	paste(1)
paste: merge same	lines of several files or subsequent	paste(1)
link, unlink:	link and unlink files and directories	link(1M)
ld:	link editor for common object files	ld(1)
a.out: common assembler and	link editor output	a.out(4)
	link: link to a file	link(2)
cp, ln, mv: copy	link, or move files	cp(1)
link:	link to a file	link(2)
	link, unlink: link and unlink files and	link(1M)
	lint: a C program checker	lint(1)
clist:	list C programs	clist(1)
ls:	list contents of directory	ls(1)
ff:	list file names and statistics for	ff(1M)
nlist: get entries from name	list	nlist(3C)
nm: print name	list of common object file	nm(1)
checklist:	list of file systems processed by fsck	checklist(4)
	list: produce C source listing from a	list(1)
varargs: handle variable argument	list	varargs(5)
print formatted output of a varargs	list	vprintf(3S)
cref: make cross-reference	listing	cref(1)
list: produce C source	listing from a common object file	list(1)
xargs: construct argument	list and execute command	xargs(1)
make	literal copy of file system	volcopy(1M)
cp	ln, mv: copy, link, or move files	cp(1)
chstack: set	load module stack size	chstack(1)
ctime	localtime, gmtime, asctime, tzset: convert	ctime(3C)

end, etext, edata: last	locations in program	end(3C)
plock:	lock process. text, or data in memory	plock(2)
	lockf: record locking on files	lockf(3C)
lockf: record	locking on files	lockf(3C)
	locking: provide exclusive access to	locking(2)
gamma:	log gamma functions	gamma(3M)
newgrp:	log in to a new group	newgrp(1M)
exp	log, log10, pow, sqrt: exponential	exp(3M)
errlog:	log system errors	errlog(1M)
exp, log	log10, pow, sqrt: exponential, logarithm	exp(3M)
exp, log, log10, pow, sqrt: exponential	logarithm, power, square root	exp(3M)
chlds: change	logical disk size	chlds(1M)
getlogin: get	login name	getlogin(3C)
logname: get	login name	logname(1)
cuserid: get character	login name of the user	cuserid(3S)
logname: return	login name of user	logname(3X)
passwd: change	login password	passwd(1)
	login: sign on	login(1)
	logname: get login name	logname(1)
	logname: return login name of user	logname(3X)
a64l, l64a: convert between	long integer and base-64 ASCII	a64l(3C)
ltol3: convert btw 3-byte integers and	long integers	l3tol(3C)
setjmp	longjmp: non-local goto	setjmp(3C)
	lorder: find ordering relation for	lorder(1)
nice: run a command at	low priority	nice(1)
	lp, cancel: send or cancel req. to LP	lp(1)
lp, cancel: send or cancel requests to an	LP line printer	lp(1)
enable, disable: enable or disable	LP printers	enable(1)
accept, reject: allow or prevent	LP requests	accept(1M)
lpsched, lpshut, lpmove: start or stop the	LP scheduler and move	lpsched(1M)
lpadmin: configure the	LP spooling system	lpadmin(1M)
lpstat: print	LP status information	lpstat(1)
	lpadmin: configure the LP spooling	lpadmin(1M)
lpsched, lpshut	lpmove: start or stop the LP scheduler	lpsched(1M)
	lpsched, lpshut: start or stop the LP	lpschced(1M)
	lpmove: start and stop the LP scheduler	lpsched(1M)
lpsched	lpshut, lpmove: start and stop the LP	lpsched(1M)

	lpstat: print LP status information	lpstat(1)
drand48, erand48	lrand48: generate uniformly distrb.	drand48(3C)
	ls: list contents of directory	ls(1)
	lsearch, lfind: linear search and update	lsearch(3C)
	lseek: move file-pointer	lseek(2)
l3tol	ltol3: convert btw 3-byte integers and	l3tol(3C)
	m4: macro processor	m4(1)
machid:	m69k, pdp11, u3b, u3b2, u3b5, vax: get	machid(1)
	machid: m69k, pdp11, u3b, u3b2, u3b5:	machid(1)
smsys:	machine specific functions	smsys(2)
values:	machine-dependent values	values(5)
m4:	macro processor	m4(1)
mail, rmail: send mail to users or read	mail	mail(1)
	mail, rmail: send mail to users or read	mail(1)
mail, rmail: send	mail to users or read mail	mail(1)
	mailx: interactive message process	mailx(1)
malloc, free, realloc, calloc:	main memory allocator	malloc(3C)
free, realloc, calloc, mallopt, mallinfo:	fast main memory allocator	malloc(3X))
make:	maintain, update, and generate groups of	make(1)
ar: archive and library	maintainer	ar(1)
mkdir:	make a directory	mkdir(1)
mknod:	make a directory, or a special or	mknod(2)
mktemp:	make a unique file name	mktemp(3C)
cref:	make cross-reference listing	cref(1)
	make literal copy of file system	volcopy(1M)
	make: maintain, update, and generate	make(1)
banner:	make posters	banner(1)
malloc, free, realloc, calloc, mallopt	mallinfo: fast main memory allocator	malloc(3X)
	malloc, free, realloc, calloc: main memory	malloc(3C)
	malloc. free, realloc, calloc, mallopt	malloc(3X)
malloc, free, realloc, calloc	mallopt, mallinfo: fast main memory	malloc(3X)
tsearch, tfind, tdelete, twalk:	manage binary search trees	tsearch(3C)
hsearch, hcreate, destroy:	manage hash search tables	hsearch(3C)
shl: shell layer	manager	shl(1)
fwtmp, wtmpfix:	manipulate connect accounting records	fwtmp(1M)
ldlread, ldlinit, ldlitem:	manipulate line number entries	ldlread(3X)
frexp, ldexp, modf:	manipulate parts of floating-point	frexp(3C)

ldshread, ldnshread: read an indexed or	named section header of a common	ldshread(3X)
ldsseek, ldnseek: seek to an indexed or	named section of a common object	ldsseek(3X)
ff: list file	names and statistics for a file	ff(1M)
basename, dirname: deliver portions of path	names	basename(1)
term: conventional	names for terminals	term(5)
ncheck: generate path	names from i-numbers	ncheck(1M)
id: print user and group IDs and	names	id(1)
isnand, isnanf: test for floating-point	NaN (Not-a-Number)	isnan(3C)
	ncheck: generate path names from	ncheck(1M)
creat: create a	new file or rewrite an	creat(2)
newgrp: log in to a	new group	newgrp(1M)
fork: create a	new process	fork(2)
	newform: change the format of a	newform(1)
	newgrp: log in to a new group	newgrp(1M)
news: print	news items	news(1)
	news: print news items	news(1)
	nice: change priority of a process	nice(2)
	nice: run a command at low priority	nice(1)
	nl: line numbering filter	nl(1)
	nlist: get entries from name list	nlist(3C)
	nm: print name list of common object	nm(1)
	nohup: run a command immune to	nohup(1)
setjmp, longjmp:	non-local goto	setjmp(3C)
isnand, isnanf: test floating-point NaN	(Not-a-Number)	isnan(3C)
	nrand48, mrand48, jrand48: generate	drand48(3C)
deroff: remove	nroff, troff, tbl and eqn constructs	deroff(1)
null: the	null device	null(7)
	null: the null device	null(7)
lastlogin, monacct	nulladm: shell procedures for	acctsh(1M)
linenum: line	number entries in a common object	linenum(4)
ldlread, ldlinit, ldlitem: manipulate line	number entries of a common object	ldlread(3X)
ldlseek, ldnseek: seek to line	number entries of a section of a	ldlseek(3X)
factor: obtain the prime factors of a	number	factor(1)
arithmetic: provide drill in	number facts	arithmetic(6)
rand, srand: simple random	number generator	rand(3C)
strip: strip symbol and line	number information from a common	strip(1)
df: report	number of free disk blocks and	df(1)

atof: convert string to dbl-precision	number	strtod(3C)
ecvt, fcvt, gcvt: convert floating-point	number to string	ecvt(3C)
nl: line	numbering filter	nl(1)
generate uniformly distrb.pseudo-random	numbers	drand48(3C)
ldexp: manipulate parts of floating-point	numbers	frexp(3C)
modf: manipulate parts of floating-point	numbers	frexp(3C)
gendev: generate device	numbers	gendev(1M)
ldfhread: read the file header	of a common object file	ldfhread(3X)
ldfcn: common	object file access routines	ldfcn(4)
mcs: manipulate the	object file comment section	mcs(1)
cprs: compress a common	object file	cprs(1)
dump: dump selected parts of an	object file	dump(1)
ldopen, ldaopen: open a common	object file for reading	ldopen(3X)
manipulate line number entries of common	object file function	ldlread(3X)
ldclose, ldaclose: close a common	object file	ldclose(3X)
ldfhread: read file header of common	object file	ldfhread(3X)
line number entries of secrion of common	object file	ldlseek(3X)
seek optional file header of common	object file	ldohseek(3X)
to relocation entries of section of common	object file	ldrseek(3X)
indexed or name section header of common	object file	ldshread(3X)
seek indexed or named section of common	object file	ldsseek(3X)
the index of symbol table entry of common	object file	ldtbindex(3X)
read indexed symbol table entry of common	object file	ldtbread(3X)
ldtbseek: seek symbol table of common	object file	ldtbseek(3X)
linenum: line number entries in common	object file	linenum(4)
list: produce C source listing from	object file	list(1)
nm: print name list of common	object file	nm(1)
reloc: relocation info for a common	object file	reloc(4)
scnhdr: section header for common	object file	scnhdr(4)
an line number info from common	object file	strip(1)
ldgetname: retrieve symbol name for	object file symbol table entry	ldgetname(3X)
syms: common	object file symbol table format	syms(4)
filehdr: file header for common	object files	filehdr(4)
ld: link editor for common	object files	ld(1)
size: print section sizes in bytes of	object files	size(1)
lorder: find ordering relation for an	object library	lorder(1)
factor:	obtain the prime factors of a number	factor(1)

stty2: set the	options for a terminal or printer	stty2(1)
getopt: parse command	options	getopt(1)
lorder: find	ordering relation for object library	lorder(1)
mknod: make a directory, or special, or	ordinary file	mknod(2)
vi, view, vedit: screen	oriented (visual) display editor	vi(1)
cpio: copy file archived in and	out	cpio(1)
cut: cut	out selected fields of each line of	cut(1)
dial: establish an	out-going terminal line connection	dial(3C)
a.out: common assembler and link editor	output	a.out(4)
fread, fwrite: binary input and	output	fread(3S)
vprintf, vfprintf, vsprintf: print formatted	output of varargs argument file	vprintf(3S)
printf, fprintf, sprintf: print formatted	output	printf(3S)
acctdis, acctusg, accton, acctwtmp:	overview of accounting and misc.	acct(1M)
chown: change	owner and group of a file	chown(2)
chown, chgrp: change	owner and group ID of a file	chown(1)
par_cho: change	owner-ID and group-ID of a partition	par_cho(2)
	pack, pcat, unpack: compres and expand	pack(1)
curses: CRT screen handling and optim.	package	curses(3X)
stdio: standard buffered input-output	package	stdio(3S)
ftoc: standard inter-process communication	package	stdipc(3C)
set_parm: define additional system call	parameters	set_parm(2)
	par_att: attach a memory partition	par_att(2)
	par_chm: change access rights to	par_chm(2)
	par_cho: change owner-ID and group-ID	par_cho(2)
	par_cre: create a memory partition	par_cre(2)
	par_del: delete a named partition	par_del(2)
	par_det: detach a memory partition	par_det(2)
getppid: get process-ID, process-group-ID	parent-process-ID	getpid(2)
getopt:	parse command options	getopt(1)
tail: deliver the last	part of a file	tail(1)
flook: inspect and edit data file or named	partition	fedit(1)
par_att: attach a memory	partition	par_att(2)
par_chm: change access rights to	partition	par_chm(2)
par_cho: change owner-ID and group-ID	partition	par_cho(2)
par_cre: create a memory	partition	par_cre(2)
par_del: delete a named	partition	par_del(2)
par_det: detach a memory	partition	par_det(2)

47

	printf, fprintf, sprintf: print	printf(3S)
nice: run a command at low	priority	nice(1)
nice: change	priority of a process	nice(2)
brc, bcheckrc: system initialization	procedures	brf(1M)
chargefee, ckpacct, dodisk: shell	procedures for accounting	acctsh(1M)
lastlogin, monacct, nulladm: shell	procedures for accounting	acctsh(1M)
prctmp, prdaily, prtacct: shell	procedures for accounting	acctsh(1M)
runacct, shutacct, startup, turnacct: shell	procedures for accounting	acctsh(1M)
acct: enable or disable	process accounting	acct(2)
acctprc1, acctprc2:	process accounting	acctprc(1M)
acctcom: search and print	process accounting files	acctcom(1)
alarm: set a	process alarm clock	alarm(2)
times: get	process and child process elapsed times	times(2)
init, telinit:	process control initialization	init(1M)
timex: time a command: report	process data and system activity	timex(1)
times: get process and child	process elapsed times	times(2)
resume: resume	process execution	resume(2)
suspend: suspend	process execution	suspend(2)
exit, _exit: terminate	process	exit(2)
fork: create a new	process	fork(2)
inittab: script for the init	process	inittab(4)
kill: terminate a	process	kill(1)
nice: change priority of a	process	nice(2)
kill: send a signal to a	process or group of	kill(2)
popen, pclose: initiate pipe to or from a	process	popen(3S)
ps: report	process status	ps(1)
plock: lock	process, text, or data in memory	plock(2)
wait, waitx: wait for child	process to stop or terminate	wait(2)
ptrace:	process trace	ptrace(2)
pause: suspend	process until signal	pause(2)
wait: await completion of	process	wait(1)
checklist: list of file systems	processed by fsck	checklist(4)
kill: send signal to process or group of	processes	kill(2)
killall: kill all active	processes	killall(1M)
fuser: identify	processes using a file or file structure	fuser(1M)
getpid, getpgrp, getppid: get process-ID	process-goup-ID, and parent-process-ID	getpid(2)
setgrp: set	process-group-ID	setgrp(2)

jrand48: generate uniformly distributed	pseudo-random numbers	drand48(3C)
seed48, lcong48: gen. uniformly distrb	pseudo-random numbers	drand48(3C)
	ptrace: process trace	ptrace(2)
ungetc:	push character back into input stream	ungetc(3C)
puts, fputs:	put a string on a stream	puts(3S)
putc, putchar, fputc, putw:	put character or word on a stream	putc(3S)
	putc, putchar, fputc, putw: put char. or	putc(3S)
putc	putchar, fputc, putw: put character	putc(3S)
	putenv: change or add value to env.	putenv(3C)
	putpwent: write password file	putpwent(3C)
	puts, fputs: put a string on a stream	puts(3S)
	pututline, setutent, endutent, utmpname:	getut(3C)
putc, putchar, fputc	putw: put character or word on a	putc(3S)
	pwck, grpck: password and group	pwck(1M)
	pwd: print working directory	pwd(1)
	qsort: quicker sort	qsort(3C)
tput: initialize a terminal or	query terminfo database	tput(1)
msgget: get message	queue	msgget(2)
ipcrm: remove a message	queue, semaphore set, or shared	ipcrm(1)
qsort:	quicker sort	qsort(3C)
nohup: run a command immune to hangups and	quits	nohup(1)
	rand, srand: simple random number	rand(3C)
rand, srand: simple	random number generator	rand(3C)
locking: provide excl.access to a byte	range	locking(2)
	rc0: run cmds performed to stop the	rc0(1M)
	rc2: run cmds performed for multi-user	rc2(1M)
getpass:	read a password	getpass(3C)
ldshread, ldnshread:	read an indexed or named section	ldshread(3X)
ldtbread:	read an indexed symbol table entry	ldtbread(3X)
aread: asynchronous	read	aread(2)
mail, rmail: send mail to users or	read mail	mail(1)
line:	read one line	line(1)
ldahread:	read the archive header of a member	ldahread(3X)
ldfhread:	read the file header of a common	ldfhread(3X)
ldopen, ldaopen: open common obj file for	reading	read(2)
getegid: get	real-user-ID, effective-user-ID	getuid(2)
getegid: get	real-group-ID, effective-group-ID	getuid(2)

50

ipcs:	report inter-process communication	iprs(1)
df:	report number of free disk blocks	df(1)
timex: time a command:	report process data and system	timex(1)
ps:	report process status	ps(1)
uniq:	report repeated lines in a file	uniq(1)
fseek, ftell, rewind:	reposition a file pointer in a stream	fseek(3S)
accept, reject: allow or prevent LP	requests	accept(1M)
lpmove: start or stop LP scheduler and move	requests	lpsched(1M)
lp, cancel: send or cancel	requests to an LP	lp(1)
sh, rsh: shell, the standard and	restricted command prg. language	sh(1)
resume:	resume process execution	resume(2)
	resume: resume process execution	resume(2)
ldgetname:	retrieve symbol name for common	ldgetname(3X)
logname:	return login name of user	logname(3X)
getenv:	return value for environment name	getenv(3C)
stat: data	returned by stat system call	stat(5)
fseek, ftell	rewind: reposition a file pointer	fseek(3C)
creat: create a new file or	rewrite an existing one	creat(2)
par_chm: change access	rights to partition	par_chm(2)
	rm, rmdir: remove files or directories	rm(1)
mail	rmail: send mail to users or read	mail(1)
rm	rmdir: remove files or directories	rm(1)
	robots: escape from the robots	robots(6)
robots: excape from the	robots	robots(6)
chroot: change	root directory	chroot(2)
chroot: change	root directory for a command	chroot(1M)
sqrt: exponential, logarth.,power, square	root functions	exp(3M)
ldfcn: common object file access	routines	ldfcn(4)
regexp: regular expression compile and match	routines	regexp(5)
	rsetsioc: initialize term or printer	rsetsioc(1)
sh	rsh: shell, the standard and restricted	sh(1)
nice:	run a command at low priority	nice(1)
nohup:	run a command immune to hangup and	nohup(1)
rc2:	run commands performed for multi-user	rc2(1M)
rc0:	run commands performed to stop the	rc0(1M)
runacct:	run daily accounting	runacct(1M)
	runacct: run daily accounting	runacct(1M)

	runacct, shutacct, startup, turnacct: shell ...	acctsh(1M)
paste: merge	same lines of several files or	paste(1)
brk	sbrk: change data segment space alloc	brk(2)
	scanf, fscanf, sscanf: conv formatted input ..	scanf(3S)
bsf: big file	scanner	bfs(1)
awk: pattern	scanning and processing language	awk(1)
sccsfile: format of an	SCCS file	sccsfile(4)
	sccsfile: format of an SCCS file	sccsfile(4)
lpsched, lpshut, lpmove: start and stop LP	scheduler and move requests	lpsched(1M)
	scnhdr: section header for a common	scnhdr(4)
curses: CRT	screen handling and optimization	curses(3X)
vi, view, vedit:	screen oriented (visual) display	vi(1)
inittab:	script for the init process	inittab(4)
	sdb: symbolic debugger	sdb(1)
	sdiff: side-by-side diff program	sdiff(1)
fgrep:	search a file for a character	fgrep(1)
grep:	search a file for a pattern	grep(1)
egrep:	search a file for pattern using full	egrep(1)
bsearch: binary	search a sorted table	bsearch(3C)
acctcom:	search and print process accounting	acctcom(1)
lsearch, lfind: linear	search and update	lsearch(3C)
hsearch, hcreate, hdestroy: manage hash	search tables	hsearch(3C)
tsearch, tfind, tdelete, twalk: manage binary	search trees	tsearch(3C)
scnhdr:	section header for a common object	scnhdr(4)
ldshread, ldnshread: read an indexed and named	section header for a common object	ldshread(3X)
mcs: manipulate the object file comment	section	mcs(1)
ldnlseek: seek to line number entries of	section of a common object file	ldlseek(3X)
ldnrseek: seek to relocation entries of	section of a common object file	ldrseek(3X)
ldsseek, ldnseek: seek to an indexed or name	section of a common object file	ldsseek(3X)
size: print	section sizes in bytes of common object	size(1)
	sed: stream editor	sed(1)
srand48	seed48, lcong48: generate uniformly distrb..	drand48(3C)
ldsseek, ldnseek:	seek to an indexed or named section	ldsseek(3X)
ldsseek, ldnlseek:	seek to line number entries of section	ldlseek(3X)
ldrseek, ldnrseek:	seek to relocation entries of section	ldrseek(3X)
ldohseek:	seek to optional file header	ldohseek(3X)
ldtbseek:	seek to the symbol table of common	ldtbseek(3X)

shmget: get a shared memory	segment	shmget(2)
brk, sbrk: change data	segment space allocations	brk(2)
comm:	select or reject lines common to	comm(1)
cut: cut out	selected fields of each lines of	cut(1)
dump: dump	selected part of an object file	dump(1)
semctl:	semaphore control operations	semctl(2)
semop:	semaphore operations	semop(2)
ipcrm: remove a message queue	semaphore set, or shared memory ID	ipcrm(1)
semget: get set of	semaphores	semget(2)
	semctl: semaphore control operations	semctl(2)
	semget: get set of semaphores	semget(2)
	semop: semaphore operations	semop(2)
kill:	send a signal to process or group of	kill(2)
mail, rmail:	send mail to users or read mail	mail(1)
lp, cancel:	send or cancel requests to an LP	lp(1)
calendar: reminder	service	calendar(1)
alarm:	set a process alarm clock	alarm(2)
umask:	set and get file creation mask	umask(2)
mcumask:	set and get MCU mask	mcumask(2)
ascii: map of ASCII character	set	ascii(5)
timezone:	set default system time zone	timezone(4)
setdioc: display or	set disk operation modes	setdioc(1M)
seteof:	set end-of-file	seteof(2)
env:	set environment for command execution	env(1)
utime:	set file access and modification	utime(2)
umask:	set file-creation mode mask	umask(1)
ISO-8859/1: international character	set	ISO-8859/1(5)
chstack:	set load module stack size	chstack(1)
mcumask:	set MCU mask	mcumask(1)
uname: get and	set name of current operating system	uname(2)
semget: get	set of semaphores	semget(2)
ipcrm: remove a message queue, semaphore	set, or shared memory ID	ipcrm(1)
setpgrp:	set process-group-ID	setpgrp(2)
settime:	set system time	settime(1)
tabs:	set tabs on a terminal	tabs(1)
getty:	set terminal type, modes, speed	getty(1M)
date: print and	set the date	date(1)

	shmctl: shared memory control operations..	shmctl(2)
	shmget: get a shared memory segment	shmget(2)
	shmop: shared memory operations	shmop(2)
shutdown:	shut down system, change system................	shutdown(1M)
runacct	shutacct, startup, turnacct: shell procedure	acctsh(1M)
	shutdown: shut down system, change...........	shutdown(1M)
sdiff:	side-by-side difference program	sdiff(1)
login:	sign on..	login(1)
pause: suspend process until	signal	pause(2)
signal: specify what to do upon receipt	signal	signal(2)
	signal: specify what to do upon receipt..........	signal(2)
kill: send a	signal to process or a group of procs............	kill(2)
ssignal, gsignal: software	signals	ssignal(3C)
lex: generate programs for	simple lexical tasks	lex(1)
rand, srand:	simple random number generator.................	rand(3C)
	sin, cos, tan, asin, acos, atan, atan2:............	trig(3M)
	sinh, cosh, tanh: hyperbolic functions	sinh(3M)
chlds: change logical disk	size..	chlds(1M)
chstack: set load module stack	sixe	chstack(1)
dsize: display disk	size..	dsize(1)
	size: print section sizes in bytes of................	size(1)
size: print section	sizes in bytes of common object files............	size(1)
	sleep: suspend execution for an interval.......	sleep(1)
	sleep: suspend execution for interval	sleep(3C)
ttyslot: find the	slot in the utmp file of the	ttyslot(3C)
	smos_var: get Supermax Opr. System	smos_var(2)
	smsys: machine specific functions	smsys(2)
ssignal, gsignal:	software signals..	ssignal(3C)
sort:	sort or merge files	sort(1)
qsort: quicker	sort	qsort(3C)
	sort: sort or merge files	sort(1)
tsort: topological	sort	tsort(1)
comm: select or reject lines common to two	sorted files..	comm(1)
bsearch: binary search a	sorted table	bsearch(3C)
list: produce C	source listing from a common object	list(1)
brk, sbrk: change data segment	space allocation	brk(2)
mknod: build	special file	mknod(1M)

mknod: make a directory, or a	special or ordinary file	mknod(2)
smsys: machine	specific functions	smsys(2)
fspec: format	specifications in text files	fspec(4)
signal:	specify what to do upon receipt of	signal(2)
getty: set terminal type, modes	speed, and line discipline	getty(1M)
gettydefs:	speed and terminal settings used by	gettydefs(4)
split:	split a file into pieces	split(1)
csplit: context	split	csplit(1)
	split: split a file into pieces	split(1)
lpadmin: configure the LP	spooling system	lpadmin(1M)
printf, fprintf	sprintf: print formatted output	printf(3S)
exp, log, log10, pow	sqrt: exponential, logarithm, power	exp(3M)
exp, sqrt: exponential, logarithm, power	square root functions	exp(3M)
rand	srand: simple random number generator	rand(3C)
	srand48, seed48, lcong48: generate	drand48(3C)
scanf, fscanf	sscanf: convert formatted input	scanf(3S)
	ssignal, gsignal: software signals	ssignal(3C)
chstack: set load module	stack size	chstack(1)
sh, rsh: shell, the	standard and restricted command	sh(1)
stdio:	standard buffered input-output package	stdio(3S)
stdipc, ftoc:	standard inter-process communication	stdipc(3C)
lpsched, lpshut, lpmove:	start or stop the LP scheduler	lpsched(1M)
runacct, shutacct	startup, turnacct: shell procedures for	acctsh(1M)
	stat: data returned by stat system call	stat(5)
	stat, fstat, get file status	stat(2)
stat: data returned by	stat system call	stat(5)
shutdown: shut down system, change system	state	shutdown(1M)
ff: list file names and	statistics for a file system	ff(1M)
ustat: get file system	statistics	ustat(2)
fsstat: report file	status	fsstat(1M)
lpstat: print LP	status information	lpstat(1)
ferror, feof, clearerr, fileno: stream	status inquiries	ferror(3S)
report inter-proc.communication facilities	status	ipcs(1)
ps: report process	status	ps(1)
stat, fstat: get file	status	stat(2)
	stdio: standard buffered input-output	stdio(3S)
	stdipc, ftoc: standard inter-process	stdipc(3C)

	stime: set time	stime(2)
wait, waitx: wait for child process to	stop or terminate	wait(2)
lpsched, lpshut, lpmove: start or	stop LP scheduler and move requests	lpsched(1M)
rc0: run commands performed to	stop the operating system	rc0(1M)
	strcat, strdup, strncat: string operations	string(3C)
strncpy, strlen	strchr: string operations	string(3C)
	strcmp, strncmp, strcpy: string operations	string(3C)
strcmp, strncmp	strcpy: string operations	string(3C)
	strcspn, strtok: string operations	string(3C)
strcat	strdup, strncat: string operations	string(3C)
sed:	stream editor	sed(1)
fclose, fflush: close or flush a	stream	fclose(3S)
fopen, freopen, fdopen: open a	stream	fopen(3S)
ftell, rewind: reposition a file pointer in a	stream	fseek(3S)
fgetc, getw: get character or word from	stream	getc(3S)
gets, fgets: get a string from a	stream	gets(3S)
fputc, putw: put character or word on a	stream	putc(3S)
puts, fputs: put a string on a	stream	puts(3S)
setbuf, setvbuf: assign buffering to a	stream	setbuf(3S)
ferror, feof, clearerr, fileno:	stream status inquiries	ferror(3S)
ungetc: push character back into input	stream	ungetc(3S)
	streamdrv: copy with buffering	streamdrv(1)
fcvt, gcvt: convert floating-point number to	string	ecvt(3C)
fgrep: search a file for a character	string	fgrep(1)
gets, fgets: get a	string from a stream	gets(3S)
puts, fputs: put a	string on a stream	puts(3S)
strcat, strdup, strncat:	string operations	string(3C)
strcmp, strncmp, strcpy:	string operations	string(3C)
strcspn, strtok:	string operations	string(3C)
strncpy, strlen, strchr:	string operations	string(3C)
strrchr, strpbrk, strspn:	string operations	string(3C)
strtod, atof: convert	string to dbl-precision number	strtod(3C)
strtol, atol, atoi: convert	string to integer	strtol(3C)
	strip: strip symbol and line number	strip(1)
strip:	strip symbol and line number information	strip(1)
strncpy	strlen, strchr: string operations	string(3C)
strcat, strdup	strncat: string operations	string(3C)

strcmp	strncmp, strcpy: string operations	string(3C)
	strncpy, strlen, strchr: string operations	string(3C)
strrchr	strpbrk, strspn: string operations	string(3C)
	strrchr, strpbrk, strspn: string operations	string(3C)
strrchr, strpbrk	strspn: string operations	string(3C)
	strtod, atof: convert string to	strtod(3C)
strspn	strtok: string operations	string(3C)
	strtol, atol, atoi: convert string to	strtol(3C)
fuser: identify proc. using a file of file	structure	fuser(1M)
	stty: set the options for a terminal or	stty(1)
	stty2: set the options for a terminal or	stty2(1)
	su: become super-user or another user	su(1M)
paste: merge same lines of several lines	subsequent lines of one file	paste(1)
sumdir:	sum and count characters in the files	sumdir(1)
	sum: print checksum and block count	sum(1)
	sumdir: sum and count characters in	sumdir(1)
du:	summarize disk usage	du(1)
acctcms: command	summary from per-process accounting	acctcms(1M)
sync: update the	super block	sync(1M)
smos_var: get	Supermax Operating System information	smos_var(2)
su: become	super-user of another user	su(1M)
asuspend: asynchronous	suspend	asuspend(2)
sleep:	suspend execution for an interval	sleep(1)
sleep:	suspend execution for interval	sleep(3C)
suspend:	suspend process execution	suspend(2)
pause:	suspend process until signal	pause(2)
	suspend: suspend process execution	suspend(2)
	swab: swap bytes	swab(3C)
swab:	swap bytes	swab(3C)
strip: strip	symbol and line number information	strip(1)
ldgetname: retrieve	symbol name for common object file	ldgetname(3X)
retrieve symbol name for common obj file	symbol table entry	ldgetname(3X)
ldtbindex: compute the index of a	symbol table entry of a common	ldtbindex(3X)
ldtbread: read an indexed	symbol table entry for a common	ldtbread(3X)
syms: common object file	symbol table format	syms(4)
ldtbseek: seek to the	symbol table of a common object	ldtbseek(3X)
sdb:	symbolic debugger	sdb(1)

	syms: common object file symbol table	syms(4)
	sync: update the super block	sync(1M)
	sync: write disk cache to disk	sync(2)
csh: a shell with C-like	syntax	csh(1)
perror, errno	sys_errlist, sys_nerr: system error	perror(3C)
perror, errno, errlist	sys_nerr: system error messages	perror(3C)
timex: time a command: report process	system activity	timex(1)
set_parm: define additional	system call parameters	set_parm(2)
stat: data returned by stat	system call	stat(5)
shutdown: shut down	system, change system state	shutdown(1M)
chhw: change	system configuration	chhw(1M)
config: print	system configuration	config(1M)
fsck, fsck512, dfsck: file	system consistency check and	fsck(1M)
crash: provoke	system crash	crash(1M)
types: primitive	system data types	types(5)
fsdb: file	system debugger	fsdb(1M)
error: the Operating	System error device	error(7)
perror, errno, sys_errlist, sys_nerr:	system error messages	perror(3C)
errlog: log	system errors	errlog(1M)
ff: list file names and statistics for	system	ff(1M)
file	system: format of system volume	fs(4)
smos_var: get Supermax Operating	System information	smos_var(2)
brx, bcheckrc:	system initialization procedures	brf(1M)
	system: issue a shell command	system(3S)
lpadmin: configure the LP spooling	system	lpadmin(1M)
mailx: interactive message processing	system	mailx(1)
mkfs, mkfs512: construct a file	system	mkfs(1M)
mount, umount: mount and unmount file	system	mount(1M)
mount: mount file	system	mount(2)
run commands performed to stop the	system	rc0(1M)
shutdown: shut down system, change	system state	shutdown(1M)
ustat: get file	system statistics	ustat(2)
mnttab: mouned file	system table	mnttab(4)
settime: set	system time	settime(1)
timezone: set default	system time zone	timezone(4)
umount: unmount a file	system	umount(2)
print name of current UNIX	system	uname(1)

	term: conventional names for terminals	term(5)
term: format of compiled	term file	term(4)
	term: format of compiled term file	term(4)
	term: terminals	term(7)
terminfo:	terminal capability data base	terminfo(4)
ctermid: generate file name for	terminal	ctermid(3S)
edit, medit: update a line of text from	terminal	edit(2)
terminology: compile Virtual	Terminal Interface programs	terminology(1)
termio: general	terminal interface	termio(7)
vti: Virtual	Terminal Interface	vti(5)
dial: establish an out-going	terminal line connection	dial(3C)
rsetsioc: initialize	terminal or printer	rsetsioc(1)
stty: set the option for a	terminal or printer	stty(1)
stty2: set the options for a	terminal or printer	stty2(1)
tput: initialize a	terminal or query terminfo database	tput(1)
gettydefs: speed and	terminal settings used by getty	gettydefs(4)
tabs: set tabs on a	terminal	tabs(1)
tty: get the name of the	terminal	tty(1)
tty: controlling	terminal	tty(7)
ttyname, isatty: find name of a	terminal	ttyname(3C)
getty: set	terminal type, modes, speed and line	getty(1M)
term: conventional names for	terminals	term(5)
term:	terminals	term(7)
kill:	terminate a process	kill(1)
exit, _exit:	terminate process	exit(2)
wait, waitx: wait for child process to stop or	terminate	wait(2)
tic:	terminfo compiler	tic(1M)
tput: initialize a terminal or query	terminfo database	tput(1)
	terminfo: terminal capability data	terminfo(4)
	terminology: compile Virtual Terminal	terminology(1)
termio: general	terminal interface	termio(7)
	test: condition evaluation cmd	test(1)
isnand, isnanf:	test for floating-point NaN (Not-a-Number)	isnan(3C)
ed, red:	text editor	ed(1)
ex:	text editor	ex(1)
edit:	text editor (variant of ex for casual user)	edit(1)
newform: change the format of a	text file	newform(1)

editor: edit	text files	editor(1)
fspec: format specification in	text files	fspec(4)
edit, medit: update a line of	text from a terminal	edit(2)
plock: lock process	text, or data in memory	plock(2)
tsearch	tfind, tdelete, twalk: manage binary	tsearch(3C)
	tic: terminfo compiler	tic(1M)
ttt:	tic-tac-toe	ttt(6)
timex:	time a command: report process data	timex(1)
time:	time a command	time(1)
at, batch: execute commands at a later	time	at(1)
gmtime, asctime, tzset: convert date and	time	ctime(3C)
dcopy: copy file systems for optimal search	time	dcopy(1M)
	time: get time	time(2)
profil: execution	time profile	profil(2)
settime: set system	time	settime(1)
stime: set	time	stime(2)
	time: time a command	time(1)
time: get	time	time(2)
clock: report CPU	time used	clock(3C)
timezone: set default system	time zone	timezone(4)
	times: get process and child process	times(2)
touch: opdate access and modification	times of a file	touch(1)
times: get process and child process elapsed	times	times(2)
utime: set file access and modification	times	utime(2)
	timex: time a command: report process	timex(1)
	timezone: set default system time zone	timezone(4)
	tmpfile: create a temporary file	tmpfile(3S)
	tmpnam, tempnam: create a name for a	tmpfile(3S)
toupper, tolower	toascii: translate characters	conv(3C)
toupper	tolower, toascii: translate characters	conv(3C)
tsort:	topological sort	tsort(1)
acctmerg: merge or add	total accounting files	acctmerg(1M)
	touch: update access and modification	touch(1)
	toupper, tolower, toascii: translate char	conv(3C)
	tput: initialize a terminal or query	tput(1)
	tr: translate characters	tr(1)
ptrace: process	trace	ptrace(2)

toupper, tolower, toascii:	translate characters	conv(3C)
tr:	translate characters	tr(1)
ftw: walk a file	tree	ftw(3C)
tfind, tdelete, twalk: manage binary search	trees	tsearch(3C)
sin, cos, tan, asin, acos, atan, atan2:	trigonometric functions	trig(3M)
deroff: remove nroff	troff, tbl, and eqn constructs	deroff(1)
	true, false: provide truth values	true(1)
pdp11, u3b, u3b2: get processor type	truth value	machid(1)
u3b5, vax: get processor type	truth value	machid(1)
true, false: provide	truth values	true(1)
	tsearch, tfind, tdelete, twalk: manage	tsearch(3C)
	tsort: topological sort	tsort(1)
	ttt: tic-tac-toe	ttt(6)
	tty: controlling terminal	tty(7)
	tty: get the name of the terminal	tty(1)
	ttyname, iastty: find name of terminal	ttyname(3C)
	ttyslot: find the slot in the utmp file	ttyslot(3C)
runacct, shutacct, startup	turnacct: shell procedures for	acctsg(1M)
tasearch, tfind, tdelete	twalk: manage binary search trees	tsearch(3C)
cmp: compare	two files	cmp(1)
comm: select or reject lines common to	two sorted files	comm(1)
file: determine file	type	file(1)
getty: set terminal	type, mode, speed, and line disc.	getty(1M)
pdp11, u3b, u3b2: get processor	type truth value	machid(1)
u3b5, vax: get processor	type truth value	machid(1)
	types: primitive system data types	types(5)
types: primitive system data	types	types(5)
ctime, localtime,gmtime, asctime	tzset: convert date and time	ctime(3C)
machid: m69k, psp11	u3b, u3b2, u3b5, vax: get processor	machid(1)
machid: m69k, pdp11, u3b	u3b2, u3b5, vax: get processor type	machid(1)
machid: m69k, pdp11, u3b, u3b2	u3b5, vax: get processor type truth	machid(1)
	uadmin: administrative control	uadmin(2)
	uadmind: administrative control	uadmin(1M)
	ulimit: get and set user limits	ulimit(2)
	umask: set and get file creation mask	umask(2)
	umask: set file-creation mode mask	umask(1)
mount	umount: mount and unmount file system	mount(1M)

su: become super-user or another	user	su(1M)
find the slot in the utmp file of current	user	ttyslot(3C)
write: write to another	user	write(1)
setuid, setgid: set	user-ID and group-ID	setuid(2)
edit: text editor (variant of ex for	user)	edit(1)
mail, rmail: send mail to	users or read mail	mail(1)
wall: write to all	users	wall(1)
fuser: identify processes	using a file or file structure	fuser(1M)
egrep: search a file for pattern	using full regular expressions	egrep(1)
	ustat: get file system statistics	ustat(2)
	utime: get file access and modification	utime(2)
utmp, wtmp:	utmp and wtmp entry formats	utmp(4)
getutent, getuid, getutline: access	utmp file entry	getut(3C)
pututline, setutent: access	utmp file entry	getut(3C)
endutent, utmpname: access	utmp file entry	getut(3C)
ttyslot: find the slot in the	utmp file of current user	ttyslot(3C)
	utmp, wtmp: utmp, wtmp entry formats	utmp(4)
pututline, setutent, endutent	utmpname: access utmp file entry	getut(3C)
abs: integer absolute	value	abs(3C)
getenv: return	value for environment name	getenv(3C)
fmod, fabs: floor, ceiling remainder, absolute	value functions	floor(3M)
u3b, u3b2: get processor type truth	value	machid(1)
u3b5, vax: get processor type truth	value	machid(1)
putenv: change or add	value to environment	putenv(3C)
	values: machine-dependent values	values(5)
true, false: provide truth	values	true(1)
values: machine-dependent	values	values(5)
vfprintf, vsprintf: print formatted output of	varargs argument list	vprintf(3S)
	varargs: handle variable argument list	varargs(5)
varargs: handle	variable argument list	varargs(5)
edit: text editor	(variant of ex for casual users)	edit(1)
machid: m69k, pdp11, u3b, u3b2, u3b5	vax: get processor type truth	machid(1)
getopt: get option letter from argument	vector	getopt(3C)
vi view	vedit: screen oriented (visual) display	vi(1)
assert:	verify program assertion	assert(3X)
sysvers: display operating system	versions	sysvers(1M)
vprintf	vfprintf, vsprintf: print formatted output	vprintf(3S)

	vi, view, vedit: screen oriented (visual).........	vi(1)
vi	view, vedit: screen oriented (visual)..............	vi(1)
terminology: compile	Virtual Terminal Interface programs	terminology(1)
vti:	Virtual Terminal Interface.............................	vti(5)
vi, view, vedit: screen oriented	(visual) display editor based...........................	vi(1)
file system: format of system	volume..	fs(4)
	vprintf, vfprintf, vsprintf: print formatted...	vprintf(3S)
vprintf, vfprintf	vsprintf: print formatted output of...............	vprintf(3S)
	vti: Virtual Terminal Interface......................	vti(5)
	wait: await completion of process..................	wait(1)
wait, waitx:	wait for child process to stop or	wait(2)
	wait, waitx: wait for child process to	wait(2)
wait	waitx: wait for child process to stop or.........	wait(2)
ftw:	walk a file tree..	ftw(3C)
	wall: write to all users....................................	wall(1)
	wc: word count	wc(1)
signal: specify	what to do upon receipt..................................	signal(2)
whodo: who is doing	what..	whodo(1M)
whodo:	who is doing what..	whodo(1M)
who:	who is on the system	who(1)
	who: who is on the system.............................	who(1)
	whodo: who is doing what...............................	whodo(1M)
streamdrv: copy	with buffering..	streamdrv(1)
csh: a shell	with C-like syntax...	csh(1)
dsh: shell	with history facility...	dsh(1)
prof: profile	within a function...	prof(5)
wc:	word count...	wc(1)
getc, getchar, fgetc: get character or	word from stream	getc(3S
fgetc, getw: get character or	word from stream	getc(3S)
putc, putchar: put character or	word on a stream	putc(3S)
fputc, putw: put character or	word on a stream	putc(3S
cd: change	working directory...	cd(1)
chdir: change	working directory...	chdir(2)
getcwd: get path-name of current	working directory...	getcwd(3C)
pwd: print	working directory name	pwd(1)
errout:	write an error message......................................	errout(3C)
awrite: asynchronous	write	awrite(2)

sync:	write disk cache to disk	sync(2)
putpwent:	write password file entry	putpwent(3C)
wall:	write to all users	wall(1)
write:	write to another user	write(1)
write, write_t:	write to file	write(2)
	write: write to another user	write(1)
	write, write_t: write to file	write(2)
write	write_t: write to file	write(2)
utmp, wtmp: utmp and	wtmp entry formats	utmp(4)
utmp	wtmp: utmp and wtmp entry formats	utmp(4)
fwtmp	wtmpfix: manipulate connect accounting	fwtmp(1M)
	xargs: construct argument lists and	xargs(1)
j0, j1, jn	y0, y1, yn: Bessel functions	bessel(3M)
j0, j1, jn, y0	y1, yn: Bessel functions	bessel(3M)
	yacc: yet another compiler-compiler	yacc(1)
yacc:	yet another compiler-compiler	yacc(1)
j0, j1, jn, y0, y1	yn: Bessel functions	bessel(3M)
timezone: set default system time	zone	timezone(4)

NAME

intro – introduction to commands

DESCRIPTION

This section describes publicly accessible commands of *Essential Utilities* in alphabetic order. The *Essential Utilities* are programs intended to be invoked directly by the user or by command language procedures, as opposed to subroutines, which are called by the user's programs.

The commands generally resides in the directory **/bin** or **/usr/bin**. These directories are searched automatically by the command interpreter **shell**.

Some commands are marked (**1M**) and will often be found in the directory **/etc**. These (**1M**) commands are primarily intended for the system administrator.

This page is intentionally left blank

NAME

accept, reject – allow or prevent LP requests

SYNOPSIS

/usr/lib/accept destinations
/usr/lib/reject [**−r**[reason]] destinations

DESCRIPTION

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

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

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

FILES

/usr/spool/lp/ *

SEE ALSO

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

This page is intentionally left blank

NAME

apropos – locate commands by keyword lookup

SYNOPSIS

/usr/bin/apropos keyword ...

DESCRIPTION

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

Try

```
apropos password
```

and

```
apropos editor
```

If the line starts '*filename (section)* ...' you can do '**man** *section filename*' to get the documentation for it. Try

```
apropos format
```

and then

```
man 3s printf
```

to get the manual page on the subroutine **printf**.

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

FILES

/usr/man/whatis	data base

SEE ALSO

man(1), whatis(1).

This page is intentionally left blank

NAME

at, batch − execute commands at a later time

SYNOPSIS

at *time* [*date*] [**+** *increment*]
at −*r* job...
at −*l* [*job* ...]

batch

DESCRIPTION

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

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

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

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

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

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

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

Thus legitimate commands include:

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

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

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

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

EXAMPLES

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

This sequence can be used at a terminal:

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

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

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

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

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

75

FILES

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

SEE ALSO

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

DIAGNOSTICS

Complains about various syntax errors and times out of range.

This page is intentionally left blank

NAME

awk − pattern scanning and processing language

SYNOPSIS

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

DESCRIPTION

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.

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).

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 new-line)
RS	input record separator (default new-line)

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

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

Statements are terminated by semicolons, new-lines, or 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) in the *Reference Manual*].

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.

80

split(*s*, *a*, *fs*)	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 }
```

82

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), oawk(1), sed(1) and printf(3S).

Programmer's Guide.

83

BUGS

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

This page is intentionally left blank

NAME

banner – make posters

SYNOPSIS

banner strings

DESCRIPTION

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

SEE ALSO

echo(1).

85

This page is intentionally left blank

NAME

basename, dirname – deliver portions of path names

SYNOPSIS

basename string [suffix]
dirname string

DESCRIPTION

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

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

EXAMPLES

The following example, invoked with the argument **/usr/src/cmd/cat.c**, compiles the named file and moves the output to a file named **cat** in the current directory:

```
cc $1
mv a.out basename $1 \.c
```

The following example will set the shell variable **NAME** to **/usr/src/cmd**:

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

SEE ALSO

sh(1).

This page is intentionally left blank

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 as follows:

–c Compile only. The output is send to the standard output.

–l Argument stands for the name of an arbitrary precision math library.

The syntax for *bc* programs is as follows; L means letter a–z, E means expression, S means statement.

Comments
  are enclosed in /* and */.

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

Other operands
  arbitrarily long numbers with optional sign and decimal point.
  (E)
  sqrt (E)

length (E)	number of significant decimal digits
scale (E)	number of digits right of decimal point

  L (E , ... , E)

Operators

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

Statements

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

Function definitions

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

Functions in −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 | |.

The *for* statement must have all three expressions (E's).

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

NAME

bcpio – cpio with buffering

SYNOPSIS

bcpio –o[**acvV**] [**–M** *message*] [**–O** *file*]

bcpio –i[**cdmrtuvVfsSb6k**] [**–M** *message*] [**–I** *file*] [*pattern* ...]

DESCRIPTION

bcpio is a shell script setting up **cpio** and **streamdrv** to make **cpio** read and write through **streamdrv**, using standard output and standard input pipelines. *bcpio* is mostly meant for storing and restoring files on streamer tapes, where it is important to write and read data in as big chunks as possible, which **streamdrv** takes care of.

The function of the options is described in the documentation of **cpio(1)**, except the option **M**. The meaning of this options is as follows:

–M *message*	Define a message to use when switching media. When you specify a character special device as input or output 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.

EXAMPLE:

```
find /usr –print | bcpio –oacv –O /dev/stream \
–M"Insert tape no %d"
```

will copy the files in **/usr** subdirectories to **/dev/stream**. If the file archive exceeds the size of the physical medium the user is prompted to insert a new tape.

SEE ALSO

streamdrv(1), cpio(1)

NOTE

The *bcpio* and the *cpio* utilities do not write on streamer tape or floppies using exactly the same format. This means that these utilities will not always produce media readable for each other.

NAME

bdiff − big diff

SYNOPSIS

bdiff file1 file2 [n] [**−s**]

DESCRIPTION

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

The parameters to *bdiff* are:

file1 (file2)	The name of a file to be used. If *file1 (file2)* is −, the standard input is read.
n	The number of line segments. The value of *n* is 3500 by default. If the optional third argument is given and it is numeric, it is used as the value for *n*. This is useful in those cases in which 3500-line segments are too large for *diff*, causing it to fail.
−s	Specifies that no diagnostics are to be printed by *bdiff* (silent option). Note, however, that this does not suppress possible diagnostic messages from *diff*(1), which *bdiff* calls.

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

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

FILES

/tmp/bd?????

SEE ALSO

diff(1).

96

NAME

bfs – big file scanner

SYNOPSIS

bfs [–] name

DESCRIPTION

The *bfs* command is (almost) like *ed*(1) except that it is read-only and processes much larger files. Files can be up to 1024K bytes and 32K lines, with up to 512 characters, including new-line, per line (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:

97

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

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

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

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

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

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

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

```
xb/^/ label
```

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

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

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

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

will all print the first 100 lines.

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

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

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

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

Another feature of the **xv** command is that the first line of output from a UNIX system command can be stored into a variable. The only 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 UNIX system command (**!***command*) or nonzero value, respectively, to the specified label. The two examples below both search for the next five lines containing the string **size**.

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

```
xv4!expr %4 - 1
!if 0%4 = 0 exit 2
xbz 1
```

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.

This page is intentionally left blank

NAME

boot – reboot system from an available boot disk

SYNOPSIS

/etc/boot [−*x*]

DESCRIPTION

The *boot* program is used for rebooting the system, if convenient with non-operator test programs.

The Supermax boot system supports up to 4 different boot disks numbered from 0 to 3. These boot disks should be allocated as subdisks on the same physical disk as the root disk.

The optional parameter **−x** to *boot* is a number from 0 to 3, specifying from which of the 4 bootdisks the system should be loaded.

If no boot disk number is specified the system will be loaded from the last used boot disk.

103

SEE ALSO

mkwboot(1M).

This page is intentionally left blank

NAME

bootgen – generate a boot device

SYNOPSIS

/etc/bootgen [**–dlic**] device [files]

DESCRIPTION

bootgen is used to inspect, initialize, and update a boot disk. The *device* argument is the name of a (special) file identifying the boot device. Normally, this will be a floppy disk or a partition on a hard disk set aside for that purpose by *mkwboot*(1M). The following flags may be specified:

d	Display boot device information (short form).
l	Display boot device information (long form).
i	Initialize the boot device.
c	Add the named *files* to the boot disk.

With the **–c** option *bootgen* loads onto the boot device the files that are to be booted into the computer. The last component of the path name for each file must consist of one of the following names, possibly followed by a period and extra characters:

cioc	for the software to be loaded into CIOCs.
config	for the hardware configuration prepared with *chhw*(1M).
dioc	for the software to be loaded into DIOCs.
dioc2	for the software to be loaded into DIOC2s.
dioc3	for the software to be loaded into DIOC3s.
nioc	for the software to be loaded into NIOCs.
os00	for the software to be loaded into MCUs with MC68000 processors.
os20	for the software to be loaded into MCUs with MC68020 processors.

os30	for the software to be loaded into MCUs with MC68030 processors.
sioc	for the software to be loaded into SIOCs.
sioc2	for the software to be loaded into SIOC2s.

Thus **/use/os/myos/nioc.x1** is a valid name for software to be loaded into NIOCs, but **/use/os/myos/mynioc.x1** is not.

If *bootgen* is requested to place, for example, an *os00* file on a boot disk, and an *os00* file is already present on the boot device the old *os00* file is replaced by the new one.

It is strongly recommended that *dioc* files be the first ones specified, as their position on the boot disk is critical. *bootgen* will issue an error if it cannot place a *dioc* file where it should.

SEE ALSO

chhw(1M), makeos(1M), mkwboot(1M).

NAME

brc, bcheckrc – system initialization procedures

SYNOPSIS

/etc/brc

/etc/bcheckrc

DESCRIPTION

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

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

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

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

Note that *bcheckrc* should always be executed before *brc*. Also, these shell procedures may be used for several run-level states.

SEE ALSO

fsck(1M), init(1M), rc2(1M), shutdown(1M).

This page is intentionally left blank

NAME

btar – tar with buffering

SYNOPSIS

btar [key] [files]

DESCRIPTION

btar is a shell script setting up tar and streamdrv, to make tar read and write through streamdrv, using standard output and standard input pipelines. *btar* is mostly meant for storing and restoring files on streamer tapes, where it is important to write and read data in as big chunks as possible, which streamdrv takes care of.

The following *tar* options are available **t, x, c, v, f, d, m.** The function of the options is described in the documentation of *tar*(1).

EXAMPLE

```
          btar cfvd /dev/stream 10102200 /usr \
                  /usr1  2> /tmp/log
```

will copy the files in /usr and /usr1 newer than Oct 10 22:00 this year, to /dev/stream and make a /tmp/log file of the files copied.

SEE ALSO

streamdrv(1), tar(1)

NOTE

The *btar* and the *tar* utilities do not write on streamer tape or floppies using exactly the same format. This means that these utilities will not always produce media readable for each other.

This page is intentionally left blank

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.

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

This page is intentionally left blank

NAME

calendar – reminder service

SYNOPSIS

calendar [–]

DESCRIPTION

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

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

FILES

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

SEE ALSO

mail(1).

BUGS

Your calendar must be public information for you to get reminder service.

calendar's extended idea of 'tomorrow' does not account for holidays.

This page is intentionally left blank

NAME

captoinfo – convert a termcap description into a terminfo description

SYNOPSIS

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

DESCRIPTION

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

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

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

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

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

–w change the output to *width* characters.

FILES

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

CAVEATS

Certain *termcap* defaults are assumed to be true. For example, the bell character (*terminfo bel*) is assumed to be *^G*. The linefeed capability (*termcap nl*) is assumed to be the same for both *cursor_down* and *scroll_forward* (*terminfo cud1* and *ind*, respectively.) Padding information is assumed to belong at the end of the string.

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

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

DIAGNOSTICS

tgetent failed with return code n (reason).

The termcap entry is not valid. In particular, check for an invalid 'tc=' entry.

unknown type given for the termcap code *cc*.

The termcap description had an entry for *cc* whose type was not boolean, numeric or string.

wrong type given for the boolean (numeric, string) termcap code *cc*.

The boolean *termcap* entry *cc* was entered as a numeric or string capability.

the boolean (numeric, string) termcap code *cc* is not a valid name.

An unknown *termcap* code was specified.

tgetent failed on TERM=term.

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

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

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

a function key for *cc* was specified, but it already has the value *vv*.
: When parsing the **ko** capability, the key *cc* was specified as having the same value as the capability *cc*, but the key *cc* already had a value assigned to it.

the unknown termcap name *cc* was specified in the **ko** termcap capability.
: A key was specified in the **ko** capability which could not be handled.

the *vi* character *v* (**info** *ii*) has the value *xx*, but **ma** gives *n*.
: The **ma** capability specified a function key with a value different from that specified in another setting of the same key.

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

Warning: *termcap* **sg** (*nn*) and *termcap* **ug** (*nn*) had different values.
: *terminfo* assumes that the **sg** (now **xmc**) and **ug** values were the same.

Warning: the string produced for *ii* may be inefficient.
: The parameterized string being created should be rewritten by hand.

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

cannot open *file* for reading.
The specified file could not be opened.

SEE ALSO

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

Chapter 10 in the *Programmer's Guide*.

NOTES

captoinfo should be used to convert *termcap* entries to *terminfo*(4) entries because the *termcap* database (from earlier versions of UNIX System V) may not be supplied in future releases.

NAME

cat – concatenate and print files

SYNOPSIS

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

DESCRIPTION

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

cat file

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

When used with the **–v** option, the following options may be used:

–t Causes tabs to be printed as ^**I**'s.

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

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

WARNING

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

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.

121

SEE ALSO

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

This page is intentionally left blank

NAME

checkfsys – check a file system on a removable disk

SYNOPSIS

The *checkfsys* command allows the user to check for and optionally repair a damaged file system on a removable disk.

The user is asked one of the following three functions:

1. check the file system
   No repairs are attempted.
2. repair it interactively
   The user is informed about each instance of damage and asked if it should be repaired.
3. repair it automatically
   The program applies a standard repair to each instance of damage.

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

sysadm checkfsys

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

WARNING

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

If losing data is of particular concern, "check" the file system first to discover if it appears to be damaged. If it is damaged, use one of the repair mechanisms or the file system debugging utility, **fsdb**.

SEE ALSO

fsck(1M), fsdb(1M), makefsys(1M), mountfsys(1M), sysadm(1).

This page is intentionally left blank

NAME

chhw – change system configuration

SYNOPSIS

/etc/chhw [file]

DESCRIPTION

The *chhw* program is used for editing the system configuration description on a boot disk or in a file that is to be used with the *bootgen*(1M) command.

The *file* argument, if present, must be either the name of the special file identifying the boot disk or the name of the file that is to be used with *bootgen*(1M). If the *file* argument is present, *chhw* will load an initial configuration from that file, and the file will be the default file name used with the **load** and **save** subcommands.

Once a configuration is loaded, either because a *file* argument was specified, or through the execution of the **load** command, the user may display and change the configuration.

If the boot disk is a floppy disk, it may be removed and replaced by another boot floppy before the possibly changed system configuration is written to the disk. In this way a system configuration may be copied from one boot floppy to another.

chhw will prompt for commands with an asterisk.

In the command descriptions below, a text such as "SIOC number 8" is to be interpreted as "CPU unit number 8 which is a SIOC".

Legal Commands

abort	will terminate *chhw* without storing the new system configuration. End-of-file is equivalent to **abort**.
check	will make a consistency check of the system configuration.

cioc <*unit number*> [<*channel number*> – <*spec*>] . . .

adds, removes, or changes CIOC communication devices. <*unit number*> must either be a CIOC or undefined, in which case it will be defined as a CIOC. <*spec*> is **0** to disable an already enabled channel, and **1** to enable the channel. For example,

cioc 7 0 – 1 1 – 1

specifies that unit number 7 is a CIOC and channels 0 and 1 are enabled.

cmd [<*file*>]

prints the configuration in form of commands that may later be used as input to *chhw*. If <*file*> is present the commands will be written to that file, otherwise the commands will be written to the standard output.

command [<*file*>]

prints the configuration is the same way as **cmd**, but with an initial **reset** and final **save** and **quit** commands.

console <*unit number*> <*channel number*> <*spec*>

specifies the console terminal, that is, the terminal on which system messages should be displayed during bootstrapping or a system crash. <*spec*> is any of the following:

b – <*baudrate*>

specifies the baud rate for the console. The legal values are: **600, 1200, 1800, 2400, 4800, 9600, 19200.**

d – <*data bits*>

specifies the number of data bits. The legal values are **5, 6, 7, 8.**

s – <*stop bits*>	specifies the number of stop bits. The legal values are **1** and **2.**
p – <*parity*>	specifies the parity. The legal values are **0** for parity check disabled, **1** for odd parity, **2** for even parity and
r	resets the characteristics for the console.

For example,

console 8 0 r	sets the console to be located at SIOC number 8, channel 0, and the characteristics for the terminal to be: 9600 baud, parity even, 7 data-bits and 2 stop bits. See the **SIOC** command for a description of the channel numbers.

defstreams

enables the *streams* commands described below under the heading "Streams Commands". When the streams commands are enabled, the **cmd, command, display,** and **list** commands will include information about the streams in the system.

defulimit <*number of 512 byte blocks*>

set the *ulimit* for the system. Default is 2048 blocks.

delete <*unit number*>

specifies that the indicated unit (that is, an MCU, a SIOC, etc.) should be deleted from the configuration. For example,

delete 9	will cause CPU number 9 to be removed from the configuration.

delstreams

disables the *streams* commands described below under the heading "Streams Commands". When the streams commands are disabled, the **cmd, command, display,** and **list** commands will not include information about the streams in the system.

dioc <*unit number*> [<*type*> <*channel number*> – <*spec*>]

adds, removes, or changes disks on a DIOC. <*unit number*> must either be a DIOC or undefined, in which case it will be defined as a DIOC. <*type*> can be either **d** for disk channels, or **t** for tty channels. The channel numbers for disks are:

0	Reserved
1	First 1 megabyte 8″ floppy disk.
2	Second 1 megabyte 8″ floppy disk.
3	First 560 kilobyte 5¼″ floppy disk.
4	Second 560 kilobyte 5¼″ floppy disk.
5	First IBM compatible 8″ floppy disk.
6	Second IBM compatible 8″ floppy disk.
7	Streamer tapes.
8	First hard disk on first controller.
9	Second hard disk on first controller.
10	First hard disk on second controller.
11	Second hard disk on second controller.
12	First hard disk on third controller.
13	First hard disk on fourth controller.
14	First hard disk on fifth controller.
15	First hard disk on sixth controller.
16	Magtape, videostreamer or optical disk, (Variabel Block Mode).
17	Magtape, videostreamer or optical disk, (Fixed Block Mode).

20	First 360 kilobyte PC-DOS compatible 5¼″ floppy disk (40 tracks, 9 sectors per track).
21	Second 360 kilobyte PC-DOS compatible 5¼″ floppy disk (40 tracks, 9 sectors per track).
22	First 720 kilobyte PC-DOS compatible 5¼″ floppy disk (80 tracks, 9 sectors per track).
23	Second 720 kilobyte PC-DOS compatible 5¼″ floppy disk (80 tracks, 9 sectors per track).
24	First 320 kilobyte X/OPEN compatible 5¼″ floppy disk (40 tracks, 8 sectors per track).
25	Second 320 kilobyte X/OPEN compatible 5¼″ floppy disk (40 tracks, 8 sectors per track).
26	First 640 kilobyte X/OPEN compatible 5¼″ floppy disk (80 tracks, 8 sectors per track).
27	Second 640 kilobyte X/OPEN compatible 5¼″ floppy disk (80 tracks, 8 sectors per track).

For disks <*spec*> is **0** to disable an already enabled channel, and **1** to enable the channel. For streamer tapes, <*spec*> may be either **1** (for short tapes, 20 MB), **2** (for medium size tapes, 45 MB), (for long tapes, 130 MB), **4** (for 60 MB tapes), **5** (for 150 MB tapes), or **6** (for 320 MB tapes). For magtapes and optical disks, <*spec*> may have the following values:

0	Not installed.
1	45 MB on channel 16.
	90 MB on channel 17.
2...	Megabytes.

For ttys <*spec*> is the same as for the **sioc**-command. For example,

dioc 13 t1-1	enables channel 1 (tty) on DIOC 13.
dioc 13 d3-1 t3-1	enables channel 3 (disk) and channel 3 (tty) on DIOC 13.

dioc3 <*unit number*> [<*channel number*> – <*spec*>] . . .
adds, removes, or changes disks on a DIOC3. <*unit number*> must either be a DIOC3 or undefined, in which case it will be defined as a **DIOC3**.

The channel numbers for disks are:

0	For future use.
1	For future use.
2 – 15	Hard disks. (See the table page 7).
28	Mirror hard disk; channel 8 and 9.
29	Mirror hard disk; channel 10 and 11.
30	Mirror hard disk; channel 12 and 13.
31	Mirror hard disk; channel 14 and 15.

<*spec*> is a textstring:

flop	to enable floppy disks
hard	to enable hard disks
mirror	to enable mirror hard disks
dis	to disable channel.

Channel Number	SCSI Interface	SCSI id.
2	0	4
3	1	4
4	0	5
5	1	5
6	0	6
7	1	6
8	0	0
9	1	0
10	0	1
11	1	1
12	0	2
13	1	2
14	0	3
15	1	3

display displays various configuration information.

dumpdisk <*unit number*> <*channel number*>
specifies the physical disk on which a system crash dump should be generated. For example,

dumpdisk 14 1 sets the crash dump disk to be located at DIOC number 14, channel 1.

files <*number of files*>
specifies the number of simultaneously open files in the system. For example,

files 30 will allow 30 simultaneously open files.

globalp <*number of processes*>
allocates room for the global part of process control blocks. For example,

globalp 100	will allocate room for the global part of process control blocks for 100 processes.

help displays the legal commands.

hypcache <*size of cache*>
sets the size in kilobytes of the hyper disk cache in the MCU. Size must be 0 or a power of 2 in the range 256 to 4096.

init <*initial program*>
specifies the program that is to be executed in the master MCU after bootstrapping. A maximum of 32 characters are allowed in the file name. File names not starting with / will be sought in the root directory. For example,

init /etc/init	will cause /etc/init to be executed in the master-MCU after bootstrapping.

list [<*param*>] . . .
Displays a part of the information, like **display**. <*param*> is any of the following:

(No argument)	The same as **display**.
c	Displays CIOC information
d	Displays DIOC information
g	Displays global information
m	Displays MCU information
n	Displays NIOC information

s	Displays SIOC information
S	Displays streams information (if enabled)

load [<*file*>]
will load a system configuration. If <*file*> is specified, the configuration will be loaded from that file. Otherwise the configuration will be loaded from the file whose name was given as argument to the invocation of *chhw*, if any. The **load** command is able to determine if the file is a boot device or a file that is intended for a later *bootgen*(1M) command.

locks <*number of lock elements*>
specifies the number of record locking elements in the system. For example,

lock 100	will allocate 100 record locking elements.

master <*MCU number*>
specifies the master MCU. This is the MCU that will contain the common data area. For example,

master 2	specifies CPU 2 to be the master MCU.

maxio <*number of file descriptors per process*>
sets the number of file descriptors that a process may open. If the number is set to a value less than 32, the operating system will use 32. If the number is set to a value greater than 128, the operating system will use 128.

mcu <*unit number*> [<*spec*>] . . .
specifies the parameters for an MCU. <*unit number*> must either be an MCU or undefined, in which case it will be defined as an MCU.
<*spec*> is any of the following:

m – <*megabytes*>	specifies the maximum allowed memory per process. This number must be given with a decimal point; thus 1 megabyte must be specified as **1.0**.
l – <*number*>	specifies the number of local process control blocks.
t – <*number*>	specifies the number of text-descriptors (different programs).
p – <*number*>	specifies the number of partition descriptors.
i – <*kilobytes*>	specifies the size of item area.
s – <*dioc*> / <*channel*> / <*subdisk*>	specifies the swap disk.
s – 0	no swapping.

messages <*number of message queues*>
allocates room for message queues. For example,

messages 2	will allocate room for 2 message queues.

nfsmounts <*number of nfs filesystems*>
specifies the maximum number of simultaneously mounted NFS filesystems.

nioc <*unit number*> [<*channel number*> – <*spec*>] . . .
adds, removes, or changes tty channels on a NIOC. The parameters have the same significance as with the **sioc** command (see this). There are 32 channels (numbered 0-31) on a NIOC.

opens <*number of openings*>
specifies the maximum number of simultaneously active *open* operation on files. Every time an *open*(2) or a *creat*(2) system call is executed, one

opening is used. The *opening* is released, when the last *close*(2) on that file descriptor and any derived file descriptor is issued. For example,

opens 300	will allocate room for 300 openings.

pwbacktime <*minutes*>
sets the alternative backup power time, i.e. the time the system uses the backup power before the power fail signal is send to the init process. */etc/inittab* tells *init* what to do in case of power failure. Typically shutdown is executed.

quit is equivalent to **abort**, but without writing a message.

reset This command will reset the configuration as follows: Master MCU: 3, files: 20, locks: 2, global processes: 30, 30 opens, initial program: /etc/init, all other parameters: 0.

root <*dioc number*> <*channel number*> \
[<*subdisk number*>]
specifies the root disk. For example,

root 14 8 0	The root is located on subdisk 0, channel 8 on dioc 14.
root 13 1	The location of the root is: channel 1 on dioc 13.

save [<*file*>]
will store the system configuration. First a configuration consistency check is made (see the **check** command); if this check is unsuccessful the user will be asked if he wants to save the configuration anyway (if the input device is not a terminal, **save** will never save an inconsistent configuration).

If the configuration is indeed to be saved, **save** proceeds as follows:

If <*file*> is specified, the configuration will be stored in that file. Otherwise the configuration will be stored in the file whose name was given as argument to the invocation of *chhw*, if any. The **save** command checks if the file already exists, and if it does, **save** determines if it is a boot device or a file that is intended for a later *bootgen*(1M) command. Then the system configuration is written onto the file in the relevant format.

sema <*number of semaphore identifiers*>

specifies the number of semaphore identifiers. For example,

sema 14	allocates 14 semaphore identifiers.

shared <*number of shared memory identifiers*>

allocates room for shared memory identifiers. For example,

shared 14	allocates 14 shared memory identifiers.

sioc <*unit number*> [<*channel number*> – <*spec*>] . . .

adds, removes, or changes tty channels on a SIOC. <*Unit number*> must either be a SIOC or undefined, in which case it will be defined as a SIOC.

<*spec*> is:

0 – disable an already enabled channel.

<*number of windows*>

– enable channel with the specified number of windows.

<*number of windows*> **a**

– enable with associated printer.

For example,

sioc 8 3-1 4-6 5-1a
: specifies that unit 8 is a SIOC with channel 3 enabled for a normal tty, channel 4 enabled for a window terminal with 6 windows, channel 5 enabled for a normal terminal with an associated printer.

Channels 0-7 are the serial input/output channels on the SIOC. Channel 8 is the parallel input/output channel.

sioc2 <*unit number*> [<*channel number*> – <*spec*>] . . .
: adds, removes, or changes tty channels on a SIOC2. The parameters have the same significance as with the **sioc** command (see this). There are 33 channels (numbered 0-32) on a SIOC2.

Streams Commands

If the operating system is equipped with the streams mechanism a number of extra configurable parameters exist. The following streams commands are enabled if the **load** command has loaded a system where the number of message blocks in the configuration is non-zero (see the **strmsize** command), or if the **defstreams** command has been executed.

The streams commands are:

strdef <*module name*> [<*parameter*>] . . .
: defines a stream module name. The module may be given up to 4 parameters. This command informs the operating system that the specified module is present and that its initialization routines is to be called with the specified parameters. The significance of each parameter depends on the module and is specified on the relevant pages of section 7 of this manual.

strevent <*number of stream event cells*>
: specifies the number of stream event cells. Stream event cells are used for recording process-specific information in the *poll*(2) system call. They are also used in the implementation of the streams I_SETSIG *ioctl*(2) calls and in the operating system streams scheduling mechanism. A minimum value to configure would be the expected number of processes to be simultaneously using *poll*(2) times the expected number of streams being polled per process, plus the expected number of processes expected to be using streams concurrently.

strmcnt <*number of message blocks*>
: specifies the number of streams message blocks to be allocated. This number should be at least twice the number of processes expected to be using streams concurrently, and probably considerably greater.

strlowf <*percentage*>
: and

strmedf <*percentage*>
: set the low and medium fraction for message block allocation. These numbers are the percentage of data blocks of a given size at which low or medium priority block allocation requests in the operating system fail. Sensible values are 80 and 90, respectively. For example,

strlowf 80

strmedf 90 All requests will be honored if less than 80% of the message blocks are used. Medium priority requests will be honored if between 80% and 90% are used and only high priority requests will be honored if more than 90% of the message blocks are used.

strmsize <*message block size*> <*no of message blocks*>
: allocates the specified number of message blocks of the given size. The message block sizes available are 4, 16, 64, 128, 256, 512, 1024, 2048, and 4096. For example,

strmsize 4 10	allocates 10 blocks of 4 bytes each.
strmsize 256 20	allocates 20 blocks of 256 bytes each.

strmul <*number of stream multiplexer links*>
: specifies the number of multiplexer links available. One link structure is required for each active multiplexer link (as set up by the streams I_LINK *ioctl*(2) call). This number is application dependent.

strqpair <*number of stream queue pairs*>
: specifies the number of stream queue pairs available. Each time a stream is opened, two queue pairs are used. Whenever a module is pushed onto a stream, an extra queue pair is used.

strrm <*module name*>
: Removes a stream module defined with **strdef**.

Additional Information

All numbers in the commands are decimal.

Commands may be abbreviated as long as they remain unambiguous. Thus **command** may be abbreviated to **com**, but not to **co** as it would then be indistinguishable from the **console** command.

The **help** command displays all the legal commands. It does not, however, display the parameters of each command. The parameter format can be found by omitting the parameters to a command. For example, merely giving the command **sioc** without parameters, will make *chhw* display the legal parameter formats.

chhw accepts command editing in a manner identical to that used in *dsh*(1). The reader is referred to the manual page on *dsh*(1) for further information.

SEE ALSO

bootgen(1M), chlds(1M), config(1M).

NAME

chlds – change logical disk size

SYNOPSIS

chlds

DESCRIPTION

The hard disks on a Supermax computer may be partitioned into sub-disks. The *chlds* program is used to edit the sub-disk configuration of a hard disk. The program will prompt the user for a unit number and a channel number to identify a hard disk. (*chhw*(1M)).

Initially *chlds* reads the physical size of the hard disk and its current sub-disk configuration. The user is now allowed to change the configuration. The edited configuration will not be saved on the hard disk before the user explicitly asks for it.

chlds will prompt for commands with an asterisk. Pressing the 'Restore' function key will cause the last command to be displayed, whereupon the user may edit it.

Legal commands:

abort	will terminate *chlds* without storing the new sub-disk configuration. End-of-file is equivalent to **abort**.
check	will make a consistency check of the sub-disk configuration. This check makes sure that the total size of the sub-disks does not exceed the physical size of the hard disk.
clear	will delete all sub-disks.
display	This command may be used to display the sub-disk configuration as edited by the user.
end	will store the sub-disk configuration and terminate *chlds*. First a consistency check is made (see the check command); if this check is unsuccessful the user will be asked if he wants to save the configuration anyway. If the user answers

yes, the changed configuration is written onto the hard disk, and *chlds* terminates. If the user does not want to save the configuration he may continue editing it.

help	This command will display the legal commands.
quit	will terminate *chlds* without storing the new sub-disk configuration.
readold	will read the old sub-disk configuration from the hard disk. This configuration can be displayed with the **display** command and modified by the **clear** or **subdisk** commands.
subdisk	adds or changes a sub-disk. The format of this command is as indicated by the following examples:

subdisk 3 17M	defines sub-disk number 3 to be of size 17Mb (= 17825792 bytes decimal)
subdisk 3 2.5M	defines sub-disk number 3 to be of size 2.5Mb (= 2621440 bytes decimal)
subdisk 3 1048576	defines sub-disk number 3 to be of size 1048576 bytes decimal. The size is adjusted so the disk size will be the lowest multiple of 2048 bytes greater than the specified number.

The **help** command displays all the legal commands. It does not, however, display the parameters of the **subdisk** command. The parameter format can be found by omitting the parameters to the command: Merely giving the command **subdisk** without parameters, will make *chlds* display the legal parameter formats.

142

The new sub-disk configuration will not be effective before the Supermax computer has been re-booted.

SEE ALSO

chhw(1M), config(1M), dsize(1).

This page is intentionally left blank

NAME

chmod – change mode

SYNOPSIS

chmod mode file ...

chmod mode directory ...

DESCRIPTION

The permissions of the named *files* or *directories* are **ch**anged 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

145

themselves. Permissions to a file may vary depending on your user identification number (UID) or group identification number (GID). Permissions are described in three sequences each having three characters:

User	Group	Other
rwx	rwx	rwx

This example (meaning that **u**ser, **g**roup, and **o**thers all have **r**eading, **w**riting, and e**x**ecution permission to a given file) demonstrates two categories for granting permissions: the access class and the permissions themselves.

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

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

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

chmod g+rw *file*

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

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

u	**u**ser's permissions
g	**g**roup's permissions
o	**o**thers permissions

The letter **a** (**a**ll) 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,

```
chmod g+x,+l file

chmod g+s,+l file
```

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

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

EXAMPLES

chmod a−x *file*

chmod 444 *file*

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

chmod go+rw *file*

chmod 606 *file*

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

chmod +l *file*

This causes a file to be locked during access.

chmod =rwx,g+s *file*

chmod 2777 *file*

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

NOTES

In a Remote File Sharing environment, you may not have the permissions that the output of the **ls −l** command leads you to believe. For more information see the "Mapping Remote Users" section of Chapter 10 of the *System Administrator's Guide*.

SEE ALSO

ls(1), chmod(2).

NAME

chown, chgrp − change owner or group

SYNOPSIS

chown owner file ...

chown owner directory ...

chgrp group file ...

chgrp group directory ...

DESCRIPTION

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

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

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

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

FILES

/etc/passwd
/etc/group

NOTES

In a Remote File Sharing environment, you may not have the permissions that the output of the **ls −l** command leads you to believe. For more information see the "Mapping Remote Users" section of Chapter 10 of the *System Administrator's Guide*.

SEE ALSO

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

This page is intentionally left blank

NAME

chroot – change root directory for a command

SYNOPSIS

/etc/chroot newroot command

DESCRIPTION

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

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

```
chroot newroot command >x
```

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

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

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

SEE ALSO

cd(1), chroot(2).

BUGS

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

This page is intentionally left blank

NAME

chrtbl – generate character classification and conversion tables

SYNOPSIS

chrtbl [file]

DESCRIPTION

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

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

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

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

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

chrclass	name of the data file to be created by *chrtbl*.
isupper	character codes to be classified as upper-case letters.
islower	character codes to be classified as lower-case letters.
isdigit	character codes to be classified as numeric.
isspace	character codes to be classified as a spacing (delimiter) character.
ispunct	character codes to be classified as a punctuation character.

iscntrl	character codes to be classified as a control character.
isblank	character code for the space character.
isxdigit	character codes to be classified as hexadecimal digits.
ul	relationship between upper- and lower-case characters.

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

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

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

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

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

EXAMPLE

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

```
chrclass  ascii
isupper   0x41 - 0x5a
islower   0x61 - 0x7a
isdigit   0x30 - 0x39
isspace   0x20 0x9 - 0xd
ispunct   0x21 - 0x2f 0x3a - 0x40    \
          0x5b - 0x60 0x7b - 0x7e
iscntrl   0x0 - 0x1f  0x7f
isblank   0x20
isxdigit  0x30 - 0x39 0x61 - 0x66    \
          0x41 - 0x46
ul       <0x41 0x61> <0x42 0x62> <0x43 0x63>  \
         <0x44 0x64> <0x45 0x65> <0x46 0x66>  \
         <0x47 0x67> <0x48 0x68> <0x49 0x69>  \
         <0x4a 0x6a> <0x4b 0x6b> <0x4c 0x6c>  \
         <0x4d 0x6d> <0x4e 0x6e> <0x4f 0x6f>  \
         <0x50 0x70> <0x51 0x71> <0x52 0x72>  \
         <0x53 0x73> <0x54 0x74> <0x55 0x75>  \
         <0x56 0x76> <0x57 0x77> <0x58 0x78>  \
         <0x59 0x79> <0x5a 0x7a>
```

FILES

/lib/chrclass/ *	data file containing character classification and conversion tables created by *chrtbl*
/usr/include/ctype.h	header file containing information used by character classification and conversion routines.

SEE ALSO

ctype(3C), environ(5).

DIAGNOSTICS

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

This page is intentionally left blank

NAME

chstack – set load module stack size

SYNOPSIS

chstack [option] file

DESCRIPTION

chstack is used to inspect or change the size of the stack allocated for a load module. The load module file name is specified by the file parameter.

The options are:

–s stacksize	the new stack size is specified by the **–s** parameter. The stack size is a number indicating the number of bytes in the stack. This number must be even. If the first two characters of the stack size are 0x the number is hexadecimal, otherwise if the first character is 0 the number is octal, otherwise the number is decimal.
–v	the stacksize is displayed. **–v** is default.
–c	a command line for setting the stack to the actual value is generated.

The default stack size for a program is 6K.

NOTE

It is only necessary to use the *chstack* program on programs that are to be run on Supermax Computers without automatic stack growth. Further, *chstack* need not be run on programs that require less than 6K stack.

This page is intentionally left blank

NAME

ckbupscd – check file system backup schedule

SYNOPSIS

/etc/ckbupscd [**–m**]

DESCRIPTION

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

The System Administration commands *bupsched/schedcheck* are provided to review and edit the **/etc/bupsched** file.

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

time[,**time**] **day**[,**day**] **month**[,**month**] **fsyslist**

where:

time	Specifies an hour of the day (*0* through *23*), matching any time within that hour, or an exact time of day (*0:00* through *23:59*).
day	Specifies a day of the week (*sun* through *sat*) or day of the month (*1* through *31*).
month	Specifies the month in which the time and day fields are valid. Legal values are the month numbers (*1* through *12*).
fsyslist	The rest of the line is taken to be a file system list to print.

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

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

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

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

EXAMPLES

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

06:00-09:00 fri 1,2,3,4,5,6,7,8,9,10,11 /applic
: Prints the file system name */applic* if *ckbupscd* is run between 6:00am and 9:00am any Friday during any month except December.

00:00-06:00,16:00-23:59 1,2,3,4,5,6,7 1,8 /
: Prints a reminder to backup the root (/) file system if *ckbupscd* is run between the times of 4:00pm and 6:00am during the first week of August or January.

FILES

/etc/bupsched specification file containing times and file system to back up

SEE ALSO

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

BUGS

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

NAME

clri – clear i-node

SYNOPSIS

/etc/clri special i-number ...

DESCRIPTION

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

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

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

SEE ALSO

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

WARNINGS

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

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

This page is intentionally left blank

NAME

clrscr – clear screen

SYNOPSIS

/usr/ucb/clrscr

DESCRIPTION

clrscr clears the screen.

This page is intentionally left blank

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.) During 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.

This page is intentionally left blank

NAME

col – filter reverse line-feeds

SYNOPSIS

col [**−b**] [**−f**] [**−x**] [**−p**]

DESCRIPTION

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

If the **−b** option is given, *col* assumes that the output device in use is not capable of backspacing. In this case, if two or more characters are to appear in the same place, only the last one read will be output.

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

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

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

On input, the only control characters accepted are space, backspace, tab, return, new-line, **SI**, **SO**, **VT** (\013), and **ESC** followed by **7**, **8**, or **9**.

The **VT** character is an alternate form of full reverse line-feed, included for compatibility with some earlier programs of this type. All other non-printing characters are ignored.

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

SEE ALSO

nroff(1), tbl(1) in the *DOCUMENTER's WORKBENCH Software Release 2.0 Technical Discussion and Reference Manual* .

NOTES

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

BUGS

Cannot back up more than 128 lines.
Allows at most 800 characters, including backspaces, on a line.

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

NAME

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).

This page is intentionally left blank

171

NAME

config – print system configuration

SYNOPSIS

config [options]

DESCRIPTION

config writes to the standard output device a description of the configuration of the running system as set by the *chhw*(1M) program and defined by the hardware.

The options are:

a	List all parameters. Default.
c	List the information about the enabled channels on the ciocs.
d	List the information about the enabled channels on the diocs.
g	List the global parameters.
h	Make output as input to *chhw*(1M)
m	List the information about the mcus.
s	List the information about the enabled channels on the siocs.
v	Print the version of the *config* program.

EXAMPLES

To print the configuration of the disks on the Supermax:

config d

```
Dioc #13:
Channel 3:    First 560 KB 5¼" floppy.

Dioc2 #14:
Channel 1:    First  1 MB 8" floppy.
Channel 7:    Streamer tape length: 19,00 MB
Channel 8:    First hard disk on first controller,
              length: 63,25 MB
```

SEE ALSO

chhw(1M).

NAME

cp, ln, mv – copy, link or move files

SYNOPSIS

cp file1 [file2 ...] target
ln [**–fs**] file1 [file2 ...] target
mv [**–f**] file1 [file2 ...] target

DESCRIPTION

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

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

The **–s** option causes *ln* to create symbolic links. A symbolic link contains the name of the file to which it is linked. Symbolic links may span file systems and may refer to directories.

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

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

SEE ALSO

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

WARNINGS

ln will not 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

cpio − copy file archives in and out

SYNOPSIS

cpio −o [**acBvVHL**] [**−C** *bufsize*] [**−O** *file*]

cpio −i [**BcdmrtuvVfsSb6kHL**] [**−C** *bufsize*] [**−I** *file*]
[*pattern* ...]

cpio −p [**adlmuvVHL**] 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 filename-generating notation of *sh*(1). In *patterns*, meta-characters **?**, ∗ , and **[. . .]** match the slash (/) character, and backslash (\) is an escape character. A **!** meta-character means *not*. (For example, the **!abc*** pattern would exclude all files that begin with **abc**.) Multiple *patterns* may be specified and if no *patterns* are specified, the default for *patterns* is ∗ (i.e., select all files). Each *pattern* must be enclosed in double quotes otherwise the name of a file in the current directory is used. Extracted files are conditionally created and copied into the current directory tree based upon the options described below. The permissions of the files will be those of the previous **cpio −o**. The owner and group of the files will be that of the current user unless the user is super-user, which causes *cpio* to retain the owner and group of the files of the previous **cpio −o**.

NOTE: If **cpio −i** tries to create a file that already exists and the existing file is the same age or newer, *cpio* will output a warning message and not replace the file. (The **−u** option can be used to unconditionally overwrite the existing file.)

cpio −p (pass) reads the standard input to obtain a list of path names of files that are conditionally created and copied into the destination *directory* tree based upon the options described below.

The meanings of the available options are

−a	Reset *access* times of input files after they have been copied. Access times are not reset for linked files when **cpio −pla** is specified.
−b	Reverse the order of the *bytes* within each word. Use only with the **−i** option.
−B	Input/output is to be blocked 5,120 bytes to the record. The default buffer size is 512 bytes when this and the **C** options are not used. **−B** does not apply to the *pass* option.
−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.
−d	*Directories* are to be created as needed.
−f	Copy in all *files* except those in *patterns*. (See the paragraph on **cpio −i** for a description of *patterns*.)
−I *file*	Read the contents of *file* as input. Use only with the **−i** option.
−k	Attempt to skip corrupted file headers and I/O errors that may be encountered. If you want to copy files from a medium that is corrupted or out of sequence, this option lets you read only those files with good headers. (For *cpio* archives that contain other *cpio* archives, if an error is encountered *cpio* may terminate prematurely. *cpio* will find the next good header, which may be one for a smaller archive, and terminate when the smaller

archive's trailer is encountered.) Used only with the **−i** option.

−l Whenever possible, *link* files rather than copying them. Usable only with the **−p** option.

−m Retain previous file *modification* time. This option is ineffective on directories that are being copied.

−O *file* Direct the output of cpio to *file*. Use only with the **−o** option.

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

−s *swap* bytes within each half word. Use only with the **−i** option.

−S *Swap* halfwords within each word. Use only with the **−i** option.

−t Print a *table of contents* of the input. No files are created.

−u Copy *unconditionally* (normally, an older file will not replace a newer file with the same name).

−v *verbose*: causes a list of 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. UNIX System *Sixth* Edition format) file. Use only with the **−i** option.

−H Do not follow symbolic links (default). Symbolic link records are saved in the archive to be extracted on the other side. This is not portable to all system types.

−L Follow symbolic links, placing in archive records for the files they point to.

NOTE: *cpio* assumes four-byte words.

cpio will only read or write until it reaches end of medium.

cpio should not be used when accessing character special files. Instead use *bcpio* which gives data buffering optimal for the physical device in question, and handles media shift.

EXAMPLES

The following examples show three uses of *cpio*.

When standard input is directed through a pipe to **cpio −o**, it groups the files so they can be directed (>) to a single file (**../newfile**). The **c** option insures that the file will be portable to other machines. Instead of *ls*(1), you could use *find*(1), *echo*(1), *cat*(1), etc. to pipe a list of names to *cpio*. You could direct the output to a device instead of a file.

ls | cpio −oc > ../*newfile*

cpio −i uses the output file of **cpio −o** (directed through a pipe with **cat** in the example), extracts those files that match the patterns (**memo/a1**, **memo/b** ∗), creates directories below the current directory as needed (**−d** option), and places the files in the appropriate directories. The **c** option is used when the file is created with a portable header. If no patterns were given, all files from *newfile* would be placed in the directory.

cat newfile | cpio −icd *"memo/a1" "memo/b ∗ "*

cpio −p takes the 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*(1) to generate path names for *cpio*. This eliminates problems *cpio* could have trying to create files under read-only directories.)

find . −depth −print | cpio −pdlmv *newdir*

SEE ALSO

ar(1), bcpio(1), cat(1), echo(1), find(1), ls(1), tar(1), cpio(4).

NOTES

1) Path names are restricted to 256 characters.
2) Only the super-user can copy special files.
3) Blocks are reported in 512-byte quantities.
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) Will only read or write until end of media.
6) Never use cpio to acces streamer tapes.
7) SVID option "M" is not supported (see bcpio(1)).

This page is intentionally left blank

NAME

crash – provoke system crash

SYNOPSIS

crash

DESCRIPTION

crash causes the operating system to crash with the number 75 in the MCU display. *crash* will twice ask the user for confirmation before crashing.

The program can only be run by the super-user.

SEE ALSO

smsys(2).

This page is intentionally left blank

NAME

cron − clock daemon

SYNOPSIS

/etc/cron

DESCRIPTION

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

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

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

FILES

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

SEE ALSO

at(1), crontab(1), sh(1), ctime(3C).

DIAGNOSTICS

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

BUGS

Due to an error in function *ctime*(3C), the change between summertime and wintertime takes place at 1:am localtime, (and not at 2:am as expected).

184

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.

NOTE: A user will have no more than one crontab file. A second call will overwrite any previous crontab file.

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

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

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

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

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

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

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

FILES

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

SEE ALSO

cron(1M), sh(1).

WARNINGS

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

NAME

csh – a shell (command interpreter) with C-like syntax

SYNOPSIS

csh [**–cefinstvVxX**] [arg ...]

DESCRIPTION

csh is a command language interpreter incorporating a history mechanism (see **History Substitutions)** and a C-like syntax.

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

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

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

When a login shell terminates, it executes commands from the file '.logout' in the user's home directory.

LEXICAL STRUCTURE

The shell splits input lines into words at blanks and tabs with the following exceptions. The characters '&' '|' ';' '<' '>' '(' ')' form separate words. If doubled in '&&', '||', '<<' or '>>' these pairs form single words. These parser metacharacters may be made part of other words, or their special meaning may be prevented, by preceding them with a backslash, '\'. A newline preceded by a '\' is equivalent to a blank. It is usually necessary to use the backslash to 'escape' the parser metacharacters when you want to use them literally rather than as metacharacters.

Strings enclosed in matched pairs of quotation marks, either single or double quotation marks, '´', '`' or '"', form parts of a word. Metacharacters in these strings, including blanks and tabs, do not form separate words. Such quotations have semantics to be described subsequently.

Within pairs of single or double quotation marks a newline (carriage return) preceded by a '\' gives a true newline character. This is used to set up a file of strings separated by newlines, as for *fgrep(1).*

When the shell's input is not a terminal, the character '#' introduces a comment which continues to the end of the input line. It is prevented from having this special meaning when preceded by '\' or if bracketed by a pair of single or double quotation marks.

COMMANDS

A simple command is a sequence of words, the first of which specifies the command to be executed.

A simple command or a sequence of simple commands separated by '|' characters forms a pipeline. The output of each command in a pipeline is connected to the input of the next.

Sequences of pipelines may be separated by ';', and are then executed sequentially. A sequence of pipelines may be executed without immediately waiting for it to terminate by following it with an '&', which means 'run it in background'.

Parentheses '(' and ')' around a pipeline or sequence of pipelines cause the whole series to be treated as a simple command, which may in turn be a component of a pipeline, etc. It is also possible to separate pipelines with '||' or '&&' indicating, as in the C language, that the second is to be executed only if the first fails or succeeds respectively. (See *Expressions.)*

PROCESS I.D. NUMBERS

When a process is run in background with '&', the shell prints a line which looks like:

```
1234
```

indicating that the process which was started asynchronously was number 1234.

STATUS REPORTING

This shell learns immediately whenever a process changes state. It normally informs you whenever a job becomes blocked so that no further progress is possible, but only just before it prints a prompt. This is done so that it does not otherwise disturb your work.

To check on the status of a process, use the *ps* (process status) command.

SUBSTITUTIONS

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

History substitutions

History substitutions place words from previous command input as portions of new commands, making it easy to repeat commands, repeat arguments of a previous command in the current command, or fix spelling mistakes in the previous command with little typing and a high degree of confidence.

History substitutions begin with the character '!' and may begin **anywhere** in the input stream (with the proviso that they **do not** nest.)

This '!' may be preceded by an '\' to turn off its special meaning; for convenience, a '!' is also passed unchanged when it is followed by a blank, tab, newline, '=' or '('.

Therefore, do *not* put a space after the '!' and the command reference when you are invoking the shell's history mechanism. (History substitutions also occur when an input line begins with '^'. This special abbreviation will be described later.)

An input line which invokes history substitution is echoed on the terminal before it is executed, as it would look if typed out in full.

The shell's history list, which may be seen by typing the 'history' command, contains all commands input from the terminal which consist of one or more words. History substitutions reintroduce sequences of words from these saved commands into the input stream. The *history* variable controls the size of the input stream. The previous command is always retained, regardless of its value. Commands are numbered sequentially from 1.

Consider the following output from the *history* command:

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

The commands are shown with their event numbers. It is not usually necessary to use event numbers, but the current event number can be made part of the *prompt* by placing an '!' in the prompt string. This is done by SETting Prompt = ! and the prompt character of your choice.

For example, if the current event is number 13, we can call up the command recorded as event 11 in several ways: as !-2 [i.e., 13-2];

by the first letter of one of its command words, such as !c referring to the 'c' in *cat;*

or !wri for event 9, or by a string contained in a word in the command as in '!?mic?' also referring to event 9.

These forms, without further modification, simply reintroduce the words of the specified events, each separated by a single blank. As a special case '!!' refers to the previous command; thus '!!' alone is essentially a *redo.*

Words are selected from a command event and acted upon according to the following formula:

```
event:position:action
```

The 'event' is the command you wish to retrieve. As mentioned above, it may be summoned up by event number and in several other ways. All that the 'event' notation does is to tell the shell *which* command you have in mind.

'Position' picks out the words from the command event on which you want the 'action' to take place. The 'position' notation can do anything from altering the command completely to making some very minor substitution, depending on which words from the command event you specify with the 'position' notation.

To select words from a command event, follow the event specification with a ':' and a designator (by position) for the desired words.

The words of a command event are picked out by their position in the input line. Positions are numbered from 0, the first word (usually command) being position 0, the second word having position 1, and so forth. If you designate a word from the command event by stating its position, that means you want to include it in your revised command. All the words that you want to include in a revised command must be designated by position notation in order to be included.

The basic position designators are:

0	first (command) word
n	*n*'th argument
^	first argument, i.e. '1'
$	last argument
%	matches the word of an ?s? search which immediately precedes it; used to strip one word out of a command event for use in another command. Example: !?four?:%:p prints 'four'.

x −*y*	range of words (e.g. 1-3 means 'from position 1 to position 3').
−*y*	abbreviates '0−*y*'
∗	stands for '^−$', or indicates position 1 if only one word in event.
x ∗	abbreviates '*x* −$' where x is a position number.
x −	like '*x* ∗ ' but omitting last word '$'

The ':' separating the event specification from the word designator can be omitted if the argument selector begins with a '^', '$', ' ∗ ' '−' or '%'.

Modifiers, each preceded by a ':', may be used to act on the designated words in the specified command event. The following modifiers are defined:

h	Remove a trailing pathname component, leaving the head.
r	Remove a trailing '.xxx' component, leaving the root name.
e	Remove all but the extension '.xxx' part.
s*old*/*new*	Substitute *new* for *old*
t	Remove all leading pathname components, leaving the tail.
&	Repeat the previous substitution.
g	Apply the change globally, prefixing the above, e.g. 'g&'.
p	Print the new command but do not execute it.
q	Quote the substituted words, preventing further substitutions.
x	Like q, but break into words at blanks, tabs and newlines.

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

The left hand side of substitutions are not regular expressions in the sense of the editors, but rather strings. Any character may be used as the delimiter in place of '/'; a '\' quotes the delimiter into the *l* and *r* strings. The character '&' in the right hand side is replaced by the text from the left. A '\' quotes '&' also. A null *l* uses the previous string either from a *l* or from a contextual scan string *s* in '!?*s*?'. The trailing delimiter in the substitution may be omitted if (but only if) a newline follows immediately as may the trailing '?' in a contextual scan.

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

You can quickly make substitutions to the previous command line by using the '^' character as the first non-blank character of an input line. This is equivalent to '!:s^' providing a convenient shorthand for substitutions on the text of the previous line. Thus '^lb^lib' fixes the spelling of 'lib' in the previous command. Finally, a history substitution may be surrounded with '{' and '}' if necessary to insulate it from the characters which follow. Thus, after 'ls −ld ~paul' we might do '!{l}a' to do 'ls −ld ~paula', while '!la' would look for a command starting 'la'.

Quotations with ´ and "

The quotation of strings by '´' and '"' can be used to prevent all or some of the remaining substitutions which would otherwise take place if these characters were interpreted as 'metacharacters' or 'wild card matching characters'. Strings enclosed in single quotes, '´' are prevented any further interpretation or expansion. Strings enclosed in '"' may still be variable and command expanded as described below.

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

Alias substitution

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

Thus if the alias for 'ls' is 'ls −l' the command 'ls /usr' would map to 'ls −l /usr', the argument list here being undisturbed. Similarly if the alias for 'lookup' was 'grep !^ /etc/passwd' then 'lookup bill' would map to 'grep bill /etc/passwd'.

If an alias is found, the word transformation of the input text is performed and the aliasing process begins again on the reformed input line. Looping is prevented if the first word of the new text is the same as the old by flagging it to prevent further aliasing. Other loops are detected and cause an error.

Note that the mechanism allows aliases to introduce parser metasyntax. Thus we can 'alias print ´pr \!* | lpr´' to make a command which *pr's* its arguments to the line printer.

Variable substitution

The shell maintains a set of variables, each of which has as value a list of zero or more words. Some of these variables are set by the shell or referred to by it. For instance, the *argv* variable is an image of the shell's argument list, and words of this variable's value are referred to in special ways.

The values of variables may be displayed and changed by using the *set* and *unset* commands. Of the variables referred to by the shell a number are toggles; the shell does not care what their value is, only whether they are set or not. For instance, the *verbose* variable is a toggle which causes command input to be echoed. The setting of this variable results from the **−v** command line option. Other operations treat variables numerically. The '@' command permits numeric calculations to be performed and the result assigned to a variable. Variable values are, however, always represented as (zero or more) strings. For the purposes of numeric operations, the null string is considered to be zero, and the second and subsequent words of multiword values are ignored.

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

195

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

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

Metasequences for variable substitution

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

$name
${name}

Are replaced by the words of the value of variable *name*, each separated by a blank. Braces insulate *name* from following characters which would otherwise be part of it. Shell variables have names consisting of up to 20 letters and digits starting with a letter. The underscore character is considered a letter.
If *name* is not a shell variable, but is set in the environment, then that value is returned (but **:** modifiers and the other forms given below are not available in this case).

$name[selector]
${name[selector]}

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

$#name
${#name}

Gives the number of words in the variable. This is useful for later use in a '[selector]'.

$0

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

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

$ *
Equivalent to '$argv[*]'.

The modifiers ':h', ':t', ':r', ':q' and ':x' may be applied to the substitutions above as may ':gh', ':gt' and ':gr'. If braces '{' '}' appear in the command form then the modifiers must appear within the braces. **The current implementation allows only one ':' modifier on each '$' expansion.**

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

$?name
${?name}
Substitutes the string '1' if name is set, '0' if it is not.

$?0
Substitutes '1' if the current input filename is know, '0' if it is not.

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

Command and filename substitution

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

197

Command substitution

Command substitution is indicated by a command enclosed in ’’’. The output from such a command is normally broken into separate words at blanks, tabs and newlines, with null words being discarded, this text then replacing the original string. Within ’"’s, only newlines force new words; blanks and tabs are preserved.

In any case, the single final newline does not force a new word. Note that it is thus possible for a command substitution to yield only part of a word, even if the command outputs a complete line.

Filename substitution

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

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

The character ’˜’ at the beginning of a filename is used to refer to home directories. Standing alone, i.e. ’˜’ it expands to the invokers home directory as reflected in the value of the variable *home*. When followed by a name consisting of letters, digits and ’ – ’ characters the shell searches for a user with that name and substitutes their home directory; thus ’~ken’ might expand to ’/usr/ken’ and ’~ken/chmach’ to ’/usr/ken/chmach’. If the character ’~’ is followed by a character other than a letter or ’/’ or appears not at the beginning of a word, it is left undisturbed.

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

Input/output

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

< name

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

< < word

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

> name
>! name
>& name
>&! name

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

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

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

>> name
>>& name
>>! name
>>&! name

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

A command receives the environment in which the shell was invoked as modified by the input-output parameters and the presence of the command in a pipeline. Thus, unlike some previous shells, commands run from a file of shell commands have no access to the text of the commands by default; rather they receive the original standard input of the shell. The '<<' mechanism should be used to present inline data. This permits shell command scripts to function as components of pipelines and allows the shell to block read its input.

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

200

Expressions

A number of the builtin commands (to be described subsequently) take expressions, in which the operators are similar to those of C, with the same precedence. These expressions appear in the @, *exit, if,* and *while* commands. The following operators are available:

```
          | | &&  |  ^  &  = =  !=  =~  !~  <=  >=  <  >
<<  >>  +  −  *  /  %  !  ~  ( )
```

Here the precedence increases to the right, '= =' '!=' '=~' and '!~', '<=' '>=' '<' and '>', '<<' and '>>', '+' and '−', '*' '/' and '%' being, in groups, at the same level. The '= =' '!=' '=~' and '!~' operators compare their arguments as strings; all others operate on numbers. The operators '=~' and '!~' are like '!=' and '= =' except that the right hand side is a *pattern* (containing, e.g. '*'s, '?'s and instances of '[...]') against which the left hand operand is matched. This reduces the need for use of the *switch* statement in shell scripts when all that is really needed is pattern matching.

Strings which begin with '0' are considered octal numbers. Null or missing arguments are considered '0'. The result of all expressions are strings, which represent decimal numbers. It is important to note that no two components of an expression can appear in the same word; except when adjacent to components of expressions which are syntactically significant to the parser ('&' '|' '<' '>' '(' ')') they should be surrounded by spaces.

Also available in expressions as primitive operands are command executions enclosed in '{' and '}' and file enquiries of the form '−*l* name' where *l* is one of:

r	read access
w	write access
x	execute access
e	existence
o	ownership
z	zero size

f	plain file
d	directory

The specified name is command and filename expanded and then tested to see if it has the specified relationship to the real user. If the file does not exist or is inaccessible, then all enquiries return false, i.e. '0'. Command executions succeed, returning true, i.e. '1', if the command exits with status 0, otherwise they fail, returning false, i.e. '0'. If more detailed status information is required then the command should be executed outside of an expression and the variable *status* examined.

CONTROL FLOW

The shell contains a number of commands which can be used to regulate the flow of control in command files (shell scripts) and (in limited but useful ways) from terminal input. These commands all operate by forcing the shell to reread or skip in its input and, due to the implementation, restrict the placement of some of the commands.

The *foreach, switch,* and *while* statements, as well as the *if−then−else* form of the *if* statement require that the major keywords appear in a single simple command on an input line as shown below.

If the shell's input is not seekable, the shell buffers up input whenever a loop is being read and performs seeks in this internal buffer to accomplish the rereading implied by the loop. (To the extent that this allows, backward goto's will succeed on non-seekable inputs.)

BUILTIN COMMANDS

Builtin commands are executed within the shell. If a builtin command occurs as any component of a pipeline except the last then it is executed in a subshell.

alias
alias name
alias name wordlist

The first form prints all aliases. The second form prints the alias for name. The final form assigns the specified *wordlist* as the alias of *name; wordlist* is command and filename substituted. *Name* is not allowed to be *alias* or *unalias.*

break
Causes execution to resume after the *end* of the nearest enclosing *foreach* or *while*. The remaining commands on the current line are executed. Multi-level breaks are thus possible by writing them all on one line.

breaksw
Causes a break from a *switch,* resuming after the *endsw*.

case label:
A label in a *switch* statement as discussed below.

cd
cd name
chdir
chdir name
Change the shells working directory to directory *name*. If no argument is given then change to the home directory of the user.
If *name* is not found as a subdirectory of the current directory (and does not begin with '/', './' or '../'), then each component of the variable *cdpath* is checked to see if it has a subdirectory *name*. Finally, if all else fails but *name* is a shell variable whose value begins with '/', then this is tried to see if it is a directory.

continue
Continue execution of the nearest enclosing *while* or *foreach*. The rest of the commands on the current line are executed.

default:
Labels the default case in a *switch* statement. The default should come after all *case* labels.

echo wordlist

echo **−n** wordlist
: The specified words are written to the shells standard output, separated by spaces, and terminated with a newline unless the **−n** option is specified.

else
end
endif
endsw
: See the description of the *foreach, if, switch,* and *while* statements below.

exec command
: The specified command is executed in place of the current shell.

exit
exit(expr)
: The shell exits either with the value of the *status* variable (first form) or with the value of the specified *expr* (second form).

foreach name (wordlist)
...
end
: The variable *name* is successively set to each member of *wordlist* and the sequence of commands between this command and the matching *end* are executed. (Both *foreach* and *end* must appear alone on separate lines.)

The builtin command *continue* may be used to continue the loop prematurely and the builtin command *break* to terminate it prematurely. When this command is read from the terminal, the loop is read up once prompting with '?' before any statements in the loop are executed. If you make a mistake typing in a loop at the terminal you can rub it out.

glob wordlist
: Like *echo* but no '\' escapes are recognized and words are delimited by null characters in the output. Useful

204

for programs which wish to use the shell to filename expand a list of words.

goto word
: The specified *word* is filename and command expanded to yield a string of the form 'label'. The shell rewinds its input as much as possible and searches for a line of the form 'label:' possibly preceded by blanks or tabs. Execution continues after the specified line.

history
: Displays the history event list.

if (expr) command
: If the specified expression evaluates true, then the single *command* with arguments is executed. Variable substitution on *command* happens early, at the same time it does for the rest of the *if* command. *Command* must be a simple command, not a pipeline, a command list, or a parenthesized command list. Input/output redirection occurs even if *expr* is false, when command is **not** executed (this is a bug).

if (expr) **then**

...

else if (expr2) **then**

...

else

...

endif
: If the specified *expr* is true then the commands to the first *else* are executed; else if *expr2* is true then the commands to the second else are executed, etc. Any number of *else-if* pairs are possible; only one *endif* is needed. The *else* part is likewise optional. (The words *else* and *endif* must appear at the beginning of input lines; the *if* must appear alone on its input line or after an *else.)*

kill pid
kill −sig pid ...
: Sends either the TERM (terminate) signal or the

205

specified signal to the specified processes. Signals are either given by number or by names (as given in */usr/include/signal.h,* stripped of the prefix 'SIG'). There is no default, saying just 'kill' does not send a signal to the current process. If the signal being sent is TERM (terminate) or HUP (hangup), then the job or process will be sent a CONT (continue) signal as well.

login

Terminate a login shell, replacing it with an instance of **/bin/login.** This is one way to log off, included for compatibility with *sh*(1).

logout

Terminate a login shell. Especially useful if *ignoreeof* is set.

nice
nice +number
nice command
nice +number command

The first form sets the *nice* for this shell to 4. The second form sets the *nice* to the given number. The final two forms run command at priority 4 and *number* respectively. The super-user may specify negative niceness by using 'nice −number ...'. Command is always executed in a sub-shell, and the restrictions place on commands in simple *if* statements apply.

nohup
nohup command

The first form can be used in shell scripts to cause hangups to be ignored for the remainder of the script. The second form causes the specified command to be run with hangups ignored. All processes detached with '&' are effectively *nohup'ed.*

onintr
onintr −
onintr label

Control the action of the shell on interrupts. The first form restores the default action of the shell on interrupts which is to terminate shell scripts or to return to the terminal command input level. The second form 'onintr −' causes all interrupts to be ignored. The final form causes the shell to execute a 'goto label' when an interrupt is received or a child process terminates because it was interrupted.

In any case, if the shell is running detached and interrupts are being ignored, all forms of *onintr* have no meaning and interrupts continue to be ignored by the shell and all invoked commands.

rehash

Causes the internal hash table of the contents of the directories in the *path* variable to be recomputed. This is needed if new commands are added to directories in the *path* while you are logged in. This should only be necessary if you add commands to one of your own directories, or if a systems programmer changes the contents of one of the system directories.

repeat count command

The specified *command* which is subject to the same restrictions as the *command* in the one line *if* statement above, is executed *count* times. I/O redirections occur exactly once, even if *count* is 0.

set
set name
set name=word
set name[index]=word
set name=(wordlist)

The first form of the command shows the value of all shell variables. Variables which have other than a single word as value print as a parenthesized word list. The second form sets *name* to the null string. The third form sets *name* to the single *word*. The fourth form sets the *index'th* component of name to word; this component must already exist. The final form sets *name* to the list of words in *wordlist.* In all cases the value is command and filename expanded.

These arguments may be repeated to set multiple values in a single set command. Note however, that variable expansion happens for all arguments before any setting occurs.

setenv name value
Sets the value of environment variable *name* to be *value,* a single string. The variable PATH is automatically imported to and exported from the *csh* variable *path;* there is no need to use *setenv* for these.

shift
shift variable
The members of *argv* are shifted to the left, discarding *argv[1].* It is an error for *argv* not to be set or to have less than one word as value. The second form performs the same function on the specified variable.

source name
The shell reads commands from *name. Source* commands may be nested; if they are nested too deeply the shell may run out of file descriptors. An error in a *source* at any level terminates all nested *source* commands. Input during *source* commands is **never** placed on the history list.

switch (string)
case str1:
...
breaksw
...
default:
...
breaksw
endsw
Each case label is successively matched against the specified *string* which is first command and filename expanded. The file metacharacters '*', '?' and '[...]'

may be used in the case labels, which are variable expanded. If none of the labels match before a 'default' label is found, then the execution begins after the default label. Each case label and the default label must appear at the beginning of a line. The command *breaksw* causes execution to continue after the *endsw*. Otherwise control may fall through case labels and default labels as in C. If no label matches and there is no default, execution continues after the *endsw*.

time
time command
With no argument, a summary of time used by this shell and its children is printed. If arguments are given the specified simple command is timed and a time summary as described under the *time* variable is printed. If necessary, an extra shell is created to print the time statistic when the command completes.

209

umask
umask value
The file creation mask is displayed (first form) or set to the specified value (second form). The mask is given in octal. Common values for the mask are 002 giving all access to the group and read and execute access to others or 022 giving all access except no write access for users in the group or others.

unalias pattern
All aliases whose names match the specified pattern are discarded. Thus all aliases are removed by 'unalias * '. It is not an error for nothing to be *unaliased*.

unhash
Use of the internal hash table to speed location of executed programs is disabled.

unset pattern
All variables whose names match the specified pattern are removed. Thus all variables are removed by 'unset * '; this has noticeably distasteful side-effects. It is not

an error for nothing to be *unset.*

wait

All background jobs are waited for. If the shell is interactive, then an interrupt can disrupt the wait, at which time the shell prints names and job numbers of all jobs known to be outstanding.

while (expr)

...

end

While the specified expression evaluates non-zero, the commands between the *while* and the matching end are evaluated. *Break* and *continue* may be used to terminate or continue the loop prematurely. (The *while* and *end* must appear alone on their input lines.) Prompting occurs here the first time through the loop as for the *foreach* statement if the input is a terminal.

@
@ name = expr
@ name[index] = expr

The first form prints the values of all the shell variables. The second form sets the specified *name* to the value of *expr*. If the expression contains '<', '>', '&' or '|' then at least this part of the expression must be placed within '(' ')'. The third form assigns the value of *expr* to the *index'th* argument of *name*. Both *name* and its *index'th* component must already exist.

The operators '*=', '+=', etc are available as in C. The space separating the name from the assignment operator is optional. Spaces are, however, mandatory in separating components of *expr* which would otherwise be single words.

Special postfix '++' and '−−' operators increment and decrement *name* respectively, i.e. '@ i++'.

211

PRE-DEFINED AND ENVIRONMENT VARIABLES

The following variables have special meaning to the shell. Of these, *argv, home, path, prompt, shell* and *status* are always set by the shell. Except for *status* this setting occurs only at initialization; these variables will not then be modified unless this is done explicitly by the user.

This shell copies the environment variable USER into the variable *user,* TERM into *term,* and HOME into *home,* and copies these back into the environment whenever the normal shell variables are reset. The environment variable PATH is likewise handled; it is not necessary to worry about its setting other than in the file *.cshrc* as inferior *csh* processes will import the definition of *path* from the environment, and re-export it if you then change it.

argv	Set to the arguments to the shell, it is from this variable that positional parameters are substituted, i.e. '$1' is replaced by '$argv[1]', etc.
cdpath	Gives a list of alternate directories searched to find subdirectories in *chdir* commands.
echo	Set when the **−x** command line option is given. Causes each command and its arguments to be echoed just before it is executed. For non-builtin commands all expansions occur before echoing. Builtin commands are echoed before command and filename substitution, since these substitutions are then done selectively.
history	Can be given a numeric value to control the size of the history list. Any command which has been referenced in this many events will not be discarded. Too large values of *history* may run the shell out of memory. The last executed command is always saved on the history list.

home The home directory of the invoker, initialized from the environment. The filename expansion of '~' refers to this variable.

ignoreeof If set the shell ignores end-of-file from input devices which are terminals. This prevents shells from accidentally being killed by control-D's.

mail The files where the shell checks for mail. This is done after each command completion which will result in a prompt, if a specified interval has elapsed. The shell says 'You have new mail.' if the file exists with an access time not greater than its modify time.

If the first word of the value of *mail* is numeric it specifies a different mail checking interval, in seconds, than the default, which is 10 minutes.

If multiple mail files are specified, then the shell says 'New mail in *name*' when there is mail in the file *name*.

noclobber As described in the section on *Input/output,* restrictions are placed on output redirection to insure that files are not accidentally destroyed, and that '> >' redirections refer to existing files.

noglob If set, filename expansion is inhibited. This is most useful in shell scripts which are not dealing with filenames, or after a list of filenames has been obtained and further expansions are not desirable.

nonomatch

If set, it is not an error for a filename expansion to not match any existing files; rather the primitive pattern is returned. It is still an error for the primitive pattern to be malformed, i.e. 'echo [' still gives an error.

path Each word of the path variable specifies a directory in which commands are to be sought for execution. A null word specifies the current directory. If there is no *path* variable then only full path names will execute. The usual search path is '.', '/bin' and '/usr/bin', but this may vary from system to system. For the super-user the default search path is '/etc', '/bin' and '/usr/bin'. A shell which is given neither the **−c** nor the **−t** option will normally hash the contents of the directories in the *path* variable after reading *.cshrc,* and each time the *path* variable is reset. If new commands are added to these directories while the shell is active, it may be necessary to give the *rehash* or the commands may not be found.

prompt The string which is printed before each command is read from an interactive terminal input. If a '!' appears in the string it will be replaced by the current event number unless a preceding '\' is given. Default is '% ', or '# ' for the super-user.

shell The file in which the shell resides. This is used in forking shells to interpret files which have execute bits set, but which are not executable by the system. (See the description of *Non-builtin Command Execution* below.) Initialized to the (system-dependent) home of the shell.

status The status returned by the last command. If it terminated abnormally, then 0200 is added to the status. Builtin commands which fail return exit status '1', all other builtin commands set status '0'.

time Controls automatic timing of commands. If set, then any command which takes more than this many cpu seconds will cause a line giving user, system, and real times and a utilization percentage which is the ratio of user plus system times to real time to be printed when it terminates.

verbose Set by the **−v** command line option, causes the words of each command to be printed after history substitution.

NON-BUILTIN COMMAND EXECUTION

When a command to be executed is found not to be a builtin command the shell attempts to execute the command via *exec*(2). Each word in the variable *path* names a directory from which the shell will attempt to execute the command. If it is given neither a **−c** nor a **−t** option, the shell will hash the names in these directories into an internal table so that it will only try an *exec* in a directory if there is a possibility that the command resides there. This greatly speeds command location when a large number of directories are present in the search path. If this mechanism has been turned off (via *unhash),* or if the shell was given a **−c** or **−t** argument, and in any case for each directory component of *path* which does not begin with a '/', the shell concatenates with the given command name to form a path name of a file which it then attempts to execute.

Parenthesized commands are always executed in a subshell. Thus '(cd ; pwd) ; pwd' prints the *home* directory; leaving you where you were (printing this after the home directory), while 'cd ; pwd' leaves you in the *home* directory. Parenthesized commands are most often used to prevent *chdir* from affecting the current shell.

If the file has execute permissions but is not an executable binary to the system, then it is assumed to be a file containing shell commands an a new shell is spawned to read it.

If there is an *alias* for *shell* then the words of the alias will be prepended to the argument list to form the shell command. The first word of the *alias* should be the full path name of the shell (e.g. '$shell'). Note that this is a special, late occurring, case of *alias* substitution, and only allows words to be prepended to the argument list without modification.

ARGUMENT LIST PROCESSING

If argument 0 to the shell is '−' then this is a login shell. The flag arguments are interpreted as follows:

−c Commands are read from the (single) following argument which must be present. Any remaining arguments are placed in *argv*.

−e The shell exits if any invoked command terminates abnormally or yields a non-zero exit status.

−f The shell will start faster, because it will neither search for nor execute commands from the file '.cshrc' in the invokers home directory.

−i The shell is interactive and prompts for its top-level input, even if it appears to not be a terminal. Shells are interactive without this option if their inputs and outputs are terminals.

−n Commands are parsed, but not executed. This may aid in syntactic checking of shell scripts.

−s Command input is taken from the standard input.

−t A single line of input is read and executed. A '\' may be used to escape the newline at the end of this line and continue onto another line.

−v Causes the *verbose* variable to be set, with the effect that command input is echoed after history substitution.

−x Causes the *echo* variable to be set, so that commands are echoed immediately before execution.

−V Causes the *verbose* variable to be set even before '.cshrc' is executed.

−X Is to **−x** as **−V** is to **−v.**

After processing of flag arguments, if arguments remain but none of the **−c, −i, −s,** or **−t** options was given, the first argument is taken as the name of a file of commands to be executed. The shell opens this file, and saves its name for possible resubstitution by '$0'. Remaining arguments initialize the variable *argv*.

SIGNAL HANDLING

The shell normally ignores *quit* signals. Processes running in background (by '&') are immune to signals generated from the keyboard, including hangups. Other signals have the values which the shell inherited from its parent. The shells handling of interrupts and terminate signals in shell scripts can be controlled by *onintr*. Login shells catch the *terminate* signal; otherwise this signal is passed on to children from the state in the shell's parent. In no case are interrupts allowed when a login shell is reading the file '.logout'.

EXAMPLE

Typing in:

```
csh
```

creates a new shell which will accept shell commands with Berkeley extensions.

AUTHOR

William Joy.

FILES

˜/.cshrc	Read at beginning of execution by each shell.
˜/.login	Read by login shell, after '.cshrc' at login.
˜/.logout	Read by login shell, at logout.
/bin/sh	Standard shell, for shell scripts not starting with a '#'.

/tmp/sh *	Temporary file for '< <'.
/etc/passwd	Source of home directories for '˜name'.

LIMITATIONS

Words can be no longer than 1024 characters. The system limits argument lists to 5120 characters. The number of arguments to a command which involves filename expansion is limited to 1/6'th the number of characters allowed in an argument list. Command substitutions may substitute no more characters than are allowed in an argument list. To detect looping, the shell restricts the number of *alias* substitutions on a single line to 20.

SEE ALSO

dsh(1), sh(1), access(2), exec(2), fork(2), pipe(2), signal(2), umask(2), wait(2), environ(5), tty(7).

217

BUGS

It suffices to place the sequence of commands in ()'s to force it to a subshell, i.e. '(a ; b ; c)'.

Alias substitution is most often used to clumsily simulate shell procedures; shell procedures should be provided rather than aliases.

Commands within loops, prompted for by '?', are not placed in the *history* list. Control structure should be parsed rather than being recognized as built-in commands. This would allow control commands to be placed anywhere, to be combined with '|', and to be used with '&' and ';' metasyntax.

It should be possible to use the ':' modifiers on the output of command substitutions. All and more than one ':' modifier should be allowed on '$' substitutions.

This page is intentionally left blank

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 – then standard input is used.

The options to *csplit* are:

–s	*csplit* normally prints the character counts for each file created. If the **–s** option is present, *csplit* suppresses the printing of all character counts.
–k	*csplit* normally removes created files if an error occurs. If the **–k** option is present, *csplit* leaves previously created files intact.
–f *prefix*	If the **–f** option is used, the created files are named *prefix***00** ... *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), 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.

221

This page is intentionally left blank

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.

223

−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 new-line character.

ERROR: bad list for c / f option
Missing **−c** or **−f** option or incorrectly specified *list*. No error occurs if a line has fewer fields than the *list* calls for.

ERROR: no fields The *list* is empty.

ERROR: no delimeter
Missing *char* on **−d** option.

ERROR: cannot handle multiple adjacent backspaces
Adjacent backspaces cannot be processed correctly.

WARNING: cannot open <filename>
Either *filename* cannot be read or does not exist. If multiple filenames are present, prcessing continues.

SEE ALSO

grep(1), paste(1).

This page is intentionally left blank

NAME

date – print and set the date

SYNOPSIS

date [+ format]
date [mmddhhmm[[yy] | [ccyy]]]

DESCRIPTION

If no argument is given, or if the argument begins with +, the current date and time are printed. Otherwise, the current date is set (only by super-user). The first *mm* is the month number; *dd* is the day number in the month; *hh* is the hour number (24 hour system); the second *mm* is the minute number; *cc* is the century minus one and is optional; *yy* is the last 2 digits of the year number and is optional. For example:

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 super-user may change the date.

If the argument begins with +, the output of *date* is under the control of the user. All output fields are of fixed size (zero padded if necessary). Each Field Descriptor is preceded by % and will be replaced in the output by its corresponding value. A single % is encoded by %%. All other characters are copied to the output without change. The string is always terminated with a new-line character. If the argument contains embedded blanks it must be quoted (see the **EXAMPLE** section).

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

After successfully setting the date and time, *date* will display the new date according to the format defined in the environment variable **CFTIME** (see *environ*(5)).

Field Descriptors (must be preceded by a %):

a	abbreviated weekday name
A	full weekday name
b	abbreviated month name
B	full month name
d	day of month – 01 to 31
D	date as mm/dd/yy
e	day of month – 1 to 31 (single digits are preceded by a blank)
h	abbreviated month name (alias for %b)
H	hour – 00 to 23
I	hour – 01 to 12
j	day of year – 001 to 366
m	month of year – 01 to 12
M	minute – 00 to 59
n	insert a new-line character
p	string containing ante-meridiem or post-meridiem indicator (by default, AM or PM)
r	time as *hh:mm:ss pp* where *pp* is the ante-meridiem or post-meridiem indicator (by default, AM or PM)
R	time as hh:mm
S	second – 00 to 59
t	insert a tab character
T	time as *hh:mm:ss*
U	week number of year (Sunday as the first day of the week) – 01 to 52
w	day of week – Sunday = 0
W	week number of year (Monday as the first day of the week) – 01 to 52
x	Country-specific date format
X	Country-specific time format
y	year within century – 00 to 99

228

Y	year as *ccyy* (4 digits)
Z	timezone name

EXAMPLE

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

would have generated as output:

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

DIAGNOSTICS

No permission	if you are not the super-user and you try to change the date
bad conversion	if the date set is syntactically incorrect

FILES

/dev/kmem

WARNING

Should you need to change the date while the system is running multi-user, use *sysadm*(1) *datetime*.

NOTE

Administrators should note the following: if you attempt to set the current date to one of the dates that the standard and alternate time zones change (for example, the date that daylight time is starting or ending), and you attempt to set the time to a time in the interval between the end of standard time and the beginning of the alternate time (or the end of the alternate time and the beginning of standard time), the results are unpredictable.

SEE ALSO

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

This page is intentionally left blank

NAME

dc − desk calculator

SYNOPSIS

dc [file]

DESCRIPTION

dc is an arbitrary precision arithmetic package. Ordinarily it operates on decimal integers, but one may specify an input base, output base, and a number of fractional digits to be maintained. (See *bc*(1), a preprocessor for *dc* that provides infix notation and a C-like syntax that implements functions. *bc* also provides reasonable control structures for programs.) The overall structure of *dc* is a stacking (reverse Polish) calculator. If an argument is given, input is taken from that file until its end, then from the standard input. The following constructions are recognized:

number The value of the number is pushed on the stack. A number is an unbroken string of the digits 0−9. It may be preceded by an underscore (_) to input a negative number. Numbers may contain decimal points.

+ − / * % ^

The top two values on the stack are added (**+**), subtracted (**−**), multiplied (*****), divided (**/**), remaindered (**%**), or exponentiated (**^**). The two entries are popped off the stack; the result is pushed on the stack in their place. Any fractional part of an exponent is ignored.

s*x* The top of the stack is popped and stored into a register named *x*, where *x* may be any character. If the **s** is capitalized, *x* is treated as a stack and the value is pushed on it.

l*x* The value in register *x* is pushed on the stack. The register *x* is not altered. All registers start with zero value. If the **l** is capitalized, register *x* is treated as a stack and its top value is popped onto the main stack.

d	The top value on the stack is duplicated.
p	The top value on the stack is printed. The top value remains unchanged.
P	Interprets the top of the stack as an ASCII string, removes it, and prints it.
f	All values on the stack are printed.
q	Exits the program. If executing a string, the recursion level is popped by two.
Q	Exits the program. The top value on the stack is popped and the string execution level is popped by that value.
x	Treats the top element of the stack as a character string and executes it as a string of *dc* commands.
X	Replaces the number on the top of the stack with its scale factor.
[...]	Puts the bracketed ASCII string onto the top of the stack.
<*x* **>***x* **=***x*	The top two elements of the stack are popped and compared. Register *x* is evaluated if they obey the stated relation.
v	Replaces the top element on the stack by its square root. Any existing fractional part of the argument is taken into account, but otherwise the scale factor is ignored.
!	Interprets the rest of the line as a UNIX system command.
c	All values on the stack are popped.
i	The top value on the stack is popped and used as the number radix for further input. **I** Pushes the input base on the top of the stack.

232

o	The top value on the stack is popped and used as the number radix for further output.
O	Pushes the output base on the top of the stack.
k	The top of the stack is popped, and that value is used as a non-negative scale factor: the appropriate number of places are printed on output, and maintained during multiplication, division, and exponentiation. The interaction of scale factor, input base, and output base will be reasonable if all are changed together.
z	The stack level is pushed onto the stack.
Z	Replaces the number on the top of the stack with its length.
?	A line of input is taken from the input source (usually the terminal) and executed.
; :	are used by *bc*(1) for array operations.

EXAMPLE

This example prints the first ten values of n!:

```
[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

dcopy − copy file systems for optimal access time

SYNOPSIS

/etc/dcopy [**−s**X] [**−a**n] [**−d**] [**−v**] [**−f**fsize[:isize]] inputfs outputfs

DESCRIPTION

dcopy copies file system *inputfs* to *outputfs*. *Inputfs* is the device file for the existing file system; *outputfs* is the device file to hold the reorganized result. For the most effective optimization *inputfs* should be the raw device and *outputfs* should be the block device. Both *inputfs* and *outputfs* should be unmounted file systems (in the case of the root file system, the copy must be to a new pack).

With no options, *dcopy* copies files from *inputfs* compressing directories by removing vacant entries, and spacing consecutive blocks in a file by the optimal rotational gap. The possible options are

−s*X* supply device information for creating an optimal organization of blocks in a file. The forms of *X* are the same as the **−s** option of *fsck*(1M).

−a*n* place the files not accessed in *n* days after the free blocks of the destination file system (default for *n* is 7). If no *n* is specified then no movement occurs.

−d leave order of directory entries as is (default is to move sub-directories to the beginning of directories).

−v currently reports how many files were processed, and how big the source and destination freelists are.

−f*fsize*[:*isize*]
specify the *outputfs* file system and inode list sizes (in blocks). If the option (or :*isize*) is not given, the values from the *inputfs* are used.

dcopy catches interrupts and quits, and reports on its progress. To terminate *dcopy* send a quit signal, followed by an interrupt or quit.

SEE ALSO

fsck(1M), mkfs(1M), ps(1).

NAME

dd – convert and copy a file

SYNOPSIS

dd [option=value] ...

DESCRIPTION

dd copies the specified input file to the specified output with possible conversions. The standard input and output are used by default. The input and output block size may be specified to take advantage of raw physical I/O.

option	*values*
if=*file*	input file name; standard input is default
of=*file*	output file name; standard output is default
ibs=*n*	input block size *n* bytes (default 512)
obs=*n*	output block size (default 512)
bs=*n*	set both input and output block size, superseding *ibs* and *obs*; also, if no conversion is specified, it is particularly efficient since no in-core copy need be done
cbs=*n*	conversion buffer size
skip=*n*	skip *n* input blocks before starting copy
seek=*n*	seek *n* blocks from beginning of output file before copying
count=*n*	copy only *n* input blocks
conv=**ascii**	convert EBCDIC to ASCII
ebcdic	convert ASCII to EBCDIC
ibm	slightly different map of ASCII to EBCDIC
lcase	map alphabetics to lower case
ucase	map alphabetics to upper case
swab	swap every pair of bytes
noerror	do not stop processing on an error
sync	pad every input block to *ibs*
...,...	several comma-separated conversions
extab=*file*	convert using external user constructed table file. The convert table must be placed from address 0x100 to 0x1ff in the table file. This means that the 0x0 character becomes the value of address 0x100 and the 0xff

character is converted to the value of address 0x1ff.

Where sizes are specified, a number of bytes is expected. A number may end with **k**, **b**, or **w** to specify multiplication by 1024, 512, or 2, respectively; a pair of numbers may be separated by **x** to indicate multiplication.

cbs is used only if *conv=ascii* or *conv=ebcdic* is specified. In the former case, *cbs* characters are placed into the conversion buffer (converted to ASCII). Trailing blanks are trimmed and a new-line added before sending the line to the output. In the latter case, ASCII characters are read into the conversion buffer (converted to EBCDIC). Blanks are added to make up an output block of size *cbs*.

After completion, *dd* reports the number of whole and partial input and output blocks.

DIAGNOSTICS

f+p blocks in(out)	numbers of full and partial blocks read(written)

NAME

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

devnm – device name

SYNOPSIS

/etc/devnm [names]

DESCRIPTION

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

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

EXAMPLE

The command:

```
/etc/devnm /usr
```

produces

```
/dev/dsk/u14c8s1 /usr
```

if **/usr** is mounted on **/dev/dsk/u14c8s1**

FILES

/dev/dsk/ *
/etc/mnttab

SEE ALSO

brc(1M).

This page is intentionally left blank

NAME

df – report number of free disk blocks and i-nodes

SYNOPSIS

df [**-lt**] [**-f**] [*file-system* | *directory* | *mounted-resource*]

DESCRIPTION

The **df** command prints out the number of free blocks and free i-nodes in mounted file systems, directories, or mounted resources by examining the counts kept in the super-blocks.

file-system may be specified either by device name (e.g., **/dev/dsk/c1d0s2**) or by mount point directory name (e.g., **/usr**).

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

mounted-resource can be a remote resource name. The report presents information for the remote device that contains the resource.

If no arguments are used, the free space on all locally and remotely mounted file systems is printed.

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

−l only reports on local file systems.

−t causes the figures for total allocated blocks and i-nodes to be reported as well as the free blocks and i-nodes.

−f an actual count of the blocks in the free list is made, rather than taking the figure from the super-block (free i-nodes are not reported). This option will not print any information about mounted remote resources.

NOTE

If multiple remote resources are listed that reside on the same file system on a remote machine, each listing after the first one will be marked with an asterisk.

FILES

/dev/dsk/ *
/etc/mnttab

SEE ALSO

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

245

NAME

diff – differential file comparator

SYNOPSIS

diff [**–efbh**] file1 file2

DESCRIPTION

diff tells what lines must be changed in two files to bring them into agreement. If *file1* (*file2*) is –, the standard input is used. If *file1* (*file2*) is a directory, then a file in that directory with the name *file2* (*file1*) is used. The normal output contains lines of these forms:

n1 **a** *n3,n4*
n1,n2 **d** *n3*
n1,n2 **c** *n3,n4*

These lines resemble *ed* commands to convert *file1* into *file2*. The numbers after the letters pertain to *file2*. In fact, by exchanging **a** for **d** and reading backward one may ascertain equally how to convert *file2* into *file1*. As in *ed*, identical pairs, where *n1* = *n2* or *n3* = *n4*, are abbreviated as a single number.

Following each of these lines come all the lines that are affected in the first file flagged by <, then all the lines that are affected in the second file flagged by >.

The **–b** option causes trailing blanks (spaces and tabs) to be ignored and other strings of blanks to compare equal.

The **–e** option produces a script of *a, c*, and *d* commands for the editor *ed*, which will recreate *file2* from *file1*. The **–f** option produces a similar script, not useful with *ed*, in the opposite order. In connection with **–e**, the following shell program may help maintain multiple versions of a file. Only an ancestral file ($1) and a chain of version-to-version *ed* scripts ($2,$3,...) made by *diff* need be on hand. A "latest version" appears on the standard output.

```
(shift; cat $ * ; echo '1,$p') | ed – $1
```

Except in rare circumstances, *diff* finds a smallest sufficient set of file differences.

Option **−h** does a fast, half-hearted job. It works only when changed stretches are short and well separated, but does work on files of unlimited length. Options **−e** and **−f** are unavailable with **−h**.

FILES

/tmp/d?????
/usr/lib/diffh for **−h**

SEE ALSO

bdiff(1), cmp(1), comm(1), ed(1).

DIAGNOSTICS

Exit status is 0 for no differences, 1 for some differences, 2 for trouble.

BUGS

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

WARNINGS

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

NAME

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 64K bytes will not work.

NAME

dircmp – directory comparison

SYNOPSIS

dircmp [**–d**] [**–s**] [**–w***n*] dir1 dir2

DESCRIPTION

dircmp examines *dir1* and *dir2* and generates various tabulated information about the contents of the directories. Listings of files that are unique to each directory are generated for all the options. If no option is entered, a list is output indicating whether the 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).

This page is intentionally left blank

NAME

disk_setup – set up your machine for the sysadm 4.0 backup system

SYNOPSIS

disk_setup [–v] [–s]

DESCRIPTION

Can only be run by the superuser.

disk_setup is a utility program giving the user the opportunity to setup the machine for the **Sysadm 4.0 Backup System**.

disk_setup scans the directories **/dev/dsk**, **/dev**, **/dev/rdsk**, **/dev/SAdsk**, the file **/etc/fstab**, and the hardware configuration to obtain information about disks in the system.

Each subdisk is treated as a disk and the user is able to see the obtained information for each disk.

When going through the disks, in 'unit/channel/subdisk' order, the user will be prompted for one of the following questions depending of the mode in which *disk_setup* is run.

1) *disk_setup* could not find all links for a given disk and ask for a new directory to scan.

2) The disk contains a file system and has no label. The user OUGHT to enter a label, but may escape the question if not required.

   The label is used for a unique identification of a disk in the backup system.

3) If the disk contains a file system and the **lost+found** directory is not present, *disk_setup* gives the user the possibility to create the directory.

4) If the disk is not contained in a dataset, the user has to decide if the disk should be included in the backup system, and if so, a name for the dataset must be given.

disk_setup is able to run in the following three modes:

SCAN	(use option **−s**) Could be run in runlevel 1 and 2. Prompts 2, 3 and 4 will not appear.
UPDATE	Can only run in runlevel 1. All prompts can appear.
VERBOSE	(use option **−v**) Identical to mode UPDATE except that the user will see the special shell commandsm, as for instance *mount* instead of a user message, explaining in plain English what the program is doing.

NOTES

Before using *disk_setup* please carefully read through the chapters "*Backup Management*" and "*Backup Administration*" in the "*System Administrator's Guide, System V Release 3.1*" to fully understand the term "dataset", and how to configure the dataset.

If the disk is present in one or more datasets, *disk_setup* could not change this. Use the Sysadm 4.0 Backup System to perform any changes.

disk_setup is able to show a maximum of 4 links to a disk and handle a maximum of 150 disks.

If a label is placed on a disk, the packname (volume name) is set to date of labelling.

NAME

diskformat – format disk

SYNOPSIS

diskformat specialfile

DESCRIPTION

diskformat puts information on a floppy disk.

Formatting (*diskformat*(1)) a disk makes it possible for the hardware to perform I/O operations to a disk.

All floppy disks distributed from **Dansk Data Elektronik a/s** are formatted. No later formatting is therefore ever required for these.

Initializing a disk (*mkfs*(1M)) puts data on the disk so it is possible to look at the disk as a file system.

When *diskformat* is invoked with the **–s** option it works silent and the user will not be asked to confirm the formatting. The exit code indicate if the formatting is succeeded.

SEE ALSO

mkfs(1)

This page is intentionally left blank

NAME

ds, ts, qs – dual, tri, quad session manager

SYNOPSIS

ds [–**b**] [program [prog params]]

ts [–**b**] [program [prog params]]

qs [–**b**] [program [prog params]]

DESCRIPTION

ds (*ts, qs*) allows user to interact with two (three, four) *programs* from a single terminal. *program* is invoked with the *prog params* parameter string. By default, the content of the user enviroment SHELL is invoked. If there exists no such environment, **bin/sh** is invoked. The user controls the two (three, four) *programs* known as *layser*, using the methods described below.

The *current layer* is the layer which can receive input from the keyboard. If other layers attempt to read from the keyboard, they are blocked. Output from the layers is multiplexed onto the terminal. When the –**b** option is used, output to layers that do not receive input is blocked.

The *stty*(1) character switch (set to Control-Z if NULL) is used to switch control from one layer to the next in a cyclic manner.

A *layer* is a program which has been bound to a window on a terminal. Each layer has its own process group id.

The terminal must be configured (using *chhw*(1M)) as a window terminal with a least 2 (3, 4) windows. When the –**b** options is used, the number of windows must be 3 (4, 5) to allow proper operation. The names of the special files that identify the windows must satisfy the requirements:

Window number 1
(the terminal proper) must have the name: **/dev/tty##**, where **##** is some number. This is the name by which the terminal is identified to *getty*(1M) in /etc/inittab.

253

Window number 2
(3,4,5) must have the name **/dev/tty##B**, (/dev/tty##C, /dev/tty##D, /dev/tty##E), where **##** is the same number as above.

NAME

dsh – shell with history facility

SYNOPSIS

dsh [**-acefhiknrstuvx**] [args]

DESCRIPTION

dsh is an alternative to the standard shell, *sh*(1). *dsh* performs exactly the same tasks as *sh*(1), and the reader is referred to the manual pages about *sh*(1) for a description of that program. However, *dsh* can remember the last 22 commands issued by the user, and the program gives the user the possibility to re-issue these commands, possibly with some modifications.

On top of the commands known to *sh*(1) *dsh* has the following commands (which must all start in the first character position of the command):

?? Give a list of the last 22 commands. Each command is identified by two numbers:

A relative number that identifies the command with respect to the most recently issued command. This number is zero or negative.

An absolute number that identifies the command with respect to the first command issued. This number is positive.

!? This command is identical to **??**.

! This command requests *dsh* to re-issue the most recent command. The command will be displayed and then executed.

? This command requests *dsh* to display the most recent command, whereupon the user may edit the command and issue it by pressing the return key.

!*nn* where *nn* is a number (positive, zero, or negative). This command requests *dsh* to re-issue command number *nn*. The command will be displayed and then executed. The number *nn* may be either of the two numbers displayed for each command with the **??** command.

?*nn* where *nn* is a number (positive, zero, or negative). This command requests *dsh* to display command number *nn*,. whereupon the user may edit the command and issue it by pressing the return key. The number *nn* may be either of the two numbers displayed for each command with the **??** command.

!*string* This command requests *dsh* to re-issue the most recent command whose first characters were *string* (leading spaces must be included). The command will be displayed and then executed. The command must be one of the 22 least recently issued commands.

?*string* This command requests *dsh* to display the most recent command whose first characters were *string* (leading spaces must be included). The user may then edit the command and issue it by pressing the return key. The command must be one of the 22 least recently issued commands.

In the above description the term 'command' is used about a command line given to the shell, regardless of whether that line is really a command or just part of one.

NOTE

In order to make full use of the edit facilities of *dsh*, the terminal should be operating in line discipline 1 (see *stty*(1) and *termio*(7)) because this line discipline gives the user a full set of line editing functions.

NAME

dsize – display disk size

SYNOPSIS

dsize specialfiles

DESCRIPTION

dsize displays the sizes of the logical disks specified by the specialfiles.

SEE ALSO

chlds(1M), l_disk(2).

This page is intentionally left blank

NAME

dskback – backup and restore disks

SYNOPSIS

/etc/dskback
[**–feet** length] [**–reel** number] [**–b**] [**–log** logfile] [**–c**] [**–o**] [**–r**] [**–v**] [**–comment** string] source destination

/etc/dskback
–B [**–feet** length] [**–reel** number] [**–b**] [**–log** logfile] [**–c**] [**–r**] [**–v**] [**–comment** string] source [source . . .] destination

/etc/dskback
–T [**–b**] [**–log** logfile] [**–comment**] [**–v**] source

/etc/dskback
–R [**–b**] [**–log** logfile] [**–c**] [**–v**] source entry:destination [entry:destination . . .]

DESCRIPTION

The *dskback* utility is used for making backups of raw disks or copy one raw disk to another raw disk. The *source* and the *destination* parameters must be the names of special files identifying the source and destination medium.

When *dskback* is used for backups, the backup medium has to be a removable medium. *dskback* supports floppies, magtapes, video tapes and streamer tape as removable medium. The removable media will always be labeled by *dskback*, with informations about the size of the original source. If the removable medium is less than the source *dskback* will prompt for the next medium when the previous is full.

If one of the leading control options **–B**, **–T** or **–R** are set, *dskback* expect to operate on a videotape or a 120Mbyte streamer. If no control option is specified *dskback* operate as older versions.

Leading control options:

−B	*Backup* one or more hard disks to video tape or 120Mbyte streamer specified in the last argument. If an error occur when reading from hard disk, *dskback* switch to read in small blocks, to save most possible data. The block unable to read will be substituted by the text **"dskback hard err"** if hard error occur. Other read errors will produce the text **"dskback xxx err "**, where 'xxx' is the SMOS error number. *dskback* will make a list of the first 40 areas of read errors.
−T	*Table of contents* displays files in the directory from the first header block on tape. If an hard error occurs on tape the backup entry is lost, the other disks entries on the tape are still accessible.
−R	*Restore* one or more hard disks from videotape or 120Mbyte streamer specified in the first argument after options. The usage when restoring is `entry:disk` or `entry:`**RESTORE**, where *entry* is the number of the disk on the backup medium. The entry and contents is visualized by using option **−T**. If **RESTORE** is specified, the special file used during backup becomes the destination disk.
No option	If no leading option is specified, the *dskback* will be equal to older versions of dskback.

260

General options known by *dskback* are:

–feet *length*	specify the length of a magtape. If the backup medium is a magtape and the size is not specified by using this option, *dskback* will use the size specified in the operating system.
–reel *number*	tell *dskback* to start restoring from a particular reel given by the parameter *number*.
–b	operate without operator, like backup run by *cron* during night hours. Running without operator limits the processing to situations when one reel is able to contain the entire source medium. Operating in this mode *dskback* uses the prespecified answers placed and maintained in the program code.
–log *logfile*	redirect *stdout* and *stderr* to a specified logfile. Be careful not to place the logfile on the disk being restored or backup copied. Set option **–b**.
–c	verify the backup after each reel is written. When detecting 40 error *dskback* skip printing the addresses of difference.
–o	read a backup copy made by the first version of *dskback* February 1987.
–r	perform a retension of the tape before writing, to make more reliable copies. It is to be recommended always to use this option. This operation does only exist in new system releases. On previous operating systems retension results in an error which terminates *dskback*.

261

−v	used for informations during operation, otherwise *dskback* will remain silent if the operations are correctly performed. Used together with option −T this option displays more information.
−comment *string*	will place the string in the label on the removable medium. Except from this operation this string will be ignored by *dskback*. The string is placed at the address 0x200 in the label on the removable medium. If only this option and option **−T** are specified, *dskback* returns the comment placed in the label if the **−comment** option was used when the backup was created.

EXAMPLES

Old syntax of backup:

```
/etc/dskback −log /etc/backuplog −c −r −v
   −comment "special backup database b"
   /dev/dsk/u14c8s4     /dev/stream
```

Old syntax of restore:

```
/etc/dskback −log /etc/backuplog −c −v
   /dev/stream    /dev/dsk/u14c8s4
```

Backup of one or more disks:

```
/etc/dskback −B −Log /etc/backuplog −c −v
   /dev/dsk/u12c8s1
   /dev/dsk/u12c12s1
   /dev/dsk/u12c12s2
   /dev/dsk/u14c8s0
   /dev/dsk/u14c8s1
   /dev/video
```

262

Table of contents:

```
/etc/dskback -T -v  /dev/video
```

Get user comnment:

```
/etc/dskback -T -comment  /dev/stream
```

Restore one or more disks:

```
/etc/dskback -R -Log /etc/backuplog -c -v\
  /dev/video  2:RESTORE  4:/dev/dsk/u14c8s4
```

SEE ALSO

dd(1), ff(1M), frec(1M), volcopy(1M).

BUGS

Option **−B**, **−T** and **−R**, in version 3.0 of *dskback* dated December 1988, cannot handle backup series larger than one streamer tape or video tape. Language system is not implemented. Using **−B** when backing up, needs **−R** to restore. Making backup using no control option cannot be restored by using **−R**.

NOTE

If the size of the video tape is different to the configuration parameter the size can be adjusted by the **−feet** option. *dskback* converts one feet to 39000 byte.

53770 feet	=	2000Mbyte
26885 feet	=	1000Mbyte
13442 feet	=	500Mbyte

263

This page is intentionally left blank

NAME

du – summarize disk usage

SYNOPSIS

du [**–Lsar**] [names]

DESCRIPTION

du reports the number of blocks contained in all files and (recursively) directories within each directory and file specified by the *names* argument. The block count includes the indirect blocks of the file. If *names* is missing, the current directory is used.

The optional arguments are as follows:

–s causes only the grand total (for each of the specified *names*) to be given.

–a causes an output line to be generated for each file.

If neither **–s** or **–a** is specified, an output line is generated for each directory only.

–r will cause *du* to generate messages about directories that cannot be be read, files that cannot be opened, etc., rather than being silent (the default).

–L causes *du* to follow symbolic links. Note that this can result in looping if the symbolic link points to a parent of the directory containing the link.

A file with two or more links is only counted once.

BUGS

If the **–a** option is not used, non-directories given as arguments are not listed. Files with holes in them will get an incorrect block count.

This page is intentionally left blank

NAME

echo – echo arguments

SYNOPSIS

echo [arg] ...

DESCRIPTION

echo writes its arguments separated by blanks and terminated by a new-line on the standard output. It also understands C-like escape conventions; beware of conflicts with the shell's use of \:

\b	backspace
\c	print line without new-line
\f	form-feed
\n	new-line
\r	carriage return
\t	tab
\v	vertical tab
\\	backslash
\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)

This page is intentionally left blank

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 at the end of this manual page.

–C Encryption option; the same as the **–x** option, except that *ed* simulates a **C** command. The **C** command is like the **X** command, except that all text read in is assumed to have been encrypted.

ed operates on a copy of the file it is editing; changes made to the copy have no effect on the file until a *w* (write) command is given. The copy of the text being edited resides in a temporary file called the *buffer*. There is only one buffer.

red is a restricted version of *ed*. It 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 below) is a one-character RE that matches itself.

1.2 A backslash (\) followed by any special character is a one-character RE that matches the special character itself. The special characters are:

a. **.**, *****, **[**, and \ (period, asterisk, left square bracket, and backslash, respectively), which are always special, *except* when they appear within square brackets (**[ ]**; see 1.4 below).

b. **^** (caret or circumflex), which is special at the *beginning* of an *entire* RE (see 3.1 and 3.2 below), or when it immediately follows the left of a pair of square brackets (**[ ]**) (see 1.4 below).

c. **$** (dollar sign), which is special at the *end* of an entire RE (see 3.2 below).

d. The character used to bound (i.e., delimit) an entire RE, which is special for that RE (for example, see how slash (**/**) is used in the *g* command, below.)

1.3 A period (**.**) is a one-character RE that matches any character except new-line.

1.4 A non-empty string of characters enclosed in square brackets (**[ ]**) is a one-character RE that matches *any one* character in that string. If, however, the first character of the string is a circumflex (**^**), the one-character RE matches any character *except* new-line and the remaining characters in the string. The **^** has this special meaning *only* if it occurs first in the string. The minus (**−**) may be used to indicate a range of consecutive ASCII characters; for example, **[0−9]** is equivalent to **[0123456789]**. The **−** loses this special meaning if it occurs first (after an initial **^**, if any) or last in the string. The right square bracket (**]**) does not terminate such a string when it is the first character within it (after an initial **^**, if any); e.g., **[ ]a−f]** matches either a right square bracket (**]**) or one of the letters **a** through **f** inclusive. The four characters listed in 1.2.a above stand for themselves within such a string of characters.

The following rules may be used to construct *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).

272

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 **FILES** below.

To understand addressing in *ed* it is necessary to know that at any time there is a *current line*. Generally speaking, the current line is the last line affected by a command; the exact effect on the current line is discussed under the description of each command. *Addresses* are constructed as follows:

1. The character **.** addresses the current line.
2. The character **$** addresses the last line of the buffer.
3. A decimal number *n* addresses the *n*-th line of the buffer.
4. '*x* addresses the line marked with the mark name character *x*, which must be an ASCII lower-case 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 **FILES** below.
6. A RE enclosed in question marks (**?**) addresses the first line found by searching *backward* from the line *preceding* the current line toward the beginning of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the end of the buffer and continues up to and including

273

the current line. See also the last paragraph before **FILES** below.

7. An address followed by a plus sign (+) or a minus sign (−) followed by a decimal number specifies that address plus (respectively minus) the indicated number of lines. The plus sign may be omitted.

8. If an address begins with + or −, the addition or subtraction is taken with respect to the current line; e.g, **−5** is understood to mean **.−5**.

9. If an address ends with + or −, then 1 is added to or subtracted from the address, respectively. As a consequence of this rule and of Rule 8, immediately above, the address − refers to the line preceding the current line. (To maintain compatibility with earlier versions of the editor, the character ^ in addresses is entirely equivalent to −.) Moreover, trailing + and − characters have a cumulative effect, so − − refers to the current line less 2.

10. For convenience, a comma (**,**) stands for the address pair **1,$**, while a semicolon (**;**) stands for the pair **.,$**.

Commands may require zero, one, or two addresses. Commands that require no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when an insufficient number of addresses is given; if more addresses are given than such a command requires, the last one(s) are used.

Typically, addresses are separated from each other by a comma (**,**). They may also be separated by a semicolon (**;**). In the latter case, the current line (**.**) is set to the first address, and only then is the second address calculated. This feature can be used to determine the starting line for forward and backward searches (see Rules 5 and 6, above). The second address of any two-address sequence must correspond to a line that follows, in the buffer, the line corresponding to the first address.

In the following list of *ed* commands, the default addresses are shown in parentheses. The parentheses are *not* part of the address; they show that the given addresses are the default.

It is generally illegal for more than one command to appear on a line. However, any command (except *e*, *f*, *r*, or *w*) may be suffixed by **l**, **n**, or **p** in which case the current line is either listed, numbered or printed, respectively, as discussed below under the *l*, *n*, and *p* commands.

(.)a
<text>
.

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 new-line 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 typed in.

(.,.)d

The *d*elete command deletes the addressed lines from the buffer. The line after the last line deleted becomes the current line; if the lines deleted were originally at the end of the buffer, the new last line

275

becomes the current line.

e *file*

The *e*dit command causes the entire contents of the buffer to be deleted, and then the named file to be read in; **.** is set to the last line of the buffer. If no file name is given, the currently remembered file name, if any, is used (see the *f* command). The number of characters read is typed; *file* is remembered for possible use as a default file name in subsequent *e*, *r*, and *w* commands. If *file* is replaced by **!**, the rest of the line is taken to be a shell (*sh*(1)) command whose output is to be read. Such a shell command is *not* remembered as the current file name. See also **DIAGNOSTICS** below.

E *file*

The *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** below.

(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 new-line 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).

277

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 new-line character).

(.,.+1)j

The *j*oin command joins contiguous lines by removing the appropriate new-line characters. If exactly one address is given, this command does nothing.

(.)k*x*

The mar*k* command marks the addressed line with name *x*, which must be an ASCII lower-case 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**, below.

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

279

number n appears after the command, only the n th occurrence of the matched string on each addressed line is replaced. It is an error for the substitution to fail on *all* addressed lines. Any character other than space or new-line may be used instead of / to delimit the RE and the *replacement*; **.** is left at the last line on which a substitution occurred. See also the last paragraph before **FILES** below.

An ampersand (**&**) appearing in the *replacement* is replaced by the string matching the RE on the current line. The special meaning of **&** in this context may be suppressed by preceding it by \. As a more general feature, the characters **\***n*, where *n* is a digit, are replaced by the text matched by the *n*-th regular subexpression of the specified RE enclosed between **\\(** and **\\)**. When nested parenthesized subexpressions are present, *n* is determined by counting occurrences of **\\(** starting from the left. When the character *%* is the only character in the *replacement*, the *replacement* used in the most recent substitute command is used as the *replacement* in the current substitute command. The *%* loses its special meaning when it is in a replacement string of more than one character or is preceded by a \.

A line may be split by substituting a new-line character into it. The new-line in the *replacement* must be escaped by preceding it by \. Such substitution cannot be done as part of a *g* or *v* command list.

(. , .)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,

280

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 UNIX system shell (*sh*(1)) to be interpreted as a command. Within the text of that command, the unescaped character **%** is replaced with the remembered file name; if a **!** appears as the first character of the shell command, it is replaced with the text of the previous shell command. Thus, **!!** will repeat the last shell command. If any expansion is performed, the expanded line is echoed; **.** is unchanged.

(.+1)<new-line>

An address alone on a line causes the addressed line to be printed. A new-line alone is equivalent to **.+1p**; it is useful for stepping forward through the buffer.

If an interrupt signal (ASCII DEL or BREAK) is sent, *ed* prints a **?** and returns to *its* command level.

Some size limitations: 512 characters 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 new-line 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 new-line, that delimiter may be omitted, in which case the addressed line is printed. The following pairs of commands are equivalent:

```
s/s1/s2     s/s1/s2/p
g/s1        g/s1/p
?s1         ?s1?
```

FILES

$TMPDIR	if this environmental variable is not null, its value is used in place of **/usr/tmp** as the directory name for the temporary work file.
/usr/tmp	if **/usr/tmp** exists, it is used as the directory name for the temporary work file.
/tmp	if the environmetal variable **TMPDIR** does not exist or is null, and if **/usr/tmp** does not exist, then **/tmp** is used as the directory name for the temporary work file.
ed.hup	work is saved here if the terminal is hung up.

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 new-line 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 filename: **edit** *name*; the editor will tell you it is a `[New File]`.

The *edit* command prompt is the colon (**:**), which you should see after starting the editor. If you are editing an existing file, 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 **^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 wish to move or copy a section of text within a file or between files. Find the first and last line numbers you wish to copy or move. To move lines 10 through 20, type **10,20d a** to delete these lines from the file and place them in a buffer named **a**. *edit* has 26 such buffers named **a** through

z. To put the contents of buffer **a** after the current line, type **put a**. If you want to move or copy these lines to another file, execute an **edit** (**e**) command after copying the lines; 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 with the Security Administration Utilities package, which is available only in the United States.

289

This page is intentionally left blank

NAME

editor – edit text files

SYNOPSIS

editor [–S] [file]

editor r

DESCRIPTION

The editor is used to create and modify text files. It is a screen oriented editor and will work properly only when the terminal type has been properly specified through the *terminology*(1) program.

Starting the *editor*

editor can write it's messages to the user in either English or Danish. Default language is English. This can be changed through the environment LANGUAGE:

```
$ LANGUAGE=dk
$ export LANGUAGE
```

causes *editor* to use Danish messages.

editor uses a default value of the environment NLSPATH, if this is not set. If however NLSPATH is set for use by other applications, the path needed by editor must be appended to the NLSPATH environment:

```
$ NLSPATH=$NLSPATH:/nlslib/%N/%L
```

Case 1: You want to modify an existing file, called myfile.

Give the command

editor myfile

This will load the *editor*, and after a while the contents of *myfile* will be presented on the terminal screen.

Alternatively, give the command

editor

This will load the *editor*, and it will prompt you for the file name. Type "**myfile**" and press the RETURN key. After a while the contents of *myfile* will be presented on the terminal screen.

If anything goes wrong during the loading of the file (if, for example, the file does not exist), *editor* will present the error on the screen and give the operator the following three options:

'E' – Edit a new file
'C' – Continue editing
'U' – exit to Supermax Operating System

The operator must now enter one of the letters **E**, **C**, or **U**. Entering **E** will cause *editor* to prompt for a new file name, entering **C** will cause *editor* to proceed as if no file name had been entered (see case 2 below), entering **U** will stop *editor*.

Case 2: You want to create a new file.

Give the command

editor

This will load the *editor*, and it will prompt you for a file name. Press the RETURN key. The screen will now be cleared and you may start entering the contents of the new file.

Entering text

Pressing one of the 'normal' keys will cause the corresponding character to appear at the cursor position, possibly replacing any character that was already there. In this context a 'normal' key is any key corresponding to a printable character, for example, 'A', '6', or space.

Moving around the text

The four arrow keys will move the cursor one column or line in the specified direction. If the cursor is at the bottom line and 'down arrow' is pressed, the text on the screen will scroll upwards, unless end-of-file is at the top of the screen. If the cursor is at the top line and 'up arrow' is pressed, the text on the screen will scroll downwards, unless beginning-of-file is at the top of the screen.

The function key 'E.O.L.' (End of line, or double right arrow) will move the cursor to the position after the last character on the current line.

The function key 'S.O.L.' (Start of line, or double left arrow) will move the cursor to the leftmost column.

The HOME key will move the cursor to the upper left corner of the screen. Pressing the HOME key again will move the cursor back to where it came from.

Pressing the 'Special' key and the function keys 'First page' and 'Last page' will display the first or last screenful of text, respectively.

The function keys 'Next page' and 'Prev page' will display the next or previous screenful of text, respectively.

To go to a specific line number, press the 'Status' function key (see below), enter the desired line number, and press RETURN.

The TAB key (not to be confused with the 'Tab' function key) will move the cursor to the next tab stop. Note: The TAB key does *not* insert a tab character into the text, it is merely a cursor moving key.

The RETURN key will move the cursor to the next line. The cursor position on the new line is determined as follows:

- If the new line contains text, the cursor will be placed on the first non-blank character in that line.
- If the new line is blank and the previous line contains text, the cursor will be placed under the first non-blank character in the previous line.
- If both the new line and the previous line are blank, the cursor will be placed at the leftmost column.
- If the left margin (see below) is set to a position to the right of the column, which the cursor would choose according to the preceding rules, the cursor

will be set at the left margin rather than according to the above-mentioned rules.

Function keys

editor makes use of the socalled hard and soft function keys where the function keys f1 to f8 and F1 to F8 have the same meaning in different applications. The socalled soft function keys f9 to f16 and F9 to F16 contain the rest functions. The meaning of these can be read by pressing the 'Help' function key whereafter a helpline explaining the use of the soft keys will appear. The helpline can be removed by pressing 'Help' again. Some functions require pressing two function keys: 'Special' can be combined with some of the rest function keys. If the helpline is present on the terminal screen pressing 'Special' will give a new helpline explaining which function keys can be used.

Inserting and deleting

The function key 'Insert character' will insert a space in the current line at the cursor position.

The function key 'Delete character' will delete the character at the cursor position.

The function key 'Insert line' will insert a blank line before the line containing the cursor.

The function key 'Delete line' will delete the line containing the cursor.

If by mistake you have deleted a line which you wish to retain, pressing the 'Special' function key and thereupon the 'Insert line' function key will restore the deleted line.

The DEL (or RUB OUT) key moves the cursor one position to the left and deletes the character in this position.

Searching and replacing

If you want to find, say, the characters 'alpha' in the text, press the 'Find' function key. The editor will then prompt you for a search sting; type **alpha** and press RETURN. The cursor will now be moved to the first occurrence of 'alpha' following the current position.

If you want to find the next occurrence of 'alpha', press the 'Find again' function key. This will move the cursor to the next occurrence of the string.

If you wish to find the text 'alpha' and replace it with 'beta', press the 'Replace' function key. You will now be prompted for the search string (type: **alpha**) and the replacement string (type: **beta**). Finally, you will be asked if you want to confirm the replacement. If you answer No (or rather, **N**) to this question, the editor will merely replace the first occurrence of 'alpha' after the current position with 'beta'. If you answer Yes (**Y**) to the question, the editor will move the cursor to 'alpha', the terminal will beep and you must press the RETURN key if you want the replacement to be carried out; pressing any other key will cause the editor to leave the text unchanged.

The 'Replace again' function key will repeat the last replacement command.

If your search string or replacement string contains trailing blanks, it must be enclosed in apostrophes.

Moving and Copying

A portion of the text may be moved or copied to another place in the text. The text to be moved or copied must be completely visible on the screen before the following operations are carried out.

The operation consists of two steps:

1) Copy the text into a save buffer. The text may be removed or retained in its original position.
2) Move to the place where the content of the save buffer is to be inserted, and insert it there. The text may be inserted at several different places.

Step 1: Version A: Copying lines into the save buffer.

Place the cursor on the first (or last) line to be moved and press the 'Mark' function key. Use the up-arrow key, the down-arrow key, or the HOME key to move the cursor to the last (or first) line to be moved and press the 'Mark' function key again. The two lines delimiting the text to be moved will now stand out on the screen. Press the 'Save' (or 'Store') function key; this will copy the marked text into the save buffer, from where it may later be retrieved. If the 'Save and delete' function key is used instead of the 'Save' function key, the text will be copied into the save buffer and deleted from its original position.

Version B: Copying columns of text into the save buffer.
Place the cursor on the first (or last) line of the columns to be moved and press the 'Mark' function key. Use the up-arrow key, the down-arrow key, or the HOME key to move the cursor to the last (or first) line of the columns to be moved and press the 'Mark' function key again. The two lines delimiting the text to be moved will now stand out on the screen. Use the left-arrow key and the right-arrow key to move the cursor to the leftmost column delimiting the columns to be moved and press the 'Mark' function key. Use the left-arrow and the right-arrow key to moved the cursor to the rightmost column delimiting the columns to be moved and press the 'Mark' function key again. The columns to be moved will now stand out on the screen. Press the 'Save' (or 'Store') function key; this will copy the marked text into the save buffer, from where it may later be retrieved. If the 'Save and delete' function key is used instead of the 'Save' function key, the text will be copied into the save buffer and deleted from its original position.

Now move to the position in the text where the content of the save buffer is to be inserted.

Step 2: Version A: Inserting lines from the save buffer.
Pressing the 'Restore' function key will cause the text in the save buffer to be inserted before the current line. The editor will use insert-line operations to make room for the text.

Version B: Inserting columns of text from the save buffer.

Pressing the 'Special' function key and then the 'Special recall' function key will cause the text in the save buffer to be inserted into the current and following lines at the column indicated by the cursor. The editor will use insert-character operations to make room for the text.

Reading from and Writing to Other Files

If you wish to save some lines of the text in another file, press the 'Write to file' function key. The editor will prompt you for the filename and the first and last line to be written.

If you wish to read lines from another file into the text you are editing, press the 'Read from file' function key. The editor will prompt you for the filename and ask if you want 'Line or Key reading'. You must enter **L** or **K** to choose the option and proceed as follows:

Line reading:

The editor will ask you for the line number of the first and last line to be read. If you want to read the entire file, type, for example, 1 and 9999. The lines will be inserted before the current line. If the file contains too few lines, you will be told so, and you must press the RETURN key to proceed.

Key reading.

The editor will ask you for a search string found in the first line you wish to read. The file will be read until the string is found, and the editor will ask you if the line found is indeed the line you want. If not, searching will continue. When the first line has been found, the editor will ask you for a search string found in the last line you wish to read. Lines from the file will now be copied into the text before the current line until the final search string is found. The editor will again ask you if the line found is indeed the line you want. If not, searching will continue.

297

Status

The 'Status' function key will present on the screen information about the current cursor position, the name of the file being edited, etc.

You may revert to editing be pressing the RETURN key. If a number is typed before pressing the RETURN key, the editing will continue at that line number.

If for some reason the contents of the screen has been corrupted, pressing 'Status' and then RETURN is an easy way to redraw the screen.

Open line

Pressing the 'Special' key and then the 'Open line' function key will split the current line in two at the cursor position.

Tab

The 'Tab' function key may be used to set the left margin and tab stops. Pressing the 'Tab' function key will display a ruler containing the letter L at the left margin position and the letter T at each tab stop position. This ruler may now be edited, placing the L and T's in different positions. Any other character will be treated as a dummy. When the ruler looks as it should, press the RETURN key, this will display the new ruler as the editor sees it (dummy characters removed). Pressing the RETURN key again takes you back to editing.

Repeat

Most commands may have a repeat factor. Press the 'Repeat' function key and enter a number. This number will appear in the upper right corner. The DEL (or RUB OUT) key will zero the number. After entering the number you may issue a command. This command will then be executed as many times as the number indicates.

A few examples:

'Repeat' 10 a

will enter ten a's into the text.

'Repeat' 20 'Insert line'
will insert twenty blank lines into the text.

'Repeat' 3 'Find'
will find the third occurrence of the search string following the current position.

'Repeat' 4 'Replace'
will replace the first four occurrences of one string with another, asking for confirmation each time, if requested.

'Repeat' 2 'Restore'
will twice copy the contents of the save buffer into the text.

Starting other programs

You can start programs while you are editing a text. When the 'Special' function key is pressed followed by the 'Terminate' function key, the program specified in the user's SHELL environment is started. If this environment is not set, the program /bin/dsh will be started. When the program is terminated the user will return to *editor*.

If *editor* was started

editor −S [file]

it is not possible to start other programs inside *editor*.

Aborting commands

Where applicable, commands may be aborted by pressing the 'Cancel' function key. The 'Attention' key (ctrl 6) can be used to abort an ongoing 'Find', 'Replace', First page' or 'Last page'.

Finishing

When you have finished editing, press the 'Finish' (or 'Terminate') function key. The editor will now require you to make two decisions:

1) What should be done with the edited text?
   a) You may save it in the file from which it was originally read or in another file.

b) You may choose not to save the changed text. In this case the original file contents will be retained.

2) What do you want to do now?
   a) You may continue editing the same text.
   b) You may edit a new file.
   c) You may terminate the editor.

You make your choice by entering a command of up to two letters. One letter indicates your answer to question 1, another letter indicates your answer to question 2:

The letter **S** indicates that you have chosen option 1a. The absence of the letter **S** indicates that you have chosen option 1b.

The letter **E** indicates that you have chosen option 2b. The letter **U** indicates that you have chosen option 2c. The absence of the letters **E** and **U** indicates that you have chosen option 2a.

Thus merely pressing the RETURN key chooses options 1b and 2a, taking you back to editing the text without saving anything.

If you choose option 1a, the editor will prompt you for the name of the file in which to save the text. It will suggest the file, if any, from which the text was originally read; you may change the file name if you want to, and then press the RETURN key. If you choose option 1b, and 2b or 2c, the editor will require you to enter return for not saving or enter attention. By pressing the attention key you get the possibility to store the file.

Auxiliary files

During the editing the editor creates two files for intermediate storage. These files are called, for example, */tmp/e1090201dde* and */tmp/e2090201dde*. The numbers in these file names are derived from the terminal device number and the name of the user. The username is taken from the environment LOGNAME. If this environment is not set, the username NN will be used. When the editing is finished the editor deletes these files.

If for some reason the editor or the whole system crashes during an edit operation, the edited text is not completely lost. By starting the editor with the *shell* command

```
editor r
```

the text to be edited will be read from the auxiliary files rather than a file called 'r'. This will probably not bring back the whole edited text, but still, it is better than nothing.

BUGS

editor cannot handle files containing non-printable characters, excluding TABs. A TAB character is treated as a blank character.

editor truncates long lines to a length of 78 characters.

301

This page is intentionally left blank

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 new-line.

If no files are specified, *egrep* assumes standard input. Normally, each line found is copied to the standard output. The file name is printed before each line found if there is more than one input file.

Command line options are:

−b Precede each line by the block number on which it was found. This can be useful in locating block numbers by context (first block is 0).

−c Print only a count of the lines that contain the pattern.

−i Ignore upper/lower case distinction during comparisons.

−l Print the names of files with matching lines once, separated by new-lines. Does not repeat the names of files when the pattern is found more than once.

−n Precede each line by its line number in the file (first line is 1).

−v Print all lines except those that contain the pattern.

−e *special_expression*
Search for a *special expression* (*full regular expression* that begins with a −).

−f *file*
Take the list of *full regular expressions* from *file*.

SEE ALSO

ed(1), fgrep(1), grep(1), sed(1), sh(1).

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files (even if matches were found).

BUGS

Ideally there should be only one *grep* command, but there is not a single algorithm that spans a wide enough range of space-time tradeoffs. Lines are limited to 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 [**–c**] [**–r**[reason]] printers

DESCRIPTION

enable activates the named *printers*, enabling them to print requests taken by *lp*(1). Use *lpstat*(1) to find the status of printers.

disable deactivates the named *printers*, disabling them from printing requests taken by *lp*(1). By default, any requests that are currently printing on the designated printers will be reprinted in their entirety either on the same printer or on another member of the same class. Use *lpstat*(1) to find the status of printers. Options useful with *disable* are:

–c	Cancel any requests that are currently printing on any of the designated printers.
–r[*reason*]	Associates a *reason* with the deactivation of the printers. This reason applies to all printers mentioned up to the next **–r** option. If the **–r** option is not present or the **–r** option is given without a reason, then a default reason will be used. *reason* is reported by *lpstat*(1).

FILES

/usr/spool/lp/ *

SEE ALSO

lp(1), lpstat(1).

305

This page is intentionally left blank

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).

This page is intentionally left blank

NAME

errlog – log system errors

SYNOPSIS

errlog [–o file **]**

DESCRIPTION

errlog starts a daemon process that reads system error records from the special file */dev/error* and writes the formatted records to the logfile */usr/lib/errlog/log*. If a file is specified using the **–o** option, records will be written to that file as well. All messages should be self-explanatory.

EXAMPLE

```
errlog –o /dev/console
```

will start *errlog* so that all messages will be sent to the system console.

FILES

/usr/lib/errlog/log	logfile for resulting error records.

SEE ALSO

error(4)

This page is intentionally left blank

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 **^D** (control-d) causes the editor to scroll a half-window of text and is more useful for quickly stepping through a file than just typing return. Of course, the screen-oriented **visual** mode gives constant access to editing context.

ex gives you help when you make mistakes. The **undo** (**u**) command allows you to reverse any single change which goes astray. *ex* gives you a lot of feedback, normally printing changed lines, and indicates when more than a few lines are affected by a command so that it is easy to detect when a command has affected more lines than it should have.

The editor also normally prevents overwriting existing files, unless you edited them, so that you do not accidentally overwrite a file other than the one you are editing. If the system (or editor) crashes, or you accidentally hang up the telephone, you can use the editor **recover** command (or **−r** *file* option) to retrieve your work. This will get you back to within a few lines of where you left off.

ex has several features for dealing with more than one file at a time. You can give it a list of files on the command line and use the **next** (**n**) command to deal with each in turn. The **next** command can also be given a list of 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 lower-case 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 **^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 at the end

313

of this manual page.

−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 ^\ (control-\).

ex Command Names and Abbreviations

abbrev	**ab**	**map**		**set**	**se**
append	**a**	**mark**	**ma**	**shell**	**sh**
args	**ar**	**move**	**m**	**source**	**so**
change	**c**	**next**	**n**	**substitute**	**s**
copy	**co**	**number**	**nu**	**unabbrev**	**unab**
delete	**d**	**preserve**	**pre**	**undo**	**u**
edit	**e**	**print**	**p**	**unmap**	**unm**
file	**f**	**put**	**pu**	**version**	**ve**
global	**g**	**quit**	**q**	**visual**	**vi**
insert	**i**	**read**	**r**	**write**	**w**
join	**j**	**recover**	**rec**	**xit**	**x**
list	**l**	**rewind**	**rew**	**yank**	**ya**

ex Commands

shell escape	**!**
forced encryption	**C**
heuristic encryption	**X**
lshift	**<**
print next	CR
resubst	**&**
rshift	**>**
scroll	**^D**
window	**z**

ex Command Addresses

n	line *n*	*/pat*	next with *pat*
.	current	*?pat*	previous with *pat*
$	last	*x-n*	*n* before *x*
+	next	*x,y*	*x* through *y*
−	previous	*´x*	marked with *x*
+*n*	*n* forward	´´	previous context
%	1,$		

Initializing options

EXINIT	place **set**'s here in environment variable
$HOME/.exrc	editor initialization file
./.exrc	editor initialization file
set *x*	enable option *x*
set no*x*	disable option *x*
set *x*=*val*	give value *val* to option *x*
set	show changed options
set all	show all options
set *x***?**	show value of option *x*

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

316

AUTHOR

vi and *ex* are based on software developed by The University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

FILES

/usr/lib/exstrings	error messages
/usr/lib/exrecover	recover command
/usr/lib/expreserve	preserve command
/usr/lib/terminfo/ *	describes capabilities of terminals
$HOME/.exrc	editor startup file
./.exrc	editor startup file
/tmp/Ex*nnnnn*	editor temporary
/tmp/Rx*nnnnn*	named buffer temporary
/usr/preserve/*login*	preservation directory (where *login* is the user's login)

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*.

317

SEE ALSO

crypt(1), ed(1), edit(1), grep(1), sed(1), sort(1), vi(1), curses(3X), term(4), terminfo(4).
User's Guide.
Editing Guide.
curses/terminfo chapter of the *Programmer's Guide*.

WARNINGS

The encryption options and commands are provided with the Security Administration Utilities package, which is available only in the United States.

BUGS

The **z** command prints the number of logical rather than physical lines. More than a screen full of output may result if long lines are present.

File input/output errors do not print a name if the command line **−s** option is used.

There is no easy way to do a single scan ignoring case.

The editor does not warn if text is placed in named buffers and not used before exiting the editor.

Null characters are discarded in input files and cannot appear in resultant files.

NAME

expr – evaluate arguments as an expression

SYNOPSIS

expr arguments

DESCRIPTION

The arguments are taken as an expression. After evaluation, the result is written on the standard output. Terms of the expression must be separated by blanks. Characters special to the shell must be escaped. Note that **0** is returned to indicate a zero value, rather than the null string. Strings containing blanks or other special characters should be quoted. Integer-valued arguments may be preceded by a unary minus sign. Internally, integers are treated as 32-bit, 2s complement numbers.

The operators and keywords are listed below. Characters that need to be escaped are preceded by \. The list is in order of increasing precedence, with equal precedence operators grouped within { } symbols.

expr \| *expr*
returns the first *expr* if it is neither null nor **0**, otherwise returns the second *expr*.

expr **\&** *expr*
returns the first *expr* if neither *expr* is null or **0**, otherwise returns **0**.

expr { =, \>, \>=, \<, \<=, != } *expr*
returns the result of an integer comparison if both arguments are integers, otherwise returns the result of a lexical comparison.

expr { +, − } *expr*
addition or subtraction of integer-valued arguments.

expr { * , /, % } *expr*
multiplication, division, or remainder of the integer-valued arguments.

319

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 path name (i.e., file). Watch out for **/** alone as an argument: *expr* will take it as the division operator (see BUGS below).

3. `#  A better representation of example 2.`
   `expr  //$a  :  '.*/\(.*\)'`

   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=
```

This page is intentionally left blank

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.

This page is intentionally left blank

NAME

fedit, flook – inspect and edit data file or named partition

SYNOPSIS

fedit pathname
fedit **–p** partition-name
flook pathname
flook **–p** partition-name

DESCRIPTION

flook may be used to display the contents of a file or a named partition in hexadecimal and ASCII form. *fedit* further gives the possibility of editing the contents of the file or disk or partition.

The *pathname* specified in the command line is the name of the file to be inspected. The *partition-name* is the name of the partition to be inspected. The rest of this description will refer only to files, but partitions are handled in an analogous manner.

When the command has been issued, the first 256 bytes of the file will be displayed in hexadecimal and character form, whereupon the user may issue commands.

The character display displays printable characters directly, non-printable characters are displayed as an alternative-intensity period.

flook and *fedit* allow the following commands:

A hexadecimal number
The contents of 256 bytes starting at the specified location in the file will be displayed.

Function key F1 (key value 0x01 0x40) or end–of–file key
The program terminates.

Function key F10 (key value 0x01 0x49)
The program will prompt the user for a search string. It will then search forward in the file for this string and display the data where the string is found.

Function key SHIFT/F10 (key value 0x01 0x69)
The program will repeat the search for the last entered string.

Function key F11 (key value 0x01 0x4a)
fedit only. Enter update mode (see below).

Function key F13 (key value 0x01 0x4c)
The contents of the next 256 bytes will be displayed.

Function key SHIFT/F13 (key value 0x01 0x6c)
The contents of the previous 256 bytes will be displayed.

Function key F15 (key value 0x01 0x4e)
The contents of the first 256 bytes of the file will be displayed.

Function key F16 (key value 0x01 0x4f)
The contents of the last 256 bytes of the file will be displayed. The starting address of the displayed data will be at a 256-byte boundary.

Line feed or down – arrow (key value 0x0a)
The data displayed will be scrolled up 16 bytes.

Up – arrow (key value 0x0c)
The data displayed will be scrolled down 16 bytes.

In *fedit* function key F11 puts the program in update mode, where the user may change the contents of the buffer displayed on the screen. (Actually *flook* may also be put in update mode, but can never write the changed buffer back onto the file.) When the program has been put in update mode, two cursors appear on the screen, one in the hexadecimal display and one at corresponding location in the character display. At the top of the screen, the user may see if data is currently being entered data in HEX mode or ASCII (character) mode.

The basic command in update mode is merely to type a character which is then inserted at the position indicated by the cursors. When input is done in ASCII mode, any printable character (and most non-printable ones) may be typed. When input is done in HEX mode, hexadecimal digits may be typed. Note that both the character and the hexadecimal part of the screen is updated as characters are typed. If characters are typed at a location after end-of-file, the file is extended to include this new location.

In addition the following commands are allowed in update mode:

The arrow keys
These keys move the cursors around the buffer.

The home key (key value 0x1e)
Move the cursors to the first byte on the screen.

Function key F1 (key value 0x01 0x40)
Exit update mode. The program will ask whether the updated buffer is to be saved in the file or not.

Function key F9 (key value 0x01 0x48)
Go to HEX input mode.

Function key SHIFT/F9 (key value 0x01 0x68)
Go to ASCII input mode.

Function key F11 (key value 0x01 0x4a)
Set end-of-file before the position pointed to by the cursors. This is only allowed if the current end-of-file is on the screen or immediately following the last byte on the screen, a situation which is indicated at the upper right corner on the screen. This command is also illegal if the current cursor position is after the current end-of-file. Thus *fedit* can be used to move end-of-file backwards through a file, but only at a rate of 256 bytes a time.

Moving end-of-file of course does not work when the file inspected is a disk or if a partition is being inspected.

When inspecting directories, *fedit* cannot be used because it opens the file in update mode.

fedit and *flook* are just two links to the same file.

SEE ALSO

od(1).

NAME

ff – list file names and statistics for a file system

SYNOPSIS

/etc/ff [options] special

DESCRIPTION

ff reads the i-list and directories of the *special* file, assuming it is a file system. I-node data is saved for files which match the selection criteria. Output consists of the path name for each saved i-node, plus other file information requested using the print *options* below. Output fields are positional. The output is produced in i-node order; fields are separated by tabs. The default line produced by *ff* is:

path-name i-number

With all *options* enabled, output fields would be:

path-name i-number size uid

The argument *n* in the *option* descriptions that follow is used as a decimal integer (optionally signed), where +*n* means more than *n*, −*n* means less than *n*, and *n* means exactly *n*. A day is defined as a 24 hour period.

−I	Do not print the i-node number after each path name.
−l	Generate a supplementary list of all path names for multiply-linked files.
−p *prefix*	The specified *prefix* will be added to each generated path name. The default is **.** (dot).
−s	Print the file size, in bytes, after each path name.
−u	Print the owner's login name after each path name.
−a *n*	Select if the i-node has been accessed in *n* days.
−m *n*	Select if the i-node has been modified in *n* days.

−c *n*	Select if the i-node has been changed in *n* days.
−n *file*	Select if the i-node has been modified more recently than the argument *file*.
−i *i-node-list*	Generate names for only those i-nodes specified in *i-node-list*.

SEE ALSO

find(1), ncheck(1M).

BUGS

If the **−l** option is not specified, only a single path name out of all possible ones is generated for a multiply-linked i-node. If **−l** is specified, all possible names for every linked file on the file system are included in the output. However, no selection criteria apply to the names generated.

The number of links *ff* is able to handle is limited to maximum 20% of the i−nodes on the disk. This limitation depends also on the available memory.

NAME

fgrep – search a file for a character string

SYNOPSIS

fgrep [options] string [file ...]

DESCRIPTION

fgrep (fast *grep*) seaches files for a character string and prints all lines that contain that string. *fgrep* is different from *grep(1)* and *egrep(1)* because it searches for a string, instead of searching for a pattern that matches an expression. It uses a fast and compact algorithm.

The characters **$**, *****, **[**, **^**, **|**, **(**, **)**, and \ are interpreted literally by *fgrep*, that is, *fgrep* does not recognize full regular expressions as does *egrep*. Since these characters have special meaning to the shell, it is safest to enclose the entire *string* in single quotes '...'.

If no files are specified, *fgrep* assumes standard input. Normally, each line found is copied to the standard output. The file name is printed before each line found if there is more than one input file.

Command line options are:

−b	Precede each line by the block number on which it was found. This can be useful in locating block numbers by context (first block is 0).
−c	Print only a count of the lines that contain the pattern.
−i	Ignore upper/lower case distinction during comparisons.
−l	Print the names of files with matching lines once, separated by new-lines. Does not repeat the names of files when the pattern is found more than once.
−n	Precede each line by its line number in the file (first line is 1).
−v	Print all lines except those that contain the pattern.

−x Print only lines matched entirely.

−e *special_string*
Search for a *special string* (*string* begins with a −).

−f *file* Take the list of *strings* from *file*.

SEE ALSO

ed(1), egrep(1), grep(1), sed(1), sh(1).

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files (even if matches were found).

BUGS

Ideally there should be only one *grep* command, but there is not a single algorithm that spans a wide enough range of space-time tradeoffs. Lines are limited to BUFSIZ characters; longer lines are truncated. BUFSIZ is defined in **/usr/include/stdio.h**.

NAME

file – determine file type

SYNOPSIS

file [**−c**] [**−f** ffile] [**−m** mfile] arg ...

DESCRIPTION

file performs a series of tests on each argument in an attempt to classify it. If an argument appears to be ASCII, *file* examines the first 512 bytes and tries to guess its language. If an argument is an executable **a.out**, *file* will print the version stamp, provided it is greater than 0.

−c The **−c** option causes *file* to check the magic file for format errors. This validation is not normally carried out for reasons of efficiency. No file typing is done under **−c**.

−f If the **−f** option is given, the next argument is taken to be a file containing the names of the files to be examined.

−m The **−m** option instructs *file* to use an alternate magic file.

file uses the file **/etc/magic** to identify files that have some sort of *magic number*, that is, any file containing a numeric or string constant that indicates its type. Commentary at the beginning of **/etc/magic** explains its format.

FILES

/etc/magic

SEE ALSO

filehdr(4).

This page is intentionally left blank

NAME

finc − fast incremental backup

SYNOPSIS

/etc/finc [selection-criteria] file-system raw-tape

DESCRIPTION

finc selectively copies the input *file-system* to the output *raw-tape* . The cautious will want to mount the input *file-system* read-only to insure an accurate backup, although acceptable results can be obtained in read-write mode. The tape must be previously labelled by *labelit*. The selection is controlled by the *selection-criteria*, accepting only those inodes/files for whom the conditions are true.

It is recommended that production of a *finc* tape be preceded by the *ff* command, and the output of *ff* be saved as an index of the tape's contents. Files on a *finc* tape may be recovered with the *frec* command.

335

The argument *n* in the *selection-criteria* which follow is used as a decimal integer (optionally signed), where +*n* means more than *n*, −*n* means less than *n*, and *n* means exactly *n*. A day is defined as a 24 hours.

−a *n*	True if the file has been accessed in *n* days.
−m *n*	True if the file has been modified in *n* days.
−c *n*	True if the i-node has been changed in *n* days.
−n *file*	True for any file which has been modified more recently than the argument *file*.

EXAMPLES

To write a tape consisting of all files from file-system **/usr** modified in the last 48 hours:

```
finc  −m  −2  /dev/rdsk/c1d0s2  /dev/rSA/ctape1
```

SEE ALSO

cpio(1), ff(1M), frec(1M), labelit(1M).

This page is intentionally left blank

NAME

find – find files

SYNOPSIS

find path-name-list expression

DESCRIPTION

find recursively descends the directory hierarchy for each path name in the *path-name-list* (that is, one or more path names) seeking files that match a boolean *expression* written in the primaries given below. In the descriptions, the argument *n* is used as a decimal integer where +*n* means more than *n*, −*n* means less than *n* and *n* means exactly *n*. Valid expressions are:

−name *file*	True if *file* matches the current file name. Normal shell argument syntax may be used if escaped (watch out for **[**, **?** and *).
[−perm] *−onum*	True if the file permission flags exactly match the octal number *onum* (see *chmod*(1)). If *onum* is prefixed by a minus sign, only the bits that are set in *onum* are compared with the file permission flags, and the expression evaluates true if they match.
−type *c*	True if the type of the file is *c*, where *c* is **b**, **c**, **d**, **p**, **l**, or **f** for block special file, character special file, directory, fifo (a.k.a named pipe), symbolic link, or plain file respectively.
−links *n*	True if the file has *n* links.
−user *uname*	True if the file belongs to the user *uname*. If *uname* is numeric and does not appear as a login name in the **/etc/passwd** file, it is taken as a user ID.
−group *gname*	True if the file belongs to the group *gname*. If *gname* is numeric and does not appear in the **/etc/group** file, it is taken as a group ID.

−size *n*[**c**]	True if the file is *n* blocks long (512 bytes per block). If *n* is followed by a **c**, the size is in characters.
−atime *n*	True if the file has been accessed in *n* days. The access time of directories in *path-name-list* is changed by *find* itself.
−mtime *n*	True if the file has been modified in *n* days.
−ctime *n*	True if the file has been changed in *n* days.
−exec *cmd*	True if the executed *cmd* returns a zero value as exit status. The end of *cmd* must be punctuated by an escaped semicolon. A command argument {} is replaced by the current path name.
−ok *cmd*	Like **−exec** except that the generated command line is printed with a question mark first, and is executed only if the user responds by typing **y**.
−print	Always true; causes the current path name to be printed.
−cpio *device*	Always true; write the current file on *device* in *cpio* (1) format (5120-byte records).
−newer *file*	True if the current file has been modified more recently than the argument *file*.
−depth	Always true; causes descent of the directory hierarchy to be done so that all entries in a directory are acted on before the directory itself. This can be useful when *find* is used with *cpio*(1) to transfer files that are contained in directories without write permission.
−mount	Always true; restricts the search to the file system containing the directory specified, or if no directory was specified, the current directory.

−local	True if the file physically resides on the local system.
(*expression*)	True if the parenthesized expression is true (parentheses are special to the shell and must be escaped).

The primaries may be combined using the following operators (in order of decreasing precedence):

1) The negation of a primary (**!** is the unary *not* operator).

2) Concatenation of primaries (the *and* operation is implied by the juxtaposition of two primaries).

3) Alternation of primaries (**−o** is the *or* operator).

EXAMPLE

To remove all files named **a.out** or ***.o** that have not been accessed for a week:

```
find / \( −name a.out −o −name '*.o' \) −atime +7
−exec rm {} \;
```

FILES

/etc/passwd, /etc/group

SEE ALSO

chmod(1), cpio(1), sh(1), test(1), stat(2), umask(2), fs(4).

This page is intentionally left blank

NAME

frec – recover files from a backup tape

SYNOPSIS

/etc/frec [**–p** path] [**–f** reqfile]
raw_tape i_number:name **...**

DESCRIPTION

frec recovers files from the specified *raw_tape* backup tape written by *volcopy*(1M) or *finc*(1M), given their *i_numbers*. The data for each recovery request will be written into the file given by *name* .

The **–p** option allows you to specify a default prefixing *path* different from your current working directory. This will be prefixed to any *names* that are not fully qualified, i.e. that do not begin with **/** or **./**. If any directories are missing in the paths of recovery *names* they will be created.

–p *path*	Specifies a prefixing *path* to be used to fully qualify any names that do not start with **/** or **./**.
–f *reqfile*	Specifies a file which contains recovery requests. The format is i_number:newname, one per line.

EXAMPLES

To recover a file, i-number 1216 when backed-up, into a file named *junk* in your current working directory:

```
frec  /dev/rSA/ctape1  1216:junk
```

To recover files with i_numbers 14156, 1232, and 3141 into files **/usr/src/cmd/a**, **/usr/src/cmd/b** and **/usr/joe/a.c**:

```
frec   –p   /usr/src/cmd   /dev/rSA/ctape1   14156:a
 1232:b
3141:/usr/joe/a.c
```

SEE ALSO

cpio(1), ff(1M), finc(1M), labelit(1M).

BUGS

While paving a path (i.e. creating the intermediate directories contained in a pathname) *frec* can only recover inode fields for those directories contained on the tape and requested for recovery.

NAME

fsck, dfsck – check and repair file systems

SYNOPSIS

/etc/fsck [**–y**] [**–n**] [**–s**X] [**–S**X] [**–t** file] [**–q**] [**–D**] [**–f**] [**–b**] [file-systems]

/etc/dfsck [options1] filsys1 **...** – [options2] filsys2 **...**

DESCRIPTION

fsck

fsck audits and interactively repairs inconsistent conditions for file systemer. If the file system is found to be consistent, the number of files, blocks used, and blocks free are reported. If the file system is inconsistent the user is prompted for concurrence before each correction is attempted. It should be noted that most corrective actions will result in some loss of data. The amount and severity of data loss may be determined from the diagnostic output. The default action for each correction is to wait for the user to respond **yes** or **no**. If the user does not have write permission *fsck* defaults to a **–n** action.

The following options are accepted by *fsck*.

–y Assume a **yes** response to all questions asked by *fsck*.

–n Assume a **no** response to all questions asked by *fsck*; do not open the file system for writing.

–s*X* Ignore the actual free list and (unconditionally) reconstruct a new one by rewriting the super-block of the file system. The file system should be unmounted while this is done; if this is not possible, care should be taken that the system is quiescent and that it is rebooted immediately afterwards. This precaution is necessary so that the old, bad, in-core copy of the superblock will not continue to be used, or written on the file system.

The **–s***X* option allows for creating an optimal free-list organization.

If *X* is not given, the values used when the file system was created are used. The format of X is *cylinder size:gap size*.

−S*X*

Conditionally reconstruct the free list. This option is like **−s***X* above, except that the free list is rebuilt only if there were no discrepancies discovered in the file system. Using **−S** forces a **no** response to all questions asked by *fsck*. This option is useful for forcing free list reorganization on uncontaminated file systems.

−t If *fsck* cannot obtain enough memory to keep its tables, it uses a scratch file. If the **−t** option is specified, the file named in the next argument is used as the scratch file, if needed. Without the **−t** flag, *fsck* will prompts the user for the name of the scratch file. The file chosen should not be on the file system being checked, and if it is not a special file or did not already exist, it is removed when *fsck* completes.

−q Quiet *fsck*. Do not print size-check messages. Unreferenced **fifos** are silently removed. If *fsck* requires it, counts in the superblock will be automatically fixed and the free list salvaged.

−D Directories are checked for bad blocks. Useful after system crashes.

−f Fast check. Check block and sizes and check the free list. The free list is reconstructed if it is necessary.

−b Reboot. If the file system being checked is the root file system and modifications have been made, then either remount the root file system or reboot the system. A remount is done if there was minor damage.

If no *file-systems* are specified, *fsck* will read a list of default file systems from the file **/etc/checklist**.

Inconsistencies checked are as follows:

1. Blocks claimed by more than one i-node or the free list.
2. Blocks claimed by an i-node or the free list outside the range of the file system.
3. Incorrect link counts.
4. Size checks:
   - Incorrect number of blocks.
   - Directory size not 16-byte aligned.
5. Bad i-node format.
6. Blocks not accounted for anywhere.
7. Directory checks:
   - File pointing to unallocated i-node.
   - I-node number out of range.
8. Super Block checks:
   - More than 65536 i-nodes.
   - More blocks for i-nodes than there are in the file system.
9. Bad free block list format.
10. Total free block and/or free i-node count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with the user's concurrence, reconnected by placing them in the **lost+found** directory, if the files are nonempty. The user will be notified if the file or directory is empty or not. Empty files or directories are removed, as long as the **−n** is not specified. *fsck* will force the reconnection of nonempty directories. The name assigned is the i-node number. The only restriction is that the directory **lost+found** must preexist in the root of the file system being checked and must have empty slots in which entries can be made. This is accomplished by making **lost+found**, copying a number of files to the directory, and then removing them (before *fsck* is executed).

Checking the raw device is almost always faster and should be used with everything but the *root* file system.

dfsck

dfsck allows two file system checks on two different drives simultaneously. *Options1* and *options2* are used to pass options to *fsck* for the two sets of file systems. A – is the separator between the file system groups.

The *dfsck* program permits an operator to interact with two *fsck*(1M) programs at once. To aid in this, *dfsck* will print the file system name for each message to the user. When answering a question from *dfsck*, the user must prefix the response with a **1** or a **2** (indicating that the answer refers to the first or second file system group).

FILES

/etc/checklist	contains default list of file systems to check.

SEE ALSO

checkfsys(1M), crash(1M), mkfs(1M), ncheck(1M), uadmin(2), checklist(4), fs(4).

BUGS

I-node numbers for **.** and **..** in each directory are not checked for for validity.

NAME

fsdb – file system debugger

SYNOPSIS

/etc/fsdb special [**–**]

DESCRIPTION

fsdb can be used to patch up a damaged file system after a crash. It has conversions to translate block and i-numbers into their corresponding disk addresses. Also included are mnemonic offsets to access different parts of an i-node. These greatly simplify the process of correcting control block entries or descending the file system tree.

fsdb contains several error-checking routines to verify i-node and block addresses. These can be disabled if necessary by invoking *fsdb* with the optional – argument or by the use of the **O** symbol. (*fsdb* reads the i-size and f-size entries from the superblock of the file system as the basis for these checks.)

Numbers are considered decimal by default. Octal numbers must be prefixed with a zero. During any assignment operation, numbers are checked for a possible truncation error due to a size mismatch between source and destination.

fsdb reads a block at a time and will therefore work with raw as well as block I/O. A buffer management routine is used to retain commonly used blocks of data in order to reduce the number of read system calls. All assignment operations result in an immediate write-through of the corresponding block.

The symbols recognized by *fsdb* are:

#	absolute address
i	convert from i-number to i-node address
b	convert to block address
d	directory slot offset
+,–	address arithmetic
q	quit

347

>,<	save, restore an address
=	numerical assignment
=+	incremental assignment
=−	decremental assignment
="	character string assignment
O	error checking flip flop
p	general print facilities
f	file print facility
B	byte mode
W	word mode
D	double word mode
!	escape to shell

The print facilities generate a formatted output in various styles. The current address is normalized to an appropriate boundary before printing begins. It advances with the printing and is left at the address of the last item printed. The output can be terminated at any time by typing the delete character. If a number follows the **p** symbol, that many entries are printed. A check is made to detect block boundary overflows since logically sequential blocks are generally not physically sequential. If a count of zero is used, all entries to the end of the current block are printed. The print options available are:

i	print as i-nodes
d	print as directories
o	print as octal words
e	print as decimal words
c	print as characters
b	print as octal bytes

The **f** symbol is used to print data blocks associated with the current i-node. If followed by a number, that block of the file is printed. (Blocks are numbered from zero.) The desired print option letter follows the block number, if present, or the **f** symbol. This print facility works for small as well as large files. It checks for special devices and that the block pointers used to find the data are not zero.

Dots, tabs, and spaces may be used as function delimiters but are not necessary. A line with just a new-line character will increment the current address by the size of the data type last printed. That is, the address is set to the next byte, word, double word, directory entry or i-node, allowing the user to step through a region of a file system. Information is printed in a format appropriate to the data type. Bytes, words and double words are displayed with the octal address followed by the value in octal and decimal. A **.B** or **.D** is appended to the address for byte and double word values, respectively. Directories are printed as a directory slot offset followed by the decimal i-number and the character representation of the entry name. I-nodes are printed with labeled fields describing each element.

The following mnemonics are used for i-node examination and refer to the current working i-node:

md	mode
ln	link count
uid	user ID number
gid	group ID number
sz	file size
a#	data block numbers (0 – 12)
at	access time
mt	modification time
maj	major device number
min	minor device number

EXAMPLES

386i	prints i-number 386 in an i-node format. This now becomes the current working i-node.
ln=4	changes the link count for the working i-node to 4.
ln= +1	increments the link count by 1.
fc	prints, in ASCII, block zero of the file associated with the working i-node.

2i.fd	prints the first 32 directory entries for the root i-node of this file system.
d5i.fc	changes the current i-node to that associated with the 5th directory entry (numbered from zero) found from the above command. The first logical block of the file is then printed in ASCII.
512B.p0o	prints the superblock of this file system in octal.
2i.a0b.d7=3	changes the i-number for the seventh directory slot in the root directory to 3. This example also shows how several operations can be combined on one command line.
d7.nm="name"	changes the name field in the directory slot to the given string. Quotes are optional when used with **nm** if the first character is alphabetic.
a2b.p0d	prints the third block of the current i-node as directory entries.

SEE ALSO

fsck(1M), dir(4), fs(4).

NAME

fsstat – report file system status

SYNOPSIS

/etc/fsstat special_file

DESCRIPTION

fsstat reports on the status of the file system on *special_file*. During startup, this command is used to determine if the file system needs checking before it is mounted. *fsstat* succeeds if the file system is unmounted and appears okay. For the root file system, it succeeds if the file system is active and not marked bad.

SEE ALSO

fs(4).

DIAGNOSTICS

The command has the following exit codes:

0: the file system is not mounted and appears okay, (except for root where 0 means mounted and okay).

1: the file system is not mounted and needs to be checked.

2: the file system is mounted.

3: the command failed.

This page is intentionally left blank

NAME

fuser – identify processes using a file or file structure

SYNOPSIS

/etc/fuser [**–ku**] files | resources [–] [[**–ku**] files | resources]

DESCRIPTION

fuser outputs the process IDs of the processes that are using the *files* or remote *resources* specified as arguments. Each process ID is followed by a letter code, interpreted as follows: if the process is using the file as 1) its current directory, the code is **c**, 2) the parent of its current directory (only when the file is being used by the system), the code is **p**, or 3) its root directory, the code is **r**. For block special devices with mounted file systems, all processes using any file on that device are listed. For remote resource names, all processes using any file associated with that remote resource (Remote File Sharing) are reported. (**fuser** cannot use the mount point of the remote resource; it must use the resource name.) For all other types of files (text files, executables, directories, devices, etc.) only the processes using that file are reported.

The following options may be used with *fuser*:

–u the user login name, in parentheses, also follows the process ID.

–k the SIGKILL signal is sent to each process. Since this option spawns kills for each process, the kill messages may not show up immediately [see *kill*(2)].

If more than one group of files are specified, the options may be respecified for each additional group of files. A lone dash cancels the options currently in force; then, the new set of options applies to the next group of files.

The process IDs are printed as a single line on the standard output, separated by spaces and terminated with a single new line. All other output is written on standard error.

You cannot list processes using a particular file from a remote resource mounted on your machine. You can only use the resource name as an argument.

Any user with permission to read **/dev/kmem** and **/dev/mem** can use **fuser**. Only the super-user can terminate another user's process

FILES

/dev/kmem	for system image
/dev/mem	also for system image

SEE ALSO

mount(1M), ps(1), kill(2), signal(2).

NAME

gendev – generate device numbers.

SYNOPSIS

/etc/gendev [*parameter list*]

DESCRIPTION

gendev is used to generate the major/minor device number pair for *mknod* (1M). It can operate interactive by calling *gendev* without any parameter. When operating interactive *gendev* asks the user questions about device type (terminal, printer, disk or kmem), CPU number, channel number or what ever information is needed. Finally, *gendev* outputs the device number in hexadecimal form, plus the major and minor device numbers in decimal form. By adding a parameter list to the command call *gendev* will not asks any questions. *gendev* can be used to translate major/minor numbers or devices to text explanations.

355

EXAMPLE

Example of interactive operating :

```
$ /etc/gendev
Select: 1:term 2:print 3:disk 4:kmem 5:maj/min to meaning 6:dev to meaning

Select: 3
Dioc no: 14
Channel no: 8
Subdiskno (0 for none): 0
Device: 0x3880    major=56  minor=128
$
```

Example of operating on parameter list:

```
$ /etc/gendev 3 14 8 0
3880 56 128
$
```

Example of explaining major/minor number:

```
$ /etc/gendev
Select: 1:term 2:print 3:disk 4:kmem 5:maj/min to meaning 6:dev to meaning

Select: 3
major: 56
minor: 128
Disk on dioc no 14 disk no 8 Subdisk no 0   first hard disk on controller 1
$
```

BUGS

None of the input parameters to *gendev* is tested for valid value.

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, below.

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 one might process the arguments for a command that can take the options **a** or **b**, as well as the option **o**, which requires an argument:

```
set -- getopt abo: $*
if [ $? != 0 ]
then
        echo $USAGE
        exit 2
fi
for i in $*
do
        case $i in
        -a | -b)   FLAG=$i; shift;;
```

```
                    −o)           OARG=$2; shift 2;;
                    −−)           shift; break;;
                    esac
            done
```

This code 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 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 Rules 3-10. It should be used in place of the *getopt*(1) command. (See the **WARNING**, below.)

optstring must contain the option letters the command using *getopts* will recognize; if a letter is followed by a colon, the option is expected to have an argument, or group of arguments, which must be separated from it by white space.

Each time it is invoked, *getopts* will place the next option in the shell variable *name* and the index of the next argument to be processed in the shell variable **OPTIND**. Whenever the shell or a shell procedure is invoked, **OPTIND** is initialized to **1**.

When an option requires an option-argument, *getopts* places it in the shell variable **OPTARG**.

If an illegal option is encountered, **?** will be placed in *name*.

When the end of options is encountered, *getopts* exits with a non-zero exit status. The special option "– –" may be used to delimit the end of the options.

By default, *getopts* parses the positional parameters. If extra arguments (*arg* ...) are given on the *getopts* command line, *getopts* will parse them instead.

/usr/lib/getoptcvt reads the shell script in *file*, converts it to use *getopts*(1) instead of *getopt*(1), and writes the results on the standard output.

−b the results of running */usr/lib/getoptcvt* will be portable to earlier releases of the 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, they should use *getopts*(1) or *getopt*(3C) to parse positional parameters and check for options that are legal for that command (see **WARNINGS**, below).

EXAMPLE

The following fragment of a shell program shows how one might process the arguments for a command that can take the options **a** or **b**, as well as the option **o**, which requires an option-argument:

```
while 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).

SUPERMAX System V Release Notes, Essential Utilities

WARNING

Although the following command syntax rule relaxations are permitted under the current implementation, they should not be used because they may not be supported in future releases of the system. As in the **EXAMPLE** section above, **a** and **b** are options, and the option **o** requires an option-argument:

```
cmd -aboxxx file
```

(Rule 5 violation:
options with option-arguments must not be grouped with other options)

```
cmd -ab -oxxx file
```

(Rule 6 violation:
there must be white space after an option that takes an option-argument)

Changing the value of the shell variable **OPTIND** or parsing different sets of arguments may lead to unexpected results.

DIAGNOSTICS

getopts prints an error message on the standard error when it encounters an option letter not included in *optstring*.

This page is intentionally left blank

NAME

getty – set terminal type, modes, speed, and line discipline

SYNOPSIS

/etc/getty [**–u**] [**–i**] [**–T** terminologytable] [**–v**] [**–n**] [**–b**] [**–r** [delaytime]] [**–t** timeout] line [speed [type [linedisc]]]

uugetty [**–T** terminology] [**–v**] [**–h**] [**–b**] [**–r** [delaytime]] [**–t** timeout] line [speed [type [linedisc]]]

/etc/getty **–c** file

DESCRIPTION

getty is a program that is invoked by *init*(1M). It is the second process in the series, (*init-getty-login-shell*) that ultimately connects a user with the UNIX system. It can only be executed by the super-user; that is, a process with the user-ID of **root**. Initially *getty* runs the terminology to initialize the VTI (Virtuel Terminal Interface), and prints the login message field for the entry it is using from **/etc/gettydefs**. *getty* reads the user's login name and invokes the *login*(1) command with the user's name as argument. While reading the name, *getty* attempts to adapt the system to the speed and type of terminal being used.

363

It does this by using the options and arguments specified.

–u Act like *uugetty*. If this option is set, *getty* makes a lock file like *uugetty* does (see uugetty). This feature is very useful when a modem is connected to the *line*. The lock file makes it possible for several programs to operate on the same modem (see **TTY** and **uucp**).

−i	No issue picture. If this option is set *getty* will not print out the issue file */etc/issue* to the standard output.
−T *terminologytable*	Run terminology table. By setting **−T** *getty* will run the terminology table /etc/types/*terminologytable*. *The directory name /etc/types* is added by *getty*. This option overrun the **−v** option.
−v	No terminology run. By setting **−v** *getty* will not run the terminology table specified in the **NTC.** (Network Terminal Controller) if any is connected to the *line*.
−h	No hangup. Unless *getty* is invoked with the **−h** flag, *getty* will force a hangup on the line by setting the speed to zero before setting the speed to the default or specified speed.
−b	Ignore <*break*> character. Option **−b** disables the break function which make *getty* try next entry given in the current selected */etc/gettydefs* entry.
−r [*delaytime*]	Read wait. Option **−r** will make *getty* wait for a character before writing the login message placed in the */etc/gettydefs* file. If a *delaytime* is added, *getty* will wait for the specified *delaytime* (in seconds) after first character has been detected, before writing the login message.
−t *timeout*	Set timeout. The **−t** flag plus *timeout* (in seconds), specifies that *getty* should exit if the open on the line succeeds and no one types

anything in the specified number of seconds.

If the options **−T** and **−v** are not specified, *getty* examine the **NTC** (Network Terminal Controller), if any. The TYPE field in the NTC is used as an ID to select the terminology table, required by the terminal connected to the **NTC** terminal interface. *getty* will first check the map file **/etc/termtype.map**. An entry to this map file may look somewhat like,

```
jr int/dde450.t
```

If no matching to the ID is found in the map file, *getty* will continue to scan the two directories, **/etc/types/ntc** and **/etc/types**. The **/etc/types/ntc** directory is useful for placing personal and general links to the terminology tables especially relayed to run terminology from *getty*. The directory scanning takes place in the order of which the entries appears as shown by the following command,

```
$ ls −f /etc/types/ntc /etc/types
```

The matching routine looks for the first short match. If the **/etc/types/ntc** looks like,

```
.
..
jr.i
jr400.t
jr400.i
jr.t
jr
```

the matching routine will select the entry **jr400.t** to be the first short matching to **jr**.. Notice that entries having a suffix like "**.?**", where "**?**" means any character, are ignored, except the suffix "**.t**". To select **jr.t** change **jr** to **jr.t** in the **NTC** TYPE field, or switch **jr400.t** and **jr.t** in the **/etc/types/ntc** directory. If still no match *getty* will expand all "**.** to "**/dde** except the two last positions of the **NTC** TYPE name.

365

The expansion of **uk.500.t** becomes **uk.dde500.t** making *getty* perform an extra scanning in the **/etc/types/uk** directory to match **dde500.t**.

The first argument, *line*, is the name of a tty-line in **/dev** to which *getty* is to attach itself. *getty* uses this string as the name of a file in the **/dev** directory to open for reading and writing.

speed, the optional second argument, is a label to a speed and tty definition in the file **/etc/gettydefs**. This definition tells *getty* at what speed to initially run; what the login message should look like; what the initial tty settings are and what speed to try next. The user should indicate if the speed is inappropriate (by typing a <*break*> character). The default *speed* is 300 baud.

Type, the optional third argument, is a character string describing to *getty* what type of terminal is connected to the line in question. *getty* recognizes the following types:

none	default
ds40-1	Dataspeed40/1
tektronix,tek	Tektronix
vt61	DEC vt61
vt100	DEC vt100
hp45	Hewlett-Packard 45
c100	Concept 100

The default terminal is **none**; i.e., any crt or normal terminal unknown to the system. Also, for terminal type to have any meaning, the virtual terminal handlers must be compiled into the operating system. They are available, but not compiled in the default condition.

linedisc, the optional fourth argument, is a character string describing which line discipline to use in communicating with the terminal. The line disciplines available in the operating system is **LDISC0** and **LDISC1**. The default line discipline is **LDISCO**, (see User's Guide 2-6).

When given no optional arguments, *getty* sets the *speed* of the interface to 300 baud, specifies that raw mode is to be used (awaken on every character); that echo is to be suppressed, either parity allowed; new-line characters will be converted to carriage return-line feed and tab expansion performed on the standard output. It types the login message before reading the user's name one character at a time. If a null character (or framing error) is received, it is assumed to be the result of the user activating the "break" key. This will cause *getty* to attempt the next *speed* in the series. The series that *getty* tries is determined by what it finds in **/etc/gettydefs**.

After the user's name has been typed in, it is terminated by a new-line or carriage-return character. The latter results in the system being set to treat carriage returns appropriately (see *ioctl*(2)).

The user's name is scanned to see if it contains any lower-case alphabetic characters; if not, and if the name is non-empty, the system is told to map any future upper-case characters into the corresponding lower-case characters.

Finally, *login* is **exec**'d with the user's name as an argument. Additional arguments may be typed after the login name. These are passed to *login*, which will place them in the environment (see *login*(1)).

If *getty* is activated by the link name *uugetty* the the options **−u** and **−i** will be set, which make *getty* able to behave like the *uugetty* supplied by the **uucp** package.

A check option is provided. When *getty* is invoked with the **−c** option and *file*, it scans the file as if it were scanning **/etc/gettydefs** and prints out the results to the standard output. If there are any unrecognized modes or improperly constructed entries, it reports these. If the entries are correct, it prints out the values of the various flags. See *ioctl*(2) to interpret the values. Note that some values are added to the flags automatically.

FILES

/etc/gettydefs	line setup file
/etc/termtype.map	terminology map file
/etc/types/ntc	terminology file links directory
/etc/issue	issue picture file

SEE ALSO

ct(1C), init(1M), login(1), ioctl(2), gettydefs(4), inittab(4), termtype.map(4), issue(4), tty(7).

BUGS

While *getty* understands simple, single character quoting conventions, it is not possible to quote certain special control characters used by *getty*. Thus, you cannot login via *getty* and type a **#**, **@**, **/**, **!**, **_**, backspace, **^U**, **^D**, or **&** as part of your login name or arguments. *getty* uses them to determine when the end of the line has been reached, which protocol is being used, and what the erase character is. They will always be interpreted as having their special meaning.

NAME

grep – search a file for a pattern

SYNOPSIS

grep [options] limited regular expression [file ...]

DESCRIPTION

grep searches files for a pattern and prints all lines that contain that pattern. *grep* uses limited regular expressions (expressions that have string values that use a subset of the possible alphanumeric and special characters) like those used with *ed (1)* to match the patterns. It uses a compact non-deterministic algorithm.

Be careful using the characters **$**, *****, **[**, **^**, **|**, **(**, **)**, and \ in the *limited regular expression* because they are also meaningful to the shell. It is safest to enclose the entire *limited regular expression* in single quotes '...'.

If no files are specified, *grep* assumes standard input. Normally, each line found is copied to standard output. The file name is printed before each line found if there is more than one input file.

Command line options are:

- **–b** Precede each line by the block number on which it was found. This can be useful in locating block numbers by context (first block is 0).
- **–c** Print only a count of the lines that contain the pattern.
- **–i** Ignore upper/lower case distinction during comparisons.
- **–l** Print the names of files with matching lines once, separated by new-lines. Does not repeat the names of files when the pattern is found more than once.
- **–n** Precede each line by its line number in the file (first line is 1).
- **–s** Suppress error messages about nonexistent or unreadable files
- **–v** Print all lines except those that contain the pattern.

369

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 nulls, *grep* will only match up to the first null; if it matches, it will print the entire line.

NAME

hwdate − set or get the date from an external clock device.

SYNOPSIS

hwdate [mmddhhmmyyss]

DESCRIPTION

If no argument is given, the current date and time from an external clock device are printed. Otherwise, the clock device will be initalized with the date from the argument. 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; *ss* is the seconds. For example:

```
hwdate 100800458517
```

sets the date to Oct 8, 12:45:17 AM, 1985. The system operates in GMT. *hwdate* takes care of the conversion to and from local standard and daylight time.

EXAMPLE

hwdate is useful for setting the current date and time when booting the system. The following example shows a shell script, that should be called by **init** during boot.

```
d='hwdate 2>/dev/null'  # call hwdate and save date
if [ $? != 0 ]          # if no clock device available
then
     /etc/settime       # then get date from console
     while [ $? != 0 ]  # continue until valid date
     do
          /etc/settime
     done
else
     echo Current date is: 'date $d'
                        # if clock device, set date
fi
```

DIAGNOSTICS

No permission	if your are not the super-user and you try to change the date.
bad conversion	if the date set is syntactically incorrect.

NAME

hwstatus – decode status from Non-Operator Diagnostic Programs

SYNOPSIS

/etc/hwstatus [−h hex hex hex hex] [−d special file]

DESCRIPTION

hwstatus decodes status information on the winchester disk written by the Diagnostic Programs.

hwstatus will by default use the special file */dev/boot.0*, a reference to the physical disk on which winchester boot is installed.

−h hex hex hex hex
causes *hwstatus* to translate the 4 32-bit hexadecimal values into status information.

−d <special file>
causes *hwstatus* to use *<special file>* instead of /dev/boot.0 as a reference to the physical disk on which winchester boot is installed.

The termination code from *hwstatus*

0	No errors found
1	Illegal parameters or error in opening */dev/boot.0* or *<special file>*
2	Error during one or more of the tests

SEE ALSO

boot(1M), mkwboot(1M).

This page is intentionally left blank

NAME
id – print user and group IDs and names

SYNOPSIS
id

DESCRIPTION
id outputs the user and group IDs and the corresponding names of the invoking process. If the effective and real IDs are different, both are printed.

SEE ALSO
logname(1), getuid(2).

This page is intentionally left blank

NAME

infocmp – compare or print out terminfo descriptions

SYNOPSIS

infocmp [**–d**] [**–c**] [**–n**] [**–I**] [**–L**] [**–C**] [**–r**] [**–u**] [**–s d**|**i**|**l**|**c**] [**–v**] [**–V**] [**–1**] [**–w** width] [**–B** directory] [termname ...]

DESCRIPTION

infocmp can be used to compare a binary *terminfo*(4) entry with other terminfo entries, rewrite a *terminfo*(4) description to take advantage of the **use=** terminfo field, or print out a *terminfo*(4) description from the binary file (*term*(4)) in a variety of formats. In all cases, the boolean fields will be printed first, followed by the numeric fields, followed by the string fields.

Default Options

If no options are specified and zero or one *termnames* are specified, the **–I** option will be assumed. If more than one *termname* is specified, the **–d** option will be assumed.

Comparison Options [–d] [–c] [–n]

infocmp compares the *terminfo*(4) description of the first terminal *termname* with each of the descriptions given by the entries for the other terminal's *termnames*. If a capability is defined for only one of the terminals, the value returned will depend on the type of the capability: **F** for boolean variables, **–1** for integer variables, and **NULL** for string variables.

–d produce a list of each capability that is different. In this manner, if one has two entries for the same terminal or similar terminals, using *infocmp* will show what is different between the two entries. This is sometimes necessary when more than one person produces an entry for the same terminal and one wants to see what is different between the two.

–c produce a list of each capability that is common between the two entries. Capabilities that are not set are ignored. This option can be used as a quick check to see if the **−u** option is worth using.

−n produce a list of each capability that is in neither entry. If no *termnames* are given, the environment variable **TERM** will be used for both of the *termnames*. This can be used as a quick check to see if anything was left out of the description.

Source Listing Options [−I] [−L] [−C] [−r]

The **−I**, **−L**, and **−C** options will produce a source listing for each terminal named.

−I	use the *terminfo*(4) names
−L	use the long C variable name listed in <**term.h**>
−C	use the *termcap* names
−r	when using **−C**, put out all capabilities in *termcap* form

If no *termnames* are given, the environment variable **TERM** will be used for the terminal name.

The source produced by the **−C** option may be used directly as a *termcap* entry, but not all of the parameterized strings may be changed to the *termcap* format. *infocmp* will attempt to convert most of the parameterized information, but that which it doesn't will be plainly marked in the output and commented out. These should be edited by hand.

All padding information for strings will be collected together and placed at the beginning of the string where *termcap* expects it. Mandatory padding (padding information with a trailing '/') will become optional.

All *termcap* variables no longer supported by *terminfo*(4), but which are derivable from other *terminfo*(4) variables, will be output. Not all *terminfo*(4) capabilities will be translated; only those variables which were part of *termcap* will normally be output. Specifying the **−r** option will take off this restriction, allowing all capabilities to be output in *termcap* form.

Note that because padding is collected to the beginning of the capability, not all capabilities are output, mandatory padding is not supported, and *termcap* strings were not as flexible, it is not always possible to convert a *terminfo*(4) string capability into an equivalent *termcap* format. Not all of these strings will be able to be converted. A subsequent conversion of the *termcap* file back into *terminfo*(4) format will not necessarily reproduce the original *terminfo*(4) source.

Some common *terminfo* parameter sequences, their *termcap* equivalents, and some terminal types which commonly have such sequences, are:

Terminfo	Termcap	Representative Terminals
%p1%c	%.	adm
%p1%d	%d	hp, ANSI standard, vt100
%p1%'x'%+%c	%+x	concept
%i	%i	ANSI standard, vt100
%p1%?%'x'%>%t%p1%'y'%+%;	%>xy	concept
%p2 is printed before %p1	%r	hp

Use= Option [−u]

−u produce a *terminfo*(4) source description of the first terminal *termname* which is relative to the sum of the descriptions given by the entries for the other terminals *termnames*. It does this by analyzing the differences between the first *termname* and the other *termnames* and producing a description with **use=** fields for the other terminals. In this manner, it is possible to retrofit generic terminfo entries into a terminal's description. Or, if two similar terminals exist, but were coded at different times or by different people so that each description is a full description, using *infocmp* will show what can be

379

done to change one description to be relative to the other.

A capability will get printed with an at-sign (@) if it no longer exists in the first *termname*, but one of the other *termname* entries contains a value for it. A capability's value gets printed if the value in the first *termname* is not found in any of the other *termname* entries, or if the first of the other *termname* entries that has this capability gives a different value for the capability than that in the first *termname*.

The order of the other *termname* entries is significant. Since the terminfo compiler **tic**(1M) does a left-to-right scan of the capabilities, specifying two **use=** entries that contain differing entries for the same capabilities will produce different results depending on the order that the entries are given in. *infocmp* will flag any such inconsistencies between the other *termname* entries as they are found.

Alternatively, specifying a capability *after* a **use=** entry that contains that capability will cause the second specification to be ignored. Using *infocmp* to recreate a description can be a useful check to make sure that everything was specified correctly in the original source description.

Another error that does not cause incorrect compiled files, but will slow down the compilation time, is specifying extra **use=** fields that are superfluous. *infocmp* will flag any other *termname* **use=** fields that were not needed.

Other Options [−s d|i|l|c] [−v] [−V] [−1] [−w width]

−s sort the fields within each type according to the argument below:

- **d** leave fields in the order that they are stored in the *terminfo* database.
- **i** sort by *terminfo* name.
- **l** sort by the long C variable name.
- **c** sort by the *termcap* name.

If no **−s** option is given, the fields printed out will be sorted alphabetically by the *terminfo* name within each type, except in the case of the **−C** or the **−L** options, which cause the sorting to be done by the *termcap* name or the long C variable name, respectively.

−v print out tracing information on standard error as the program runs.

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

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

−w change the output to width characters.

Changing Databases [−A directory] [−B directory]

The location of the compiled *terminfo*(4) database is taken from the environment variable **TERMINFO**. If the variable is not defined, or the terminal is not found in that location, the system *terminfo*(4) database, usually in */usr/lib/terminfo*, will be used. The options **−A** and **−B** may be used to override this location. The **−A** option will set **TERMINFO** for the first *termname* and the **−B** option will set **TERMINFO** for the other *termnames*. With this, it is possible to compare descriptions for a terminal with the same name located in two different databases. This is useful for comparing descriptions for the same terminal created by different people. Otherwise the terminals would have to be named differently in the *terminfo*(4) database for a comparison to be made.

FILES

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

DIAGNOSTICS

malloc is out of space!

There was not enough memory available to process all the terminal descriptions requested. Run *infocmp* several times, each time including a subset of the desired *termnames*.

use= order dependency found:

A value specified in one relative terminal specification was different from that in another relative terminal specification.

'use=*term*' did not add anything to the description.

A relative terminal name did not contribute anything to the final description.

must have at least two terminal names for a comparison to be done.

The **−u**, **−d** and **−c** options require at least two terminal names.

SEE ALSO

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

Chapter 10 of the *Programmer's Guide*.

NOTE

The *termcap* database (from earlier releases of UNIX System V) may not be supplied in future releases.

NAME

init, telinit – process control initialization

SYNOPSIS

/etc/init [**0123456SsQqabc**]

/etc/telinit [**0123456SsQqabc**]

DESCRIPTION

Init

init is a general process spawner. Its primary role is to create processes from information stored in the file **/etc/inittab** (see *inittab*(4)).

At any given time, the system is in one of eight possible run levels. A run level is a software configuration of the system under which only a selected group of processes exist. The processes spawned by *init* for each of these run levels is defined in **/etc/inittab**. *init* can be in one of eight run levels, **0 – 6** and **S** or **s** (run levels **S** and **s** are identical). The run level changes when a privileged user runs **/etc/init**. This user-spawned *init* sends appropriate signals to the original *init* spawned by the operating system when the system was booted, telling it which run level to change to.

The following are the arguments to *init*.

0	shut the machine down so it is safe to remove the power. Have the machine remove power if it can.
1	put the system in single-user mode. Unmount all file systems except root. All user processes are killed except those connected to the console.
2	put the system in multi-user mode. All multi-user environment terminal processes and daemons are spawned. This state is commonly referred to as the multi-user state.
3	start the remote file sharing processes and daemons. Mount and advertise remote

resources. Run level **3** extends multi-user mode and is known as the remote-file-sharing state.

4 is available to be defined as an alternative multi-user environment configuration. It is not necessary for system operation and is usually not used.

5 Stop the UNIX system and go to the firmware monitor.

6 Stop the UNIX system and reboot to the state defined by the `initdefault` entry in **/etc/inittab**.

a,b,c process only those **/etc/inittab** entries having the **a**, **b** or **c** run level set. These are pseudo-states, which may be defined to run certain commands, but which do not cause the current run level to change.

Q,q re-examine **/etc/inittab**.

S,s enter single-user mode. When this occurs, the terminal which executed this command becomes the system console. This is the only run level that doesn't require the existence of a properly formatted **/etc/inittab** file. If this file does not exist, then by default the only legal run level that *init* can enter is the single-user mode. When the system enters **S** or **s**, all mounted file systems remain mounted and only processes spawned by init are killed.

When a UNIX system is booted, *init* is invoked and the following occurs. First, *init* looks in **/etc/inittab** for the `initdefault` entry (see **inittab**(4)). If there is one, *init* uses the run level specified in that entry as the initial run level to enter. If there is no `initdefault` entry in **/etc/inittab,** *init* requests that the user enter a run level from the virtual system console. If an **S** or **s** is entered, *init* goes to the single-user state. In the single-user state the virtual console terminal is assigned to the user's terminal and is opened for reading and writing. The command **/bin/su** is invoked and a message is generated on the physical console saying where the virtual console has been relocated. Use either *init* or *telinit*, to signal *init* to change the run level of the system. Note that if the shell is terminated (via an end-of-file), *init* will only re-initialize to the single-user state if the **/etc/inittab** file does not exist.

If a **0** through **6** is entered, *init* enters the corresponding run level. Note that, on the Supermax Computer, the run levels **0**, **1**, **5**, and **6** are reserved states for shutting the system down; the run levels **2**, **3**, and **4** are available as normal operating states.

If this is the first time since power up that *init* has entered a run level other than single-user state, *init* first scans **/etc/inittab** for `boot` and `bootwait` entries (see *inittab*(4)). These entries are performed before any other processing of **/etc/inittab** takes place, providing that the run level entered matches that of the entry. In this way any special initialization of the operating system, such as mounting file systems, can take place before users are allowed onto the system. *init* then scans **/etc/inittab** and executes all other entries that are to be processed for that run level.

In a multi-user environment, **/etc/inittab** is set up so that *init* will create a *getty* process for each terminal that the administrator sets up to respawn.

To spawn each process in **/etc/inittab**, *init* reads each entry and for each entry that should be respawned, it forks a child process. After it has spawned all of the processes specified by **/etc/inittab**, *init* waits for one of its descendant processes to die, a powerfail signal, or a signal from another *init* or *telinit* process to change the system's run level. When one of these conditions occurs, *init* re-examines **/etc/inittab**. New entries can be added to **/etc/inittab** at any time; however, *init* still waits for one of the above three conditions to occur before re-examining **/etc/inittab**. To get around this, **init Q** or **init q** command wakes *init* to re-examine **/etc/inittab** immediately.

When *init* comes up at boot time and whenever the system changes from the single-user state to another run state, init sets the *ioctl*(2) states of the virtual console to those modes saved in the file **/etc/ioctl.syscon**. This file is written by *init* whenever the single-user state is entered.

When a run level change request is made *init* sends the warning signal (**SIGTERM**) to all processes that are undefined in the target run level. *init* waits 5 seconds before forcibly terminating these processes via the kill signal (**SIGKILL**).

The shell running on each terminal will terminate when the user types an end-of-file or hangs up. When *init* receives a signal telling it that a process it spawned has died, it records the fact and the reason it died in **/etc/utmp** and **/etc/wtmp** if it exists (see *who*(1)). A history of the processes spawned is kept in **/etc/wtmp.**

If *init* receives a *powerfail* signal (**SIGPWR**) it scans **/etc/inittab** for special entries of the type *powerfail* and *powerwait*. These entries are invoked (if the run levels permit) before any further processing takes place. In this way *init* can perform various cleanup and recording functions during the powerdown of the operating system. Note that in the single-user states, **S** and **s**, only *powerfail* and *powerwait* entries are executed.

telinit

telinit, which is linked to **/etc/init**, is used to direct the actions of *init*. It takes a one-character argument and signals *init* to take the appropriate action.

FILES

/etc/inittab	script file for 'init'
/etc/utmp	accounting
/etc/wtmp	accounting
/etc/ioctl.syscon	terminal set up file
/dev/console	real system console
/dev/syscon	virtual system console
/dev/systty	physical system console

SEE ALSO

getty(1M), login(1), sh(1), shutdown(1M), stty(1), who(1), kill(2), gettydefs(4), inittab(4), utmp(4), termio(7).

DIAGNOSTICS

If *init* finds that it is respawning an entry from **/etc/inittab** more than 10 times in 2 minutes, it will assume that there is an error in the command string in the entry, and generate an error message on the system console. It will then refuse to respawn this entry until either 5 minutes has elapsed or it receives a signal from a user-spawned *init* (*telinit*). This prevents *init* from eating up system resources when someone makes a typographical error in the *inittab* file or a program is removed that is referenced in **/etc/inittab**.

When attempting to boot the system, failure of *init* to prompt for a new run level may be becaue the virtual system console is linked to a device other than the physical system console.

WARNINGS

init and *telinit* can be run only by someone who is super-user.

The **S** or **s** state must not be used indiscriminately in the **/etc/inittab** file. A good rule to follow when modifying this file is to avoid adding this state to any line other than the **initdefault**.

The change to **/etc/gettydefs** described in the **WARNINGS** section of the *gettydefs*(4) manual page will permit terminals to pass 8 bits to the system as long as the system is in multi-user state (run level greater than 1). When the system changes to single-user state, the *getty* is killed and the terminal attributes are lost. To permit a terminal to pass 8 bits to the system in single-user state, after you are in single-user state, type:

```
stty -istrip cs8
```

NAME

ins_xos – install an Operating System Extension

SYNOPSIS

ins_xos XOS-file

DESCRIPTION

ins_xos is used to load an operating system extension (a so-called XOS) module that has been generated using the *makexos*(1M) program.

An XOS module is a collection of operating system services that supplement the services found in the operating system that is loaded as part of the bootstrap process. Typical elements in the XOS module are the STREAMS mechanism and communication modules, such as TCP/IP.

ins_xos is typically executed as part of the commands found in the */etc/rc.d* directory.

The argument to *ins_xos* is a file containing an XOS module. Typically, such a file has the name:

X–str.########

where ######## is a hexadecimal number identifying the operating system version. *ins_xos* may be invoked either with the complete XOS module file name or simply with the abbreviated name:

X–str.

In the latter case *ins_xos* will itself calculate the version of the operating system and append it to the abbreviated name.

Once an XOS module has been loaded into a system, *ins_xos* will refuse to load another XOS module. If, therefore, one wants to make changes to the XOS module, one must bootstrap the computer before the changes will take effect.

SEE ALSO

makexos(1M).

This page is intentionally left blank

NAME

install − install commands

SYNOPSIS

/etc/install [**−c** dira] [**−f** dirb] [**−i**] [**−n** dirc] [**−m** mode] [**−u** user] [**−g** group] [**−o**] [**−s**] file [dirx ...]

DESCRIPTION

The *install* command is most commonly used in "makefiles" [See *make*(1)] to install a *file* (updated target file) in a specific place within a file system. Each *file* is installed by copying it into the appropriate directory, thereby retaining the mode and owner of the original command. The program prints messages telling the user exactly what files it is replacing or creating and where they are going.

If no options or directories (*dirx* ...) are given, *install* will search a set of default directories (**/bin**, **/usr/bin**, **/etc**, **/lib**, and **/usr/lib**, in that order) for a file with the same name as *file*. When the first occurrence is found, *install* issues a message saying that it is overwriting that file with *file*, and proceeds to do so. If the file is not found, the program states this and exits without further action.

If one or more directories (*dirx* ...) are specified after *file*, those directories will be searched before the directories specified in the default list.

The meanings of the options are:

−c *dira* Installs a new command (*file*) in the directory specified by *dira*, only if it is not found. If it is found, *install* issues a message saying that the file already exists, and exits without overwriting it. May be used alone or with the **−s** option.

−f *dirb* Forces *file* to be installed in given directory, whether or not one already exists. If the file being installed does not already exist, the mode and owner of the new file will be set to

755 and **bin**, respectively. If the file already exists, the mode and owner will be that of the already existing file. May be used alone or with the **−o** or **−s** options.

−i	Ignores default directory list, searching only through the given directories (*dirx* ...). May be used alone or with any other options except **−c** and **−f**.
−n *dirc*	If *file* is not found in any of the searched directories, it is put in the directory specified in *dirc*. The mode and owner of the new file will be set to **755** and **bin**, respectively. May be used alone or with any other options except **−c** and **−f**.
−m *mode*	The mode of the new file is set to *mode*. Only available to the superuser.
−u *user*	The owner of the new file is set to *user*. Only available to the superuser.
−g *group*	The group id of the new file is set to *group*. Only available to the superuser.
−o	If *file* is found, this option saves the "found" file by copying it to **OLD***file* in the directory in which it was found. This option is useful when installing a frequently used file such as */bin/sh* or */etc/getty*, where the existing file cannot be removed. May be used alone or with any other options except **−c**.
−s	Suppresses printing of messages other than error messages. May be used alone or with any other options.

SEE ALSO

make(1).

NAME

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.

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).

NAME

ipcs − report inter-process communication facilities status

SYNOPSIS

ipcs [options]

DESCRIPTION

ipcs prints certain information about active inter-process communication facilities. Without *options*, information is printed in short format for message queues, shared memory, and semaphores that are currently active in the system. Otherwise, the information that is displayed is controlled by the following *options*:

−q Print information about active message queues.

−m Print information about active shared memory segments.

−s Print information about active semaphores.

If any of the options **−q**, **−m**, or **−s** are specified, information about only those indicated will be printed. If none of these three are specified, information about all three will be printed subject to these options:

−b Print biggest allowable size information. (Maximum number of bytes in messages on queue for message queues, size of segments for shared memory, and number of semaphores in each set for semaphores.) See below for meaning of columns in a listing.

−c Print creator's login name and group name. See below.

−o Print information on outstanding usage. (Number of messages on queue and total number of bytes in messages on queue for message queues and number of processes attached to shared memory segments.)

−p Print process number information. (Process ID of last process to send a message and process ID of last process to receive a message on message queues and process ID of creating process and process ID of last process to attach or detach on shared memory segments) See below.

−t Print time information. (Time of the last control operation that changed the access permissions for all facilities. Time of last *msgsnd* and last *msgrcv* on message queues, last *shmat* and last *shmdt* on shared memory, last *semop*(2) on semaphores.) See below.

−a Use all print *options*. (This is a shorthand notation for **−b**, **−c**, **−o**, **−p**, and **−t**.)

−C *corefile*
Use the file *corefile* in place of **/dev/kmem**.

−N *namelist*
The argument will be taken as the name of a *namelist*

The column headings and the meaning of the columns in an *ipcs* listing are given below; the letters in parentheses indicate the *options* that cause the corresponding heading to appear; **all** means that the heading always appears. Note that these *options* only determine what information is provided for each facility; they do *not* determine which facilities will be listed.

T (all) Type of the facility:

q	message queue;
m	shared memory segment;
s	semaphore.

ID (all) The identifier for the facility entry.

KEY (all) The key used as an argument to *msgget*, *semget*, or *shmget* to create the facility entry. (Note: The key of a shared memory segment is changed to **IPC_PRIVATE** when the segment has been removed until all processes attached to the segment detach it.)

MODE	(all)	The facility access modes and flags: The mode consists of 11 characters that are interpreted as 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.

395

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.
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

/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 to reality.

This page is intentionally left blank

NAME

is_68000, is_68020, is_68030, is_R3000, is_heterogen – identify mcu type

SYNOPSIS

is_68000
is_68020
is_68030
is_R3000
is_heterogen

DESCRIPTION

is_68000 (*is_68020, is_68030*) returns an exit code 0 when invoked on MCU68000 (MCU68020, MCU68030).

is_R3000 returns an exit code 0 when invoked on an R3000 cpu.

is_heterogen returns an exit code 0 when invoked on a Supermax, which contains both MCU68030 and R3000.

This page is intentionally left blank

NAME

join – relational database operator

SYNOPSIS

join [options] file1 file2

DESCRIPTION

join forms, on the standard output, a join of the two relations specified by the lines of *file1* and *file2*. If *file1* is –, the standard input is used.

File1 and *file2* must be sorted in increasing ASCII collating sequence on the fields on which they are to be joined, normally the first in each line [see *sort*(1)].

There is one line in the output for each pair of lines in *file1* and *file2* that have identical join fields. The output line normally consists of the common field, then the rest of the line from *file1*, then the rest of the line from *file2*.

The default input field separators are blank, tab, or new-line. In this case, multiple separators count as one field separator, and leading separators are ignored. The default output field separator is a blank.

Some of the below options use the argument *n*. This argument should be a **1** or a **2** referring to either *file1* or *file2*, respectively. The following options are recognized:

–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.

Filenames that are numeric may cause conflict when the **-o** option is used right before listing filenames.

400

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).

This page is intentionally left blank

NAME

killall – kill all active processes

SYNOPSIS

/etc/killall [signal]

DESCRIPTION

killall is used by **/etc/shutdown** to kill all active processes not directly related to the shutdown procedure.

killall terminates all processes with open files so that the mounted file systems will be unbusied and can be unmounted.

killall sends *signal* (see *kill*[1]) to all processes not belonging to the above group of exclusions. If no *signal* is specified, a default of **9** is used.

FILES

/etc/shutdown

SEE ALSO

fuser(1M), shutdown(1M), signal(2).

WARNINGS

The *killall* command can be run only by the super-user.

This page is intentionally left blank

NAME

labelit – provide labels for file systems

SYNOPSIS

/etc/labelit special [fsname volume [**−n**]]

DESCRIPTION

labelit can be used to provide labels for unmounted disk file systems or file systems being copied to tape. The **−n** option provides for initial labeling only (this destroys previous contents).

With the optional arguments omitted, *labelit* prints current label values.

The *special* name should be the physical disk section (e.g., **/dev/dsk/c0d0s6**),
or the cartridge tape (e.g., **/dev/SA/ctape1**). The device may not be on a remote machine.

The *fsname* argument represents the mounted name (e.g., **root**, **u1**, etc.) of the file system.

Volume may be used to equate an internal name to a volume name applied externally to the disk pack, diskette or tape.

For file systems on disk, *fsname* and *volume* are recorded in the superblock.

SEE ALSO

makefsys(1M), sh(1), fs(4).

This page is intentionally left blank

NAME

led – flash hyphens in MCU displays

SYNOPSIS

/etc/led [−f] [−o]

DESCRIPTION

led is used to make the hyphens in the MCU displays flash The main purpose is to signal particular phases of the boot procedure. The options are as follows:

−f sets the hyphens to a flashing state via the *smsys*(2) system call.

−o sets the hyphens to a constant state via the *smsys*(2) system call.

SEE ALSO

smsys(2).

WARNINGS

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

This page is intentionally left blank

NAME

line – read one line

SYNOPSIS

line

DESCRIPTION

line copies one line (up to a new-line) from the standard input and writes it on the standard output. It returns an exit code of 1 on **EOF** and always prints at least a new-line. It is often used within shell files to read from the user's terminal.

SEE ALSO

sh(1), read(2).

This page is intentionally left blank

NAME

link, unlink – link and unlink files and directories

SYNOPSIS

/etc/link file1 file2
/etc/unlink file

DESCRIPTION

The *link* command is used to create a file name that points to another file. Linked files and directories can be removed by the *unlink* command; however, it is strongly recommended that the *rm*(1) and *rmdir*(1) commands be used instead of the *unlink* command.

The only difference between *ln*(1) and *link*/*unlink* is that the latter do exactly what they are told to do, abandoning all error checking. This is because they directly invoke the *link*(2) and *unlink*(2) system calls.

SEE ALSO

rm(1), link(2), unlink(2).

WARNINGS

These commands can be run only by the super-user.

This page is intentionally left blank

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 *cntrl-d* to indicate an "end-of-file." (See *User's Guide* for instructions on how to establish contact with the **UNIX** system).

If *login* is invoked as a command it must replace the initial command interpreter. This is accomplished by typing:

```
        exec login
```

from the initial shell.

login asks for your user name (if not supplied as an argument), and, if appropriate, your password. Echoing is turned off (where possible) during the typing of your password, so it will not appear on the written record of the session.

At some installations, an option may be invoked that will require you to enter a second "dialup" password. This will occur only for dial-up connections, and will be prompted by the message "dialup password:". Both passwords are required for a successful login.

The optional "dialup" password is activated on the dial-up connections by creating the file **/etc/d_passwd**. (See *d_passwd*(4)). A tty-line gets the status of dial-up connection if it is specified in the **/etc/dialups** file. (See *dialups*(4)).

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.

413

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 path name (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 path names 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, then they replace the older value. There are two exceptions. The variables **PATH** and **SHELL** cannot be changed. This prevents people, logging into restricted shell environments, from spawning secondary shells which are not restricted. Both *login* and *getty* understand simple single-character quoting conventions.

Typing a backslash in front of a character quotes it and allows the inclusion of such things as spaces and tabs.

FILES

/etc/passwd	password file
/etc/dialups	file of dial-up connections (an option)
/usr/d_passwd	"shell" dial-up password file (an option)
/etc/utmp	accounting
/etc/wtmp	accounting
/etc/profile	system profile
/etc/motd	message-of-the-day
/ust/mail/*your-name*	mailbox for user *your-name*
.profile	user's login profile

SEE ALSO

mail(1), newgrp(1), sh(1), su(1M).
passwd(4), d_passwd(4), dialups(4), profile(4), environ(5).

DIAGNOSTICS

login incorrect if the user name or the password cannot be matched.

login incorrect also on dial-up connections if the initial shell or the dial-up password cannot be matched (*d_passwd*(4)).

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.

415

This page is intentionally left blank

NAME

logname – get login name

SYNOPSIS

logname

DESCRIPTION

logname returns the contents of the environment variable **$LOGNAME**, which is set when a user logs into the system.

FILES

/etc/profile

SEE ALSO

env(1), login(1), logname(3X), environ(5).

This page is intentionally left blank

NAME

lp, cancel – send/cancel requests to an LP line printer

SYNOPSIS

lp [**–c**] [**–d**dest] [**–m**] [**–n**number] [**–o**option] [**–s**] [**–t**title] [**–w**] files
cancel [ids] [printers]

DESCRIPTION

lp arranges for the named files and associated information (collectively called a *request*) to be printed by a line printer. If no file names are mentioned, the standard input is assumed. The file name – stands for the standard input and may be supplied on the command line in conjunction with named *files*. The order in which *files* appear is the same order in which they will be printed.

lp associates a unique *id* with each request and prints it on the standard output. This *id* can be used later to cancel (see *cancel*) or find the status (see *lpstat*(1)) of the request.

The following options to *lp* may appear in any order and may be intermixed with file names:

–c Make copies of the *files* to be printed immediately when *lp* is invoked. Normally, *files* will not be copied, but will be linked whenever possible. If the **–c** option is not given, then the user should be careful not to remove any of the *files* before the request has been printed in its entirety. It should also be noted that in the absence of the **–c** option, any changes made to the named *files* after the request is made but before it is printed will be reflected in the printed output.

–d*dest* Choose *dest* as the printer or class of printers that is to do the printing. If *dest* is a printer, then the request will be printed only on that specific printer. If *dest* is a class of printers, then the request will be printed on the first available printer that is a member of the class. Under certain conditions (printer unavailability, file space limitation, etc.), requests for specific destinations may not be accepted (see *accept*(1M) and *lpstat*(1)). By default, *dest* is taken from the environment variable **LPDEST** (if it is set). Otherwise, a default destination (if one exists) for the computer system is used. Destination names vary between systems (see *lpstat*(1)).

−m Send mail (see *mail(1))* after the files have been printed. By default, no mail is sent upon normal completion of the print request.

−n*number* Print *number* copies (default of 1) of the output.

−o*option* Specify printer-dependent or class-dependent *options*. Several such *options* may be collected by specifying the **−o** keyletter more than once. For more information about what is valid for *options*, see **Models** in *lpadmin*(1M).

−s Suppress messages from *lp*(1) such as "request id is ...".

−t*title* Print *title* on the banner page of the output.

−w Write a message on the user's terminal after the *files* have been printed. If the user is not logged in, then mail will be sent instead.

Cancel cancels line printer requests that were made by the *lp*(1) command. The command line arguments may be either request *ids* (as returned by *lp*(1)) or *printer* names (for a complete list, use *lpstat*(1)). Specifying a request *id* cancels the associated request even if it is currently printing. Specifying a *printer* cancels the request which is currently printing on that printer. In either case, the cancellation of a request that is currently printing frees the printer to print its next available request.

420

FILES

/usr/spool/lp/ *

SEE ALSO

enable(1), lpstat(1), mail(1), accept(1M), lpadmin(1M), lpsched(1M).

This page is intentionally left blank

NAME

lpadmin – configure the LP spooling system

SYNOPSIS

/usr/lib/lpadmin **–p** printer [options]
/usr/lib/lpadmin **–x** dest
/usr/lib/lpadmin **–d**[dest]

DESCRIPTION

lpadmin configures line printer (LP) spooling systems to describe printers, classes and devices. It is used to add and remove destinations, change membership in classes, change devices for printers, change printer interface programs and to change the system default destination. *lpadmin* may not be used when the LP scheduler, *lpsched*(1M), is running, except where noted below.

Exactly one of the **–p**, **–d** or **–x** options must be present for every legal invocation of *lpadmin*.

–p*printer* names a *printer* to which all of the *options* below refer. If *printer* does not exist then it will be created.

–x*dest* removes destination *dest* from the LP system. If *dest* is a printer and is the only member of a class, then the class will be deleted, too. No other *options* are allowed with **–x**.

–d[*dest*] makes *dest*, an existing destination, the new system default destination. If *dest* is not supplied, then there is no system default destination. This option may be used when *lpsched*(1M) is running. No other *options* are allowed with **–d**.

The following *options* are only useful with **–p** and may appear in any order. For ease of discussion, the printer will be referred to as *P* below.

–c*class* inserts printer *P* into the specified *class*. *class* will be created if it does not already exist.

−e*printer*	copies an existing *printer's* interface program to be the new interface program for *P*.
−h	indicates that the device associated with *P* is hardwired. This *option* is assumed when adding a new printer unless the **−l** *option* is supplied.
−i*interface*	establishes a new interface program for *P*. *interface* is the path name of the new program.
−l	indicates that the device associated with *P* is a login terminal. The LP scheduler, *lpsched*, disables all login terminals automatically each time it is started. Before re-enabling *P*, its current *device* should be established using *lpadmin*.
−m*model*	selects a model interface program for *P*. *model* is one of the model interface names supplied with the LP Spooling Utilities (see *Models* below).
−r*class*	removes printer *P* from the specified *class*. If *P* is the last member of the *class*, then the *class* will be removed.
−v*device*	associates a new *device* with printer *P*. *device* is the pathname of a file that is writable by *lp*. Note that the same *device* can be associated with more than one *printer*. If only the **−p** and **−v** *options* are supplied, then *lpadmin* may be used while the scheduler is running.

Restrictions.

When creating a new printer, the **−v** option and one of the **−e**, **−i** or **−m** options must be supplied. Only one of the **−e**, **−i** or **−m** options may be supplied. The **−h** and **−l** keyletters are mutually exclusive. Printer and class names may be no longer than 14 characters and must consist entirely of the characters **A−Z**, **a−z**, **0−9** and _ (underscore).

Models.

Model printer interface programs are supplied with the LP Spooling Utilities. They are shell procedures which interface between *lpsched* and devices. All models reside in the directory **/usr/spool/lp/model** and may be used as is with *lpadmin* **−m**. Copies of model interface programs may also be modified and then associated with printers using *lpadmin* **−i**. The following describes the *models* which may be given on the *lp* command line using the **−o** keyletter:

LQP−40	Letter quality printer using XON/XOFF protocol at 9600 baud.
DQP−10	Dot matrix draft quality printer using XON/XOFF protocol at 9600 baud.

EXAMPLES

1. For a DQP-10 printer named cI8, it will use the DQP-10 model interface after the command:

```
/usr/lib/lpadmin −pcI8 −mdqp10
```

2. A LQP-40 printer called pr1 can be added to the lp configuration with the command:

```
/usr/lib/lpadmin −ppr1 −v/dev/contty −mlqp40
```

FILES

/usr/spool/lp/ *

SEE ALSO

accept(1M), enable(1), lp(1), lpsched(1M), lpstat(1).

425

This page is intentionally left blank

NAME

lpsched, lpshut, lpmove – start/stop the LP scheduler and move requests

SYNOPSIS

/usr/lib/lpsched
/usr/lib/lpshut
/usr/lib/lpmove requests dest
/usr/lib/lpmove dest1 dest2

DESCRIPTION

lpsched schedules requests taken by *lp*(1) for printing on line printers (LP's).

lpshut shuts down the line printer scheduler. All printers that are printing at the time *lpshut* is invoked will stop printing. Requests that were printing at the time a printer was shut down will be reprinted in their entirety after *lpsched* is started again.

lpmove moves requests that were queued by *lp*(1) between LP destinations. This command may be used only when *lpsched* is not running.

The first form of the command moves the named *requests* to the LP destination, *dest*. *Requests* are request ids as returned by *lp*(1). The second form moves all requests for destination *dest1* to destination *dest2*. As a side effect, *lp (1)* will reject requests for *dest1*.

Note that *lpmove* never checks the acceptance status (see *accept*(1M)) for the new destination when moving requests.

FILES

/usr/spool/lp/ *

SEE ALSO

accept(1M), enable(1), lp(1), lpadmin(1M), lpstat(1).

This page is intentionally left blank

NAME

lpstat – print LP status information

SYNOPSIS

lpstat [options]

DESCRIPTION

lpstat prints information about the current status of the LP spooling system.

If no *options* are given, then *lpstat* prints the status of all requests made to *lp*(1) by the user. Any arguments that are not *options* are assumed to be request *ids* (as returned by *lp*). *lpstat* prints the status of such requests. *options* may appear in any order and may be repeated and intermixed with other arguments. Some of the keyletters below may be followed by an optional *list* that can be in one of two forms: a list of items separated from one another by a comma, or a list of items enclosed in double quotes and separated from one another by a comma and/or one or more spaces. For example:

```
−u"user1, user2, user3"
```

The omission of a *list* following such keyletters causes all information relevant to the keyletter to be printed, for example:

```
lpstat −o
```

prints the status of all output requests.

−a[*list*]	Print acceptance status (with respect to *lp*) of destinations for requests. *list* is a list of intermixed printer names and class names.
−c[*list*]	Print class names and their members. *list* is a list of class names.
−d	Print the system default destination for *lp*.
−o[*list*]	Print the status of output requests. *list* is a list of intermixed printer names, class names, and request *ids*.

−p[*list*]	Print the status of printers. *list* is a list of printer names.
−r	Print the status of the LP request scheduler
−s	Print a status summary, including the system default destination, a list of class names and their members, and a list of printers and their associated devices.
−t	Print all status information.
−u[*list*]	Print status of output requests for users. *list* is a list of login names.
−v[*list*]	Print the names of printers and the path names of the devices associated with them. *list* is a list of printer names.

FILES

/usr/spool/lp/ *

SEE ALSO

enable(1), lp(1).

NAME

ls – list contents of directory

SYNOPSIS

ls [**–RadCLHxmlnogrtucpFbqisf**] [names]

DESCRIPTION

For each directory argument, *ls* lists the contents of the directory; for each file argument, *ls* repeats its name and any other information requested. The output is sorted alphabetically by default. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments appear before directories and their contents.

There are three major listing formats. The default format is to list one entry per line, the **–C** and **–x** options enable multi-column formats, and the **–m** option enables stream output format. In order to determine output formats for the **–C**, **–x**, and **–m** options, *ls* uses an environment variable, **COLUMNS**, to determine the number of character positions available on one output line. If this variable is not set, the *terminfo*(4) database is used to determine the number of columns, based on the environment variable **TERM**. If this information cannot be obtained, 80 columns are assumed.

The *ls* command has the following options:

–R Recursively list subdirectories encountered.

–L If argument is a symbolic link, list the file or directory the link references rather than the link itself.

–H If the file is a symbolic link, list the file itself.

–a List all entries, including those that begin with a dot (**.**), which are normally not listed.

–d If an argument is a directory, list only its name (not its contents); often used with **–l** to get the status of a directory.

−C Multi-column output with entries sorted down the columns.

−x Multi-column output with entries sorted across rather than down the page.

−m Stream output format; files are listed across the page, separated by commas.

−l List in long format, giving mode, number of links, owner, group, size in bytes, and time of last modification for each file (see below). If the file is a special file, the size field will instead contain the major and minor device numbers rather than a size.

−n The same as **−l**, except that the owner's **UID** and group's **GID** numbers are printed, rather than the associated character strings.

−o The same as **−l**, except that the group is not printed.

−g The same as **−l**, except that the owner is not printed.

−r Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.

−t Sort by time stamp (latest first) instead of by name. The default is the last modification time. (See **−n** and **−c**.)

−u Use time of last access instead of last modification for sorting (with the **−t** option) or printing (with the **−l** option).

−c Use time of last modification of the i-node (file created, mode changed, etc.) for sorting (**−t**) or printing (**−l**).

−p Put a slash (/) after each filename if that file is a directory.

−F Put a slash (/) after each filename if that file is a directory and put an asterisk (*) after each filename if that file is executable. If the file is a symbolic link put an commercial at (@) after the filename.

−b Force printing of non-printable characters to be in the octal **\ddd** notation.

−q Force printing of non-printable characters in file names as the character question mark (**?**).

−i For each file, print the i-number in the first column of the report.

−s Give size in blocks, including indirect blocks, for each entry.

−f Force each argument to be interpreted as a directory and list the name found in each slot. This option turns off **−l, −t, −s,** and **−r,** and turns on **−a;** the order is the order in which entries appear in the directory.

The mode printed under the **−l** option consists of 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;
p	the entry is a fifo (a.k.a. ”named pipe”) special file;
−	the entry is an ordinary file.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to the owner's permissions; the next to permissions of others in the user-group of the file; and the last to all others. Within each set, the three characters indicate permission to read, to write, and to execute the file as a program, respectively. For a directory, ”execute” permission is interpreted to mean permission to search the directory for a specified file.

ls −l (the long list) prints its output as follows:

```
−rwxrwxrwx  1 smith  dev     10876  May 16 9:42 part2
```

This horizontal configuration provides a good deal of information. Reading from right to left, you see that the current directory holds one file, named ”part2.” Next, the last time that file's contents were modified was 9:42 A.M. on May 16. The file is moderately sized, containing 10,876 characters, or bytes. The owner of the file, or the user, belongs to the group "dev" (perhaps indicating "development"), and his or her login name is "smith." The number, in this case "1," indicates the number of links to file "part2." Finally, the row of dash and letters tell you that user, group, and others have permissions to read, write, execute "part2."

The execute (**x**) symbol here occupies the third position of the three-character sequence. A – in the third position would have indicated a denial of execution permissions.

The permissions are indicated as follows:

r	the file is readable
w	the file is writable
x	the file is executable
–	the indicated permission is *not* granted
l	mandatory locking will occur during access (the set-group-ID bit is on and the group execution bit is off)
s	the set-user-ID or set-group-ID bit is on, and the corresponding user or group execution bit is also on
S	undefined bit-state (the set-user-ID bit is on and the user execution bit is off)
t	the 1000 (octal) bit, or sticky bit, is on (see *chmod*(1)), and execution is on
T	the 1000 bit is turned on, and execution is off (undefined bit-state)

For user and group permissions, the third position is sometimes occupied by a character other than **x** or –. **s** also may occupy this position, referring to the state of the set-ID bit, whether it be the user's or the group's. The ability to assume the same ID as the user during execution is, for example, used during login when you begin as root but need to assume the identity of the user stated at "login."

In the case of the sequence of group permissions, **l** may occupy the third position. **l** refers to mandatory file and record locking. This permission describes a file's ability to allow other files to lock its reading or writing permissions during access.

For others permissions, the third position may be occupied by **t** or **T**. These refer to the state of the sticky bit and execution permissions.

EXAMPLES

An example of a file's permissions is:

```
−rwxr−−r−−
```

This describes a file that is readable, writable, and executable by the user and readable by the group and others.

Another example of a file's permissions is:

```
−rwsr−xr−x
```

This 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.

Another example of a file's permissions is:

```
−rw−rwl−−−
```

This describes a file that is readable and writable only by the user and the group and can be locked during access.

An example of a command line:

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.

Another example of a command line:

ls −aisn

This command will provide you with quite a bit of information including **a**ll 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 **s**ize (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.

FILES

/etc/passwd	user IDs for **ls −l** and **ls −o**
/etc/group	group IDs for **ls −l** and **ls −g**
/usr/lib/terminfo/?/ *	terminal information database

SEE ALSO

chmod(1), find(1).

BUGS

Unprintable characters in file names may confuse the columnar output options.

NAME

machid: m68k, pdp11, u3b, u3b2, u3b5, vax – get processor type truth value

SYNOPSIS

m68k

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 an M68000 or M68020 processor.
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).

This page is intentionally left blank

NAME

mail, rmail – send mail to users or read mail

SYNOPSIS

mail [**–bepq**] [**–f** file]

mail persons

rmail persons

DESCRIPTION

mail without arguments prints a user's mail, message-by-message, in last-in, first-out order. For each message, the user is prompted with a **?**, and a line is read from the standard input to determine the disposition of the message:

<*new-line*>	Go on to next message.
+	Same as <new-line>.
n	Same as <new-line>.
d	Delete message and go on to next message.
p	Print message again.
–	Go back to previous message.
s [*files*]	Save message in the name *files* (**mbox** is default).
y	Same as save.
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* (yourself is default).
q	Put undeleted mail back in the *mailfile* and stop.

EOT (control-d)	Same as **q**.
x	Put all mail back in the *mailfile* unchanged and stop.
!*command*	Escape to the shell to do *command*.
?	Print a command summary.

The optinal arguments alter the printing of the mail:

−e	causes the mail not to be printed. An exit value of zero (0) is returned if the user has mail; otherwise, an exit value of 1 is returned.
−p	causes all mail to be printed without prompting for disposition.
−q	causes *mail* to terminate after interrupts. Normally an interrupt only causes the termination of the message being printed.
−b	causes message to be printed in first-in, first-out order.
−f*file*	causes *mail* to use *file* (e.g.**mbox**) instead of the default *mailfile*.

When *persons* are named, *mail* takes the standard input up to an end-of-file (or up to aline consisting of just a .) and adds it to each *person's mailfile*. The message is preceded by the sender's name and a postmark. Lines that look like postmarks in the message, (i.e. "From ...") are preceded with a >. A *person* is usually a user name recognized by *login*(1). If a *person* being send is not recognized, or if *mail* is interrupted during input, the file **dead.letter** will be saved to allow editing and resending. Note that this is regarded as a temporary file in that it is recreated every time needed, erasing the previous contents of **dead.letter**.

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*. This is especially useful to forward all of a person's mail to one machine in a multiple machine environment. In order for forwarding to work properly the *mailfile* should have "mail" as group ID, and the group permission should be read-write.

rmail only permits the sending of mail; *uucp*(1) uses *rmail* as a security precaution.

When a user logs in, the presence of mail, if any, is indicated. Also, notification is made if new mail arrives while using *mail*.

Mail may be sent to a recipient on a remote system if you have the Basic Networking Utilities or SupermaxTCP installed.

The addressing of remote users using domain addressing is described in the *mailaddr*(5) manual page. This is recommended although the old address format is still supported to some extent.

Domain addresses cannot be used as a forward address, use instead *sendmail*'s forward mechanism.

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 path name of *mailfile*
/tmp/ma*	temporary file
/usr/mail/*.lock	lock for mail directory
dead.letter	unmailable text

SEE ALSO

login(1), mailx(1), sendmail(1), write(1), mailaddr(5).

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."

BUGS

Conditions sometimes result in a failure to remove a lock file.
After an interrupt, the next message may not be printed; printing may be forced by typing a **p**.

NAME

mailx – interactive message processing system

SYNOPSIS

mailx [*options*] [*name...*]

DESCRIPTION

The command *mailx* provides a comfortable, flexible environment for sending and receiving messages electronically. When reading mail, *mailx* provides commands to facilitate saving, deleting, and responding to messages. When sending mail, *mailx* allows editing, reviewing and other modification of the message as it is entered.

Many of the remote features of *mailx* will only work if the Basic Networking Utilities are installed on your system.

Incoming mail is stored in a standard file for each user, called the *mailbox* for that user. When *mailx* is called to read messages, the *mailbox* is the default place to find them. As messages are read, they are marked to be moved to a secondary file for storage, unless specific action is taken, so that the messages need not be seen again. This secondary file is called the *mbox* and is normally located in the user's HOME directory (see "MBOX" (ENVIRONMENT VARIABLES) for a description of this file). Messages can be saved in other secondary files named by the user. Messages remain in a secondary file until forcibly removed.

The user can access a secondary file by using the **–f** option of the *mailx* command. Messages in the secondary file can then be read or otherwise processed using the same COMMANDS as in the primary *mailbox*. This gives rise within these pages to the notion of a current *mailbox*.

On the command line, *options* start with a dash (–) and any other arguments are taken to be destinations (recipients). If no recipients are specified, *mailx* will attempt to read messages from the *mailbox*. Command line options are:

−e	Test for presence of mail. *mailx* prints nothing and exits with a successful return code if there is mail to read.
−f [*filename*]	Read messages from *filename* instead of *mailbox*. If no *filename* is specified, the *mbox* is used.
−F	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 also "ignore" (ENVIRONMENT VARIABLES).
−n	Do not initialize from the system default *mailx.rc* file.
−N	Do not print initial header summary.
−r *address*	Pass *address* to network delivery software. All tilde commands are disabled. (See **addsopt** under ENVIRONMENT VARIABLES)
−s *subject*	Set the Subject header field to *subject*.
−u *user*	Read *user*'s *mailbox*. This is only effective if *user*'s *mailbox* is not read protected.
−U	Convert *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 below). When sending mail, *mailx* is in *input mode*. If no subject is specified on the command line, a prompt for the subject is printed. (A "subject" longer than 1024 characters will cause *mailx* to dump core) As the message is typed, *mailx* will read the message and store it in a temporary file. Commands may be entered by beginning a line with the tilde (˜) escape character followed by a single command letter and optional arguments. See TILDE ESCAPES for a summary of these commands.

At any time, the behavior of *mailx* is governed by a set of *environment variables*. These are flags and valued parameters which are set and cleared via the **set** and **unset** commands. See ENVIRONMENT VARIABLES below for a summary of these parameters.

Recipients listed on the command line may be of three types: login names, shell commands, or alias groups. Login names may be any network address, including mixed network addressing. If mail is found to 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 below) and are lists of recipients of any type.

Regular commands are of the form

[**command**] [*msglist*] [*arguments*]

If no command is specified in *command mode*, **print** is assumed. In *input mode*, commands are recognized by the escape character, and lines not treated as commands are taken as input for the message.

Each message is assigned a sequential number, and there is at any time the notion of a current message, marked by a right angle bracket (>) in the header summary. Many

445

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 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)).

448

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 **q**uit).

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 **F**ollowup, **S**ave, and **C**opy 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 **P**rint and **T**ype 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 VARIABLES) for a description of this file. See also the **exit** and **quit** commands.

next [*message*]

Go to next message matching *message*. A *msglist* may be specified, but in this case the first valid message in the list is the only one used. This is useful for jumping to the next message from a specific user, since the name would be taken as a command in the absence of a real command. See the discussion of *msglists* above for a description of possible message specifications.

pipe [*msglist*] [*shell-command*]
| [*msglist*] [*shell-command*]
Pipe the message through the given *shell-command*. The message is treated as if it were read. If no arguments are given, the current message is piped through the command specified by the value of the "cmd" variable. If the "page" variable is set, a form feed character is inserted after each message (see ENVIRONMENT VARIABLES).

preserve [*msglist*]
hold [*msglist*]
Preserve the specified messages in the *mailbox*.

Print [*msglist*]
Type [*msglist*]
Print the specified messages on the screen, including all header fields. Overrides suppression of fields by the **ig**nore 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.

452

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.

456

˜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 WARNINGS below)

allnet
: All network names whose last component (login name) match are treated as identical. This causes the *msglist* message specifications to behave similarly. Default is **noallnet**. See also the **alternates** command and the "metoo" variable.

append
: Upon termination, append messages to the end of the *mbox* file instead of prepending them. Default is **noappend.**

askcc
: Prompt for the Cc list after message is entered. Default is **noaskcc**.

asksub
: Prompt for subject if it is not specified on the command line with the **−s** option. Enabled by default.

autoprint
: Enable automatic printing of messages after **delete** and **undelete** commands. Default is **noautoprint**.

459

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.

461

header
: Enable printing of the header summary when entering *mailx*. Enabled by default.

hold
: Preserve all messages that are read in the *mailbox* instead of putting them in the standard *mbox* save file. Default is **nohold**.

ignore
: Ignore interrupts while entering messages. Handy for noisy dial-up lines. Default is **noignore**.

ignoreeof
: Ignore end-of-file during message input. Input must be terminated by a period (.) on a line by itself or by the **˜.** command. Default is **noignoreeof**. See also "dot" above.

keep

When the *mailbox* is empty, truncate it to zero length instead of removing it. Disabled by default.

keepsave

Keep messages that have been saved in other files in the *mailbox* instead of deleting them. Default is **nokeepsave**.

MBOX=*filename*

The name of the file to save messages which have been read. The **xit** command overrides this function, as does saving the message explicitly in another file. Default is $HOME/mbox.

metoo

If your login appears as a recipient, do not delete it from the list. Default is **nometoo**.

LISTER=*shell-command*

The command (and options) to use when listing the contents of the "folder" directory. The default is *ls*(1).

onehop

When responding to a message that was originally sent to several recipients, the other recipient addresses are normally forced to be relative to the originating author's machine for the response. This flag disables alteration of the recipients' addresses, improving efficiency in a network where all machines can send directly to all other machines (i.e., one hop away).

outfolder

Causes the files used to record outgoing messages to be located in the directory specified by the "folder" variable unless the path name is absolute. Default is

nooutfolder. See "folder" above and the **Save**, **Copy**, **followup**, and **Followup** commands.

page
: Used with the **pipe** command to insert a form feed after each message sent through the pipe. Default is **nopage**.

PAGER=*shell-command*
: The command to use as a filter for paginating output. This can also be used to specify the options to be used. Default is *pg*(1).

prompt=*string*
: Set the *command mode* prompt to *string*. Default is "? ".

quiet
: Refrain from printing the opening message and version when entering *mailx*. Default is **noquiet**.

record=*filename*
: Record all outgoing mail in *filename*. Disabled by default. See also "outfolder" above.

save
: Enable saving of messages in *dead.letter* on interrupt or delivery error. See "DEAD" for a description of this file. Enabled by default.

screen=*number*
: Sets the number of lines in a screen–full of headers for the **headers** command.

sendmail=*shell-command*
: Alternate command for delivering messages. Default is */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).

465

This page is intentionally left blank

NAME

makefsys – create a file system on a diskette

SYNOPSIS

makefsys

DESCRIPTION

This command allows the user to create a file system on a diskette. It also writes an internal label in the file system super-block.

The user is asked some questions before the file system is created. Once created, the diskette is self-identifying.

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

sysadm makefsys

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

SEE ALSO

checkfsys(1M), labelit(1M), mkfs(1M), mountfsys(1M), sysadm(1).

This page is intentionally left blank

NAME

makeos – generate a bootable operating system

SYNOPSIS

/etc/boot.d/makeos

DESCRIPTION

makeos asks the user a number of questions about the operating system to be generated. Each question is answered by either **y** or **n**.

Based on the answers given by the user, *makeos* creates a bootable module called **/etc/boot.d/os00** for an MC68000 based MCU and **/etc/boot.d/os20** for an MC68020 based MCU. The file created can later be placed on the boot device by the *bootgen*(1M) command.

Note: The generation of an operating system using *makeos* must not be confused with the configuration of an already generated operating system with the *chhw*(1M) command.

SEE ALSO

bootgen(1M).

This page is intentionally left blank

NAME

makexos – generate an Operating System Extension

SYNOPSIS

makexos [system-file]

DESCRIPTION

makexos is used to generate a loadable operating system extension (a so-called XOS) module that may be loaded using the *ins_xos*(1M) program.

An XOS module is a collection of operating system services that supplement the services found in the operating system and loaded as part of the bootstrap process. Typical elements in the XOS module are the STREAMS mechanism and communication modules, such as TCP/IP.

When *makexos* is invoked, a number of files must be present in the current directory, (which will typically be */etc/boot.d*). These files include:

os20 or *os30, XOS.o, oslib.a, NIOC.o, SP.o, TIMOD.o, TIRDWR.0*, and others.

The *os20* or *os30* module must be the one that has been put onto the boot disk using *bootgen*(1M).

When invoked without argument, *makexos* will ask the user a number of questions about the XOS that will be generated. The questions relate to the modules that the user wants to include in the XOS. When all the questions have been answered, an XOS module is generated. It will have the name:

X–str.########

where *########* is a hexadecimal number identifying the operating system version. After the system has been bootstrapped, this XOS module may be loaded using the *ins_xos*(1M) command.

makexos will also generate a tect file called *system*. This file may be used as an argument to *makexos*, when an XOS module is to be generated for another version of the operating system.

When *makexos* is invoked with such an argument, it will not ask the user any questions; instead it will generate an XOS module with the same components as the last time.

SEE ALSO

bootgen(1m), ins_xos(1M).

NAME

man – display reference manual pages; find reference pages by keyword

SYNOPSIS

/usr/bin/man [–] [**–M** path] [[section] title . . .] title . . .
/usr/bin/man [**–M** path] **–k** keyword . . .
/usr/bin/man [**–M** path] **–f** filename . . .

DESCRIPTION

The *man* command displays information from the reference manuals. It can display complete manual pages that you select by *title*, or one-line summaries selected either by *keyword*(**–k**), or by the name of an associated file (**–f**).

A *section*, when given, applies to the *titles* that follow it on the command line (up to the next *section*, if any). *man* looks in the indicated section of the manual for those *titles*. *section* is either a digit (perhaps followed by a single letter indicating the type of manual page), or one of the words **new**, **local**, **old**, or **public**. If *section* is omitted, **man** searches all reference sections (giving preference to commands over functions) and prints the first manual page it finds. If no manual page is located, *man* prints an error message.

The reference page sources are typically located in the **/usr/man/man?** directories. Since these directories are optionally installed, they may not reside on your host. If there are preformatted, up-to-date versions in corresponding **/usr/man/cat?** directories, *man* simply displays or prints those versions.

If the standard output is not a terminal, or if the – flag is given, **man** pipes its output through **cat**. Otherwise, **man** pipes its output through *more* to handle paging and underlining on the screen.

The following options are available:

−M path
: Change the search path for manual pages. *path* is a colon-separated list of directories that contain manual page directory subtrees. When used with the **−k** or **−f** options, the **−M** option must appear first. Each directory in the *path* is assumed to contain subdirectories of the form *man*[1-8l-p] or *cat*[1-8l-p].

−k keyword ...
: *man* prints out one-line summaries from the *whatis* database (table of contents) that contain any of the given *keywords*.

−f filename ...
: *man* attempts to locate manual pages related to any of the given *filenames*. It strips the leading pathname components from each *filename*, and then prints one-line summaries containing the resulting basename or names.

ANUAL PAGES

Manual pages are installed preformatted.

NVIRONMENT

MANPATH	If set, its value overrides **/usr/man** as the default search path. The **−M** flag, in turn, overrides this value.
PAGER	A program to use for interactively delivering *man*'s output to the screen. If not set, '*more −s*' (see *more*(1)) is used.

LES

/usr/man	root of the standard manual page directory subtree.
/usr/man/cat?/*	manual entries preformatted.
/usr/man/whatis	table of contents and keyword database.

E ALSO

apropos(1), cat(1), whatis(1), and more(1) in the *System V Reference Manual*.

NOTES

The manual is supposed to be reproducible either on a phototypesetter or on an ASCII terminal. However, on a terminal some information (indicated by font changes, for instance) is necessarily lost.

This page is intentionally left blank

NAME

mcumask – set MCU mask

SYNOPSIS

mcumask [000]

DESCRIPTION

The mcumask determines which MCUs (Main Computing Unit) a process is allowed to spawn new processes on in a Supermax multi cpu environment. The argument will be interpreted as an octal number and each bit in this number refers to an MCU. If a bit is set, access is allowed to that MCU. When a user logs in, his *mcumask* is set to allow access to all configured MCUs. If the argument is omitted, the current value of the mask is printed.

Only the superuser is allowed to extend his *mcumask* to include new MCUs.

mcumask is recognized and executed by the shell.

471

EXAMPLE

```
mcumask 11
```

will allow access to MCU number 0 and 3.

SEE ALSO

mcumask(2).

This page is intentionally left blank

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.

This page is intentionally left blank

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 This option allows users to specify the mode to be used for new directories. Choices for modes can be found in *chmod*(1).

−p With this option, *mkdir* creates *dirname* by creating all the non-existing parent directories first.

EXAMPLE

To create the subdirectory structure **ltr/jd/jan**, type:

```
mkdir -p ltr/jd/jan
```

SEE ALSO

rm(1), sh(1), umask(1), intro(2&3), 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 non-zero. An error code is stored in *errno*.

475

This page is intentionally left blank

NAME

mkfs, mkfs512 – construct a file system

SYNOPSIS

/etc/mkfs *special* blocks[:inodes] [gap blocks/cyl]
/etc/mkfs *special* proto [gap blocks/cyl]
/etc/mkfs512 *special* blocks[:inodes] [gap blocks/cyl]
/etc/mkfs512 *special* proto [gap blocks/cyl]

DESCRIPTION

mkfs constructs a file system by writing on the special file according to the directions found in the remainder of the command line.

The command waits 10 seconds before starting to construct the file system. During this 10-second pause the command can be aborted by entering a delete (DEL).

If the second argument is a string of digits, the size of the file system is the value of *blocks* interpreted as a decimal number. This is the number of 512 byte disk blocks the file system will occupy. If the number of i-nodes is not given, the default is the number of *logical* blocks divided by 4, (minimum 32). *mkfs* builds a file system with a single empty directory on it. The boot program block (block zero) is left uninitialized.

If the second argument is %, *mkfs* will automatically calculate the correct number of logical blocks and i-nodes, based upon the size of the logical disk.

If the second argument is the name of a file that can be opened, *mkfs* assumes it to be a prototype file *proto*, and will take its directions from that file. The prototype file contains tokens separated by spaces or new-lines. A sample prototype specification follows (line numbers have been added to aid in the explanation):

```
1.      /stand/diskboot
2.      4872 110
3.      d--777 3 1
4.      usr     d--777 3 1
5.              sh          ---755 3 1 /bin/sh
6.              ken         d--755 6 1
7.                          $
8.              b0          b--644 3 1 0 0
9.              c0          c--644 3 1 0 0
10.             $
11.     $
```

Line 1 in the example is the name of a file to be copied onto block zero as the bootstrap program. (Under SMOS V this block is not used, so specify */dev/null*).

Line 2 specifies the number of 512 byte blocks the file system is to occupy and the number of i-nodes in the file system. The maximum number of i-nodes configurable is 65500. (Under SMOS V this number can be replaced by a % sign, and the number of blocks and i-nodes will be calculated by *mkfs*, based upon the size of the logical disk).

Lines 3 – 9 tells *mkfs* about files and directories to be included in this file system.

Line 3 specifies the root directory.

Lines 4 – 6 and 8 – 9 specifies other directories and files.

The **$** on line 7 tells *mkfs* to end the branch of the file system it is on, and to continue from the next higher directory. The **$** on lines 10 and 11 ends the process, since no additional specifications follow.

File specifications gives the mode, the user ID, the group ID, and the initial contents of the file. Valid syntax for the contents field depends on the first character of the mode.

The mode for a file is specified by a 6-character string. The first character specifies the type of the file. The character range is **−bcd** to specify regular, block special, character special and directory files, respectively. The second character of the mode is either **u** or **−** to specify set−user−id mode or not. The third character is **g** or **−** for the set−group−id mode. The rest of the mode is a 3 digit octal number giving the owner, group, and other read, write, execute permissions, (see *chmod*(1)).

Two decimal number tokens come after the mode; they specify the user and group ID's of the owner of the file.

If the file is a regular file, the next token of the specification may be a pathname whence the contents and size are copied. If the file is a block or character special file, two decimal numbers follow, which give the major and minor device numbers. If the file is a directory, *mkfs* makes the entries **.** and **..** , and then reads a list of names and (recursively) file specifications for the entries in the directory. As noted above, the scan is terminated with the token **$**.

479

The final argument in both forms of the command specifies the rotational *gap* and the number of *blocks/cyl*.

mkfs512

mkfs512 is the same as *mkfs*, except *mkfs512* is used for file systems with a 512 byte block size.

SEE ALSO

chmod(1), dir(4), fs(4).

BUGS

With a prototype file, it is not possible to copy in a file larger than 64K bytes, nor is there a way to specify links. The maximum number of i-nodes configurable is 65500.

This page is intentionally left blank

NAME

mknod – build special file

SYNOPSIS

/etc/mknod name **b** | **c** major minor
/etc/mknod name **p**

DESCRIPTION

mknod makes a directory entry and corresponding i-node for a special file.

The first argument is the *name* of the entry. The UNIX System convention is to keep such files in the /dev directory.

In the first case, the second argument is **b** if the special file is block-type (disks, tape) or **c** if it is character-type (other devices). The last two arguments are numbers specifying the *major* device type and the *minor* device (e.g., unit, drive, or line number). They may be either decimal or octal. The assignment of major device numbers is specific to each system. The information is contained in the system source file **conf.c**. You must be the super-user to use this form of the command.

The second case is the form of the *mknod* that is used to create FIFO's (a.k.a named pipes).

SEE ALSO

mknod(2).

This page is intentionally left blank

NAME

mkwboot − specify a subdisk as winchester boot disk

SYNOPSIS

/etc/mkwboot [**−d**
/etc/mkwboot [**−b** *x*] [**−a**] [**−s** *disk*]
/etc/mkwboot [**−b** *x*] [**−a**] [**−c**]

DESCRIPTION

The *mkwboot* program is used to specify a subdisk as a winchester boot disk.

The Supermax supports up to 4 different winchester bootdisks numbered from 0 to 3.

The options are:

−d	Display boot disks.
−b*x*	The following action is for boot entry 'x'. The 'x' must be from 0 to 3.
−a	Activate disk. This means that a boot command will cause the system to boot from boot entry 'x'.
−c	Clear boot entry 'x'.
−s*disk*	Set boot entry 'x' to boot disk on *disk*.

The parameter *disk* is the specialfile connected to the subdisk that should be used as boot disk. A boot disk **must** be located on the same physical disk as other boot disks.

SEE ALSO

boot(1M).

This page is intentionally left blank

NAME

more – file persual filter for CRT's

SYNOPSIS

/usr/bin/more [files]

DESCRIPTION

The *more* command is actually just a System III more simulator. It just execute *pg* with option **–p, –n** and **–s.**

SEE ALSO

pg(1).

This page is intentionally left blank

NAME

mount, umount – mount and unmount file system

SYNOPSIS

/etc/mount [[**–r**] [**–f** fstyp] [**–o** options] fsname directory]
/etc/umount mountpoint
/etc/umount **–a**

DESCRIPTION

File systems other than **root** (/) are considered *removable* in the sense that they can be either available to users or unavailable. *mount* announces to the system that a removable file system *fsname* is present, and is available to users. The *directory* must exist already; it becomes the name of the root of the newly mounted file system. *fsname* specifies the file system by the the *special file* for local file systems, and at the form **host:path** for remote (NFS) file systems.

umount announces to the system that the file system previously mounted at *mountpoint* should be removed. *mountpoint* is either the *fsname* or the *directory* used in the corresponding *mount* command. *umount* called with option **–a** tries to umount all file systems currently mounted.

mount, when entered with arguments, adds an entry to the table of mounted devices, */etc/mnttab*. *umount* removes the entry. If invoked with no arguments, *mount* prints the entire mount table.

The following options are available:

–r	Indicate that the file system is to be mounted read-only. *mount* will mount a file system on physically write protected media only if the commands includes the **–r** flag.
–f	Indicates the file system type. The accepted types are: **s5** (local UNIX SYS-V file system), and **nfs** (remote NFS file system). Default is that if *fsname* includes a colon ":", the type is set to **nfs**; otherwise the file system type is set to **s5**.

−o Specifies options for **nfs** type file systems. The NFS options are:

bg	Retry the mount in background if the first attempt fails.
fg	Retry mount in foreground.
retry=n	Set number of failed mount retries to **n**.
rsize=n	Set read buffer size to **n** bytes.
wsize=n	Set write buffer to **n** byte.
timeo=n	Set retransmission timeout to **n** tenth of a second.
retrans=n	Set number of retransmissions to **n**.
port=n	Call servers nfs service at IP port number **n**.
soft	IO requests fails if server does not respond.
hard	IO requests are transmitted until server responds.
suid	Set-uid file mode permitted.
nosuid	Set-uid file mode ignored.
ac_timeo=n	Set attribute cache timeout to **n** tenth of a second.
dc_timeo=n	Set data cache timeout to **n** tenth of a second.

The default settings are as follows:

−orsize=4096,wsize=4096,timeo=30,retrans=10, ac_timeo=30,dc_timeo=30,hard, suid,retry=10000, port=2049,fg

EXAMPLES

mount /dev/dsk/u14c8s1 /usr

Mount the file system from the local disk /dev/dsk/u14c8s1 at the directory /usr.

mount srv:/public/usr/src /usr/src

Mount the file system /public/usr/src on NFS server "srv" at the directory /usr/src.

```
mount –orsize=2048,wsize=2048,bg\
                              srv:/public/usr/src   /usr/src
```

Same as above but with other NFS options.

FILES

/etc/mnttab	mount table

SEE ALSO

fuser(1M), setmnt(1M), mount(2), umount(2), mnttab(4).

DIAGNOSTICS

If the mount system call fails, *mount* prints an appropriate diagnostic. *mount* issues a warning if the file system to be mounted is currently mounted under another name.

umount fails if the special file is not mounted or if it is busy. The file system is busy if it contains an open file or some user's working directory. In such a case, *fuser*(1M) can be of help.

WARNINGS

Physically removing a mounted file system diskette from the diskette drive before issuing the *umount* command damages the file system.

This page is intentionally left blank

NAME

mountall, umountall – mount, unmount multiple file systems

SYNOPSIS

/etc/mountall [**–**] [file–system–table] . . .
/etc/umountall [**–k**]

DESCRIPTION

These commands may be executed only by the super-user.

mountall is used to mount file systems according to one or more *file-system-tables*. */etc/fstab* is the normal file system table. The special file name "–" reads from the standard input.

Before each file system is mounted, it is checked using *fsstat*(1M) to see if it appears mountable. If the file system does not appear mountable, it is checked, using *fsck*(1M), before the mount is attempted.

umountall causes all mounted file systems except **root** to be unmounted. The **–k** option sends a SIGKILL signal, via *fuser*(1M), to processes that have files open.

FILES

File-system-table format:

column 1	file system specification (*mount*(1M) syntax)
column 2	mount-point directory
column 3 +	*mount*(1M) options

White-space separates columns. Lines beginning with "#" are comments. Empty lines are ignored.

A typical file-system-table might read:

```
/dev/dsk/u14c8s1  /usr  -r
srv:/public/usr/srv  /usr/srv  -obg
```

SEE ALSO

fsck(1M), fsstat(1M), fuser(1M), mount(1M), sysadm(1), signal(2), fstab(4).

DIAGNOSTICS

No messages are printed if the file systems are mountable and clean.

Error and warning messages come from *fsck*(1M), *fsstat*(1M), and *mount*(1M).

NAME

mountfsys, umountfsys – mount, unmount a diskette file system

SYNOPSIS

mountfsys [**−y**] [**−r**]
umountfsys [**−y**]

DESCRIPTION

The *mountfsys* command mounts a file system that is on a removable disk so that users can read and write on it. The options provide the following:

−r	the file system is mounted read-only.
−y	suppresses any questions asked during mounting or unmounting.

The *umountfsys* command unmounts the file system.

By default, the name of the file system is displayed and the user is asked if it should be mounted. The optional **−y** argument suppresses questions and mounts or unmounts the file system immediately.

The identical functions are available under the *sysadm* menu:

sysadm mountfsys
sysadm umountfsys

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

SEE ALSO

checkfsys(1M), makefsys(1M), mount(1M), sysadm(1).

WARNING

ONCE THE DISK IS MOUNTED IT MUST NOT BE REMOVED FROM THE DISK DRIVE UNTIL IT HAS BEEN UNMOUNTED!

Removing the disk while it is still mounted can cause severe damage to the data on the disk.

BUGS

A file system that has no label cannot be mounted with the *mountfsys* command.

NAME

mvdir – move a directory

SYNOPSIS

/etc/mvdir *dirname name*

DESCRIPTION

mvdir moves directories within a file system. *dirname* must be a directory. If *name* does not exist, it will be created as a directory. If *name* does exist, *dirname* will be created as *name/dirname*. *dirname* and *name* may not be on the same path; that is, one may not be subordinate to the other. For example:

```
mvdir x/y x/z
```

is legal, but

```
mvdir x/y x/y/z
```

is not.

SEE ALSO

mkdir(1), mv(1).

WARNINGS

Only the super-user can use *mvdir*.

This page is intentionally left blank

NAME

ncheck – generate path names from i-numbers

SYNOPSIS

/etc/ncheck [**−i** i-numbers] [**−a**] [**−s**]
[file-system]

DESCRIPTION

ncheck with no arguments generates a path-name vs. i-number list of all files on a set of default file systems (see */etc/checklist*). Names of directory files are followed by **/.**.

The options are as follows:

−i limits the report to only those files whose i-numbers follow.

−a allows printing of the names **.** and **..**, which are ordinarily suppressed.

−s limits the report to special files and files with set-user-ID mode. This option may be used to detect violations of security policy.

file system must be specified by the file system's special file.

The report should be sorted so that it is more useful.

SEE ALSO

fsck(1M), sort(1).

DIAGNOSTICS

If the file system structure is not consistent, **??** denotes the "parent" of a parentless file and a path-name beginning with **...** denotes a loop.

This page is intentionally left blank

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 "**–e**15 **–l**60" will yield results different from "**–l**60 **–e**15". Options are applied to all *files* on the command line.

495

–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.

497

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

newgrp – log in to a new group

SYNOPSIS

newgrp [–] [group]

DESCRIPTION

newgrp changes a user's group identification. The user remains logged in and the current directory is unchanged, but calculations of access permissions to files are performed with respect to the new real and effective group IDs. The user is always given a new shell, replacing the current shell, by *newgrp*, regardless of whether it terminated successfully or due to an error condition (i.e., unknown group).

Exported variables retain their values after invoking *newgrp*; however, all unexported variables are either reset to their default value or set to null. System variables (such as PS1, PS2, PATH, MAIL, and HOME), unless exported by the system or explicitly exported by the user, are reset to default values. For example, a user has a primary prompt string (PS1) other than **$** (default) and has not exported PS1. After an invocation of *newgrp* , successful or not, their PS1 will now be set to the default prompt string **$**. Note that the shell command *export* (see *sh*(1)) is the method to export variables so that they retain their assigned value when invoking new shells.

With no arguments, *newgrp* changes the group identification back to the group specified in the user's password file entry. This is a way to exit the effect of an earlier *newgrp* command.

If the first argument to *newgrp* is a –, the environment is changed to what would be expected if the user actually logged in again as a member of the new group.

A password is demanded if the group has a password and the user does not, or if the group has a password and the user is not listed in **/etc/group** as being a member of that group.

FILES

/etc/group	system's group file
/etc/passwd	system's password file

SEE ALSO

login(1), sh(1), group(4), passwd(4), environ(5).

BUGS

There is no convenient way to enter a password into **/etc/group**. Use of group passwords is not encouraged, because, by their very nature, they encourage poor security practices. Group passwords may disappear in the future.

NAME

newpkg – installation of new software package

SYNOPSIS

newpkg [device]

DESCRIPTION

newpkg(1) installs a software package from the specified device. If no device is specified **/dev/flop** is assumed. The device may either be a floppy or a streamer. *newpkg* writes a completion message when the installation is completed.

SEE ALSO

rmpkg(1)

This page is intentionally left blank

NAME

news – print news items

SYNOPSIS

news [**−a**] [**−n**] [**−s**] [items]

DESCRIPTION

news is used to keep the user informed of current events. By convention, these events are described by files in the directory **/usr/news**.

When invoked without arguments, *news* prints the contents of all current files in **/usr/news**, most recent first, with each preceded by an appropriate header. *news* stores the "currency" time as the modification date of a file named **.news_time** in the user's home directory (the identity of this directory is determined by the environment variable **$HOME**); only files more recent than this currency time are considered "current."

−a option causes *news* to print all items, regardless of currency. In this case, the stored time is not changed.

−n option causes *news* to report the names of the current items without printing their contents, and without changing the stored time.

−s option causes *news* to report how many current items exist, without printing their names or contents, and without changing the stored time. It is useful to include such an invocation of *news* in one's **.profile** file, or in the system's **/etc/profile**.

All other arguments are assumed to be specific news items that are to be printed.

If a *delete* is typed during the printing of a news item, printing stops and the next item is started. Another *delete* within one second of the first causes the program to terminate.

FILES

/etc/profile
/usr/news/ *
$HOME/.news_time

SEE ALSO

profile(4), environ(5).

NAME

nice – run a command at low priority

SYNOPSIS

nice [–increment] command [arguments]

DESCRIPTION

nice executes *command* with a lower CPU scheduling priority. If the *increment* argument (in the range 1–19) is given, it is used; if not, an increment of 10 is assumed.

The super-user may run commands with priority higher than normal by using a negative increment, e.g., **– –10**.

SEE ALSO

nohup(1), nice(2).

DIAGNOSTICS

nice returns the exit status of the subject command.

BUGS

An *increment* larger than 19 is equivalent to 19.

This page is intentionally left blank

NAME

niocctl – control command for NIOC device

SYNOPSIS

/etc/niocctl device option [parameter]

DESCRIPTION

niocctl requests a NIOC to connect or disconnect a specified NIOC device or to give the status of a NIOC device.

The *device* keyword must be the name of a special file identifying the NIOC device in question.

The following options with associated parameters are available:

–c name requests the local NIOC to make a connection to a remote device on the net, which accepts a call with the specified name. A remote device will only accept the call if its channel is configured with a name-accept that matches the name parameter, and an id-accept that matches the *id* configured for the calling channel associated with the specified channel.

–d requests the NIOC to disconnect the connection on the specified device. It is not possible with this command to disconnect a permanent connection, a data controlled connection, or a PC connection.

–s requests the NIOC to return the status for the specified device. The status shows how the channel associated with the specified device is configured, (id, type, connection quality, etc.), and gives information whether the channel is connected, and in this case to whom.

–p networknumber hostnumber socketnumber
request the local NIOC to make a PC connection to a remote device on a net. This type of

507

connection does not use the name strategy described in the Supermax LAN manual. The parameters to the command must be the complete address, in hexadecimal form, of the device to which the connection is to be established.

The *networknumber* identifies the local area network when more nets are interconnected. It should be zero to indicate the local net, since internetting has not yet been implemented.

The *hostnumber* is the unique 12-digit hexadecimal number with which any system element connected to a network is supplied. Each NTC, NIOC, and PC network adapter board is supplied with such a unique number.

The *socketnumber* is a 4-digit hexadecimal number uniquely identified within a host. The socket is an object to which data can be delivered, and from which data can be transmitted.

This command should be used to connect Supermax PC disk and printer server to a PC. If this type of connection breaks down, (e.g. if the PC is switched off), the NIOC will keep trying to reconnect.

−q requests the NIOC to remove the PC connection.

EXAMPLES

```
niocctl /dev/tty15 -c gandalf
```

will request the NIOC associated with the device */dev/tty15* to make a connection from that device on the local area net which accepts the name *gandalf*.

The following example:

```
niocctl  /dev/tty24  -p  00000000  080075010006
0550
```

will request the NIOC associated with the device */dev/tty24* to make a PC connection from that device on the local area net 0x00000000 with the host number 0x080075010006, and the specific socket with the socket number 0x0550.

This page is intentionally left blank

NAME

nl − line numbering filter

SYNOPSIS

nl [−**h**type] [−**b**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:

−h*type*	Same as **−b***type* except for header. Default *type* for logical page header is **n** (no lines numbered).

a	number all lines
t	number lines with printable text only
n	no line numbering
p*string*	number only lines that contain the regular expression specified in *string*.

Default *type* for logical page body is **t** (text 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).

512

−d*xx* The delimiter characters specifying the start of a logical page section may be changed from the default characters (\:) to two user-specified characters. If only one character is entered, the second character remains the default character (:). No space should appear between the **−d** and the delimiter characters. To enter a backslash, use two backslashes.

EXAMPLE

The command:

```
nl -v10 -i10 -d!+ file1
```

will number file1 starting at line number 10 with an increment of ten. The logical page delimiters are !+.

SEE ALSO

pr(1).

This page is intentionally left blank

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. One can then issue:

```
nohup sh file
```

and the *nohup* applies to everything in *file*. If the shell procedure *file* is to be executed often, then the need to type *sh* can be eliminated by giving *file* execute permission. Add an ampersand and the contents of *file* are run in the background with interrupts also ignored (see *sh*(1)):

```
nohup file &
```

An example of what the contents of *file* could be is:

```
sort ofile > nfile
```

SEE ALSO

chmod(1), nice(1), sh(1), signal(2).

WARNINGS

In the case of the following command:

```
nohup command1; command2
```

nohup applies only to command1. The command:

```
nohup (command1; command2)
```

is syntactically incorrect.

NAME

oawk − pattern scanning and processing language

SYNOPSIS

oawk [**−F**c] [prog] [parameters] [files]

DESCRIPTION

oawk is a previous version of the *awk* command provided for backwards compatibility with older *awk* scripts.

oawk scans each input *file* for lines that match any of a set of patterns specified in *prog*. With each pattern in *prog* there can be an associated action that will be performed when a line of a *file* matches the pattern. The set of patterns may appear literally as *prog*, or in a file specified as **−f** *file*. The *prog* string should be enclosed in single quotes (′) to protect it from the shell.

parameters, in the form x=... y=... etc., may be passed to *oawk.*

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

An input line is made up of fields separated by white space. (This default can be changed by using FS; see below). The fields are denoted **$1**, **$2**, . . . ; **$0** refers to the entire line.

A pattern-action statement has the form:

```
pattern { action }
```

A missing action means print the line; a missing pattern always matches. An action is a sequence of statements. A statement can be one of the following:

```
if ( conditional ) statement [ else statement ]
while ( conditional ) statement
for ( expression ; conditional ; expression ) statement
break
continue
```

```
{ [ statement ] ... }
variable = expression
print [ expression-list ] [ >expression ]
printf format [ , expression-list ] [ >expression ]
next    # skip remaining patterns on this input line
exit    # skip the rest of the input
```

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

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

The built-in function *length* returns the length of its argument taken as a string, or of the whole line if no argument. There are also built-in functions *exp*, *log*, *sqrt*, and *int*. The last truncates its argument to an integer; *substr*(*s*, *m*, *n*) returns the *n*-character substring of *s* that begins at position *m*. The function *sprintf*(*fmt*, *expr*, *expr*, ...) formats the expressions according to the *printf*(3S) format given by *fmt* and returns the resulting string.

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

A relational expression is one of the following:

```
expression matchop regular-expression
expression relop expression
```

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**, new-line=**\n**, return=**\r**, tab=**\t**; others appear as 3-digit octal numbers.
–d	Interpret words in unsigned decimal.
–o	Interpret words in octal.
–s	Interpret 16-bit words in signed decimal.
–x	Interpret words in hex.

The *file* argument specifies which file is to be dumped. If no file argument is specified, the standard input is used.

The offset argument specifies the offset in the file where dumping is to commence. This argument is normally interpreted as octal bytes. If **.** is appended, the offset is interpreted in decimal. If **b** is appended, the offset is interpreted in blocks of 512 bytes. If the file argument is omitted, the offset argument must be preceded by **+**.

Dumping continues until end-of-file.

This page is intentionally left blank

NAME

opendir – preopen files and directories for faster access.

SYNOPSIS

/etc/opendir files ...

DESCRIPTION

opendir is used to preopen directories or files for faster access by other programs. During boot files and directories can be opened, saving time for other processes, that want to access that file. Potential directories to be opened by *opendir* are /tmp, /bin, /usr/tmp and /usr/bin.

NOTE

All files and directories are opened with a filebuffer equal to their sizes, thus consuming space from the operating systems dynamic data area (items area).

This page is intentionally left blank

NAME

openpart – maintain language in memory partition

SYNOPSIS

/etc/openpart languagefile language

DESCRIPTION

openpart is used to load language file into a named memory partition. This speeds up execution of programs using the language system, since the language file will not have to be loaded every time such a program is called. Time is only gained by *openpart* if the program called uses the language system and the required language file is loaded.

Memory partition can be loaded during boot by adding a script to the */etc/rc.d* directory.

A potential language file to be loaded by *openpart* is the language file **sysadm** used by *sysadm*(1).

The language file is held in memory as long as any program or the *openpart* daemon is using the named language partition. The memory partition is released simply by killing the daemon and waiting for all other programs using the memory partition to terminate.

EXAMPLE

openpart calls to load **sysadm** language file into memory:

```
openpart sysadm uk
```

NOTE

Some basic utilities uses the **sysadm** language file. The basic utilities present using the **sysadm** language file are the following:

diskformat,
dskback,
passwd,
streamdrv (bcpio, btar)

This page is intentionally left blank

NAME

pack, pcat, unpack – compress and expand files

SYNOPSIS

pack [–] [**–f**] name ...

pcat name ...

unpack name ...

DESCRIPTION

pack attempts to store the specified files in a compressed form. Wherever possible (and useful), each input file *name* is replaced by a packed file *name***.z** with the same access modes, access and modified dates, and owner as those of *name*. The **-f** option will force packing of *name*. This is useful for causing an entire directory to be packed even if some of the files will not benefit. If *pack* is successful, *name* will be removed. Packed files can be restored to their original form using *unpack* or *pcat*.

pack uses Huffman (minimum redundancy) codes on a byte-by-byte basis. If the – argument is used, an internal flag is set that causes the number of times each byte is used, its relative frequency, and the code for the byte to be printed on the standard output. Additional occurrences of – in place of *name* will cause the internal flag to be set and reset.

The amount of compression obtained depends on the size of the input file and the character frequency distribution. Because a decoding tree forms the first part of each **.z** file, it is usually not worthwhile to pack files smaller than three blocks, unless the character frequency distribution is very skewed, which may occur with printer plots or pictures.

Typically, text files are reduced to 60-75% of their original size. Load modules, which use a larger character set and have a more uniform distribution of characters, show little compression, the packed versions being about 90% of the original size.

pack returns a value that is the number of files that it failed to compress.

No packing will occur if:

> the file appears to be already packed;
> the file name has more than 12 characters;
> the file has links;
> the file is a directory;
> the file cannot be opened;
> no disk storage blocks will be saved by packing;
> a file called *name*.**z** already exists;
> the .**z** file cannot be created;
> an I/O error occurred during processing.

The last segment of the file name must contain no more than 12 characters to allow space for the appended **.z** extension. Directories cannot be compressed.

Pcat does for packed files what *cat*(1) does for ordinary files, except that *pcat* cannot be used as a filter. The specified files are unpacked and written to the standard output. Thus to view a packed file named *name*.**z** use:

```
        pcat name.z
```

or just:

```
        pcat name
```

To make an unpacked copy, say *nnn*, of a packed file named *name*.**z** (without destroying *name*.**z**) use the command:

```
        pcat name >nnn
```

Pcat returns the number of files it was unable to unpack. Failure may occur if:

> the file name (exclusive of the **.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

526

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).

This page is intentionally left blank

NAME

passmgmt – password files management

SYNOPSIS

passmgmt –a *options name*
passmgmt –m *options name*
passmgmt –d *name*

DESCRIPTION

The **passmgmt** command updates information in the password files. This command works with both **/etc/passwd** and **/etc/shadow**. If there is no **/etc/shadow**, the changes done by **passmgmt** will only go to **/etc/passwd**.

passmgmt –a adds an entry for user *name* to the password files. This command does not create any directory for the new user and the new login remains locked (with the string `*LK` in the password field) until the *passwd*(1) command is executed to set the password.

passmgmt –m modifies the entry for user *name* in the password files. The name field in the **/etc/shadow** entry and all the fields (except the password field) in the **/etc/passwd** entry can be modified by this command. Only fields entered on the command line will be modified.

passmgmt –d deletes the entry for user *name* from the password files. It will not remove any files that the user owns on the system; they must be removed manually.

The following options are available:

–c *comment*	A short description of the login. It is limited to a maximum of 128 characters and defaults to an empty field.
–h *homedir*	Home directory of *name*. It is limited to a maximum of 256 characters and defaults to **/usr/***name*.

−u *uid*	UID of the *name*. This number must range from 0 to the maximum non-negative value for the system. It defaults to the next available UID greater than 99. Without the **−o** option, it enforces the uniqueness of a UID.
−o	This option allows a UID to be non-unique. It is used only with the **−u** option.
−g *gid*	GID of the *name*. This number must range from 0 to the maximum non-negative value for the system. The default is 1.
−s *shell*	Login shell for *name*. It should be the full path-name of the program that will be executed when the user logs in. The maximum size of *shell* is 256 characters. The default is for this field to be empty and to be interpreted as **/bin/sh**.
−l *logname*	This option changes the *name* to *logname*. It is used only with the **−m** option.

The total size of each login entry is limited to a maximum of 511 bytes in each of the password files.

FILES

/etc/passwd
/etc/shadow
/etc/opasswd
/etc/oshadow

SEE ALSO

passwd(1), passwd(4).

DIAGNOSTICS

The **passmgmt** command exits with one of the following values:

0	SUCCESS.
1	Permission denied.
2	Invalid command syntax. Usage message of the **passmgmt** command will be displayed.
3	Invalid argument provided to option.
4	UID in use.
5	Inconsistent password files (e.g., *name* is in the **/etc/passwd** file and not in the **/etc/shadow** file, or vice versa).
6	Unexpected failure. Password files unchanged.
7	Unexpected failure. Password file(s) missing.
8	Password file(s) busy. Try again later.
9	*name* does not exist (if **−m** or **−d** is specified), already exists (if **−a** is specified), or *logname* already exists (if **−m**

NOTE

You cannot use a colon or <cr> as part of an argument because it will be interpreted as a field separator in the password file.

This page is intentionally left blank

529

NAME

passwd – change login password

SYNOPSIS

passwd [name]

DESCRIPTION

This command changes or installs a password associated with the login *name*.

Ordinary users may change only the password which corresponds to their login *name*.

passwd prompts ordinary users for their old password, if any. It then prompts for the new password twice. The first time the new password is entered *passwd* checks to see if the old password has "aged" sufficiently. Password "aging" is the amount of time (usually a certain number of days) that must elapse between password changes. If "aging" is insufficient the new password is rejected and *passwd* terminates; see *passwd*(4).

Assuming "aging" is sufficient, a check is made to insure that the new password meets construction requirements. When the new password is entered a second time, the two copies of the new password are compared. If the two copies are not identical the cycle of prompting for the new password is repeated for at most two more times.

Passwords must be constructed to meet the following requirements:

Each password must have at least six characters. Only the first eight characters are significant.

Each password must contain at least two alphabetic characters and at least one numeric or special character. In this case, "alphabetic" means upper and lower case letters.

Each password must differ from the user's login *name* and any reverse or circular shift of that login *name*. For comparison purposes, an upper case letter and its corresponding lower case letter are equivalent.

New passwords must differ from the old by at least three characters. For comparison purposes, an upper case letter and its corresponding lower case letter are equivalent.

One whose effective user ID is zero is called a super-user; see *id*(1), and *su*(1). Super-users may change any password; hence, *passwd* does not prompt super-users for the old password. Super-users are not forced to comply with password aging and password construction requirements. A super-user can create a null password by entering a carriage return in response to the prompt for a new password.

FILES

/etc/passwd

SEE ALSO

id(1), login(1), su(1), crypt(3C), passwd(4).

NAME

paste – merge same lines of several files or subsequent lines of one file

SYNOPSIS

paste file1 file2 ...
paste **−d**list file1 file2 ...
paste **−s** **[−d**list**]** file1 file2 ...

DESCRIPTION

In the first two forms, *paste* concatenates corresponding lines of the given input files *file1*, *file2*, etc. It treats each file as a column or columns of a table and pastes them together horizontally (parallel merging). If you will, it is the counterpart of *cat*(1) which concatenates vertically, i.e., one file after the other.

In the last form above, *paste* replaces the function of an older command with the same name by combining subsequent lines of the input file (serial merging). In all cases, lines are glued together with the *tab* character, or with characters from an optionally specified *list*.

Output is to the standard output, so it can be used as the start of a pipe, or as a filter, if – is used in place of a file name.

The meanings of the options are:

−d Without this option, the new-line characters of each but the last file (or last line in case of the **−s** option) are replaced by a *tab* character. This option allows replacing the *tab* character by one or more alternate characters (see below).

list One or more characters immediately following **−d** replace the default *tab* as the line concatenation character. The list is used circularly, i.e., when exhausted, it is reused. In parallel merging (i.e., no **−s** option), the lines from the last file are always terminated with a new-line character, not from the *list*. The list may contain the special escape sequences: **\n** (new-line),

531

\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 rather than one from each input file. Use *tab* for concatenation, unless a *list* is specified with **−d** option. Regardless of the *list*, the very last character of the file is forced to be a new-line.

− May be used in place of any file name, to read a line from the standard input. (There is no prompting).

EXAMPLES

ls | paste −d" " −	list directory in one column
ls | paste − − − −	list directory in four columns
paste −s −d"\t\n" file	combine pairs of lines into lines

SEE ALSO

cut(1), grep(1), pr(1).

DIAGNOSTICS

line too long	Output lines are restricted to 511 characters.
too many files	Except for **−s** option, no more than 12 input files may be specified.

NAME

perms – check or set file permissions

SYNOPSIS

perms [options]

DESCRIPTION

perms is used for ownership and mode control of files. It operates in three different modes. In "set" mode, *perms* sets the owner, group owner, and access modes for a list of files. In "check" mode, *perms* checks the owner, group owner, and access modes against a master list of files, flagging any discrepancies. Finally, in "make" mode, *perms* creates output for a specified list of files in a format suitable for subsequent runs of *perms* in "check" or "set" modes.

The following options may be selected:

–c (check mode) Check the owner, group owner, and access mode against list of files in */etc/permlist*. Discrepancies are written to standard output.

–m (make mode) For each file listed on standard input, write a line to standard output specifying the current owner, group owner, and access mode. This output is in a format suitable for later runs of perms with the **–s** and **c** options.

–s (set mode) Set the owner, group owner, and access mode for files specified in */etc/permlist*.

–f file Use *file* instead of */etc/permlist* for **–c** and **–s** options, and instead of standard input for **–m** option.

For **–c** and **–s** modes, each line of input takes the following form:

owner group-owner octal-mode file(s)

Fields may be separated by one or more tab characters. Lines that begin with # are ignored by perms. File name substitution can be used. A default set of permissions can be given for the files in a directory *dir* by first listing the permissions for *dir*/* followed by the individual exceptions.

533

EXAMPLES

Set permissions of files as listed in *filelist*:

```
# perms -s -f filelist
```

Generate permissions for /bin and /bin/* and write to /etc/permlist:

```
# perms -m > /etc/permlist
/bin
/bin/*
Ctrl-d
```

Check permissions of files specified in /etc/permlist:

```
# perms -c
```

BUGS

Specifying too many files on a single line in the input file can generate an "arg list too long" error message after file name substitution has been done by the shell. In that case, try splitting the offending specification into multiple lines.

534

FILES

```
/etc/permlist
```

NAME

pg – file perusal filter for CRTs

SYNOPSIS

pg [–*number*] [**–p** *string*] [**–cefns**] [+*linenumber*] [+/*pattern*/] [files...]

DESCRIPTION

The *pg* command is a filter which allows the examination of *files* one screenful at a time on a CRT. (The file name – and/or NULL arguments indicate that *pg* should read from the standard input.) Each screenful is followed by a prompt. If the user types a carriage return, another page is displayed; other possibilities are enumerated below.

This command is different from previous paginators in that it allows you to back up and review something that has already passed. The method for doing this is explained below.

In order to determine terminal attributes, *pg* scans the *terminfo*(4) data base for the terminal type specified by the environment variable **TERM**. If **TERM** is not defined, the terminal type **dumb** is assumed.

The command line options are:

–*number*	An integer specifying the size (in lines) of the window that *pg* is to use instead of the default. (On a terminal containing 24 lines, the default window size is 23).
–p *string*	Causes *pg* to use *string* as the prompt. If the prompt string contains a "%d", the first occurrence of "%d" in the prompt will be replaced by the current page number when the prompt is issued. The default prompt string is ":".
–c	Home the cursor and clear the screen before displaying each page. This option is ignored if **clear_screen** is not defined for this terminal type in the *terminfo*(4) data base.

535

−e	Causes *pg not* to pause at the end of each file.
−f	Normally, *pg* splits lines longer than the screen width, but some sequences of characters in the text being displayed (e.g., escape sequences for underlining) generate undesirable results. The *−f* option inhibits *pg* from splitting lines.
−n	Normally, commands must be terminated by a <*newline*> character. This option causes an automatic end of command as soon as a command letter is entered.
−s	Causes *pg* to print all messages and prompts in standout mode (usually inverse video).
+*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 which cause further perusal normally take a preceding *address*, an optionally signed number indicating the point from which further text should be displayed. This *address* is interpreted in either pages or lines depending on the 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 as follows:

(+1)<*newline*>	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.

536

	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.
(+1) **f**	Skip page.

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.

537

The following commands are available for searching for text patterns in the text. The regular expressions described in *ed*(1) are available. They must always be terminated by a <*newline*>, even if the −*n* option is specified.

i/*pattern*/	Search forward for the *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 which will not properly handle the ?.

After searching, *pg* will normally display the line found at the top of the screen. This can be modified by appending **m** or **b** to the search command to leave the line found in the middle or at the bottom of the window from now on. The suffix **t** can be used to restore the original situation.

The user of *pg* can modify the environment of perusal with the following commands:

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 hit the quit key (normally control-\) or the interrupt (break) key. This causes *pg* to stop sending output, and display the prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

If the standard output is not a terminal, then *pg* acts just like *cat*(1), except that a header is printed before each file (if there is more than one).

EXAMPLE

A sample usage of *pg* in reading system news would be

```
news | pg -p "(Page %d):"
```

NOTES

While waiting for terminal input, *pg* responds to **BREAK**, **DEL**, and ^ by terminating execution. Between prompts, however, these signals interrupt *pg*'s current task and place the user in prompt mode. These should be used with caution when input is being read from a pipe, since an interrupt is likely to terminate the other commands in the pipeline.

Users of Berkeley's *more* will find that the z and f commands are available, and that the terminal /, ^, or ? may be omitted from the searching commands.

FILES

/usr/lib/terminfo/?/ *	terminal information database
/tmp/pg*	temporary file when input is from a pipe

SEE ALSO

ed(1), grep(1), terminfo(4).

BUGS

If terminal tabs are not set every eight positions, undesirable results may occur.

When using *pg* as a filter with another command that changes the terminal I/O options terminal settings may not be restored correctly.

This page is intentionally left blank

NAME

powerdown – stop all processes and turn off the power

SYNOPSIS

powerdown [**–y** | **–Y**]

DESCRIPTION

This command brings the system to a state where nothing is running and then turns off the power.

By default, the user is asked questions that control how much warning the other users are given. The options:

–y prevents the questions from being asked and just gives the warning messages. There is a 60 second pause between the warning messages. Note that pressing the standby button on the side of the cabinet will accomplish the same thing.

–Y is the same as **–y** except it has no pause between messages. It is the fastest way to bring the system down.

The identical function is also available under the *sysadm* command:

sysadm powerdown

Password control can be instituted on this command. See *sysadm*(1), **admpasswd** sub-command.

541

EXAMPLES

```
some-long-running-command;  powerdown –y
```

The first command is run to completion and then the machine turns off. This is useful for, say, formatting a document to the printer at the end of a day.

FILES

/etc/shutdown – invoked by powerdown

SEE ALSO

shutdown(1M), sysadm(1).

This page is intentionally left blank

NAME

pr − print files

SYNOPSIS

pr [[**−column**] [**−wwidth**] [**−a**]] [**−eck**] [**−ick**] [**−drtfp**] [**+**page] [**−nck**] [**−ooffset**] [**−llength**] [**−sseparator**] [**−hheader**] [file ...]

pr [[**−m**] [**−wwidth**]] [**−eck**] [**−ick**] [**−drtfp**] [**+**page] [**−nck**] [**−ooffset**] [**−llength**] [**−sseparator**] [**−hheader**] 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**.

543

Command line options are

+*page*	Begin printing with page numbered *page* (default is 1).
−*column*	Print *column* columns of output (default is 1). Output appears as if **−e** and **−i** are turned on for multi-column output. May not use with **−m**.
−a	Print multi-column output across the page one line per column. *columns* must be greater than one. If a line is too long to fit in a column, it is truncated.
−m	Merge and print all files simultaneously, one per column. The maximum number of files that may be specified is eight. If a line is too long to fit in a column, it is truncated. May not use with −*column*.
−d	Double-space the output. Blank lines that result from double-spacing are dropped when they occur at the top of a page.
−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).

544

−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 line-feeds). Pause before beginning the first page if the standard output is associated with a terminal.

−r Print no diagnostic reports on files that will not open.

−t Print neither the five-line identifying header nor the five-line trailer normally supplied for each page. Quit printing after the last line of each file without spacing to the end of the page. Use of **−t** overrides the **−h** option.

−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 double-spaced, three-column listing headed by "file list":

```
pr −3dh "file list" file1 file2
```

Copy **file1** to **file2**, expanding tabs to columns 10, 19, 28, 37, etc.:

```
pr −e9 −t <file1 >file2
```

Print **file1** and **file2** simultaneously in a two-column listing with no header or trailer where both columns have line numbers:

```
pr −t −n file1 | pr −t −m −n file2 −
```

FILES

/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).

This page is intentionally left blank

NAME

prgnote – print DDE special program usage note

SYNOPSIS

prgnote files

DESCRIPTION

prgnote(1) prints DDE special information appended to a load-module or to an interpreted pascal program.

prgnote(1) prints the usage note for the specifed files.

If only one filename (not containing a '/') is specified as argument to *prgnote(1)* the environment variable PATH is used by *prgnote(1)* to locate the file.

SEE ALSO

prgvers(1)

This page is intentionally left blank

NAME
prgvers – print DDE special program information

SYNOPSIS
prgvers files

DESCRIPTION
prgvers(1) prints DDE special information appended to a load-module or to an interpreted pascal program.

prgvers(1) prints programversion, date and department identification for the specifed files.

If only one filename (not containing a '/') is specified as argument to *prgvers(1)* the environment variable PATH is used by *prgvers(1)* to locate the file.

SEE ALSO
prgnote(1)

This page is intentionally left blank

NAME

prod – start a command as a new process group.

SYNOPSIS

prod command [arguments]

DESCRIPTION

prod executes *command* as a new process group leader.

EXAMPLE

It is frequently desirable to apply *prod* to servers and other programs that should be run in the background. These can be started during boot using the *prod* command. The command will no longer be associated to the terminal from which it was started, thus been immune to interupts and quits from that terminal.

NOTE

Unlike nohup(1) *prod* does not automatically redirect output.

SEE ALSO

nohup(1)

This page is intentionally left blank

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*.

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 **e**very process now running.
–d	Print information about all processes except process group leaders.
–a	Print information about **a**ll processes most frequently requested: all those except process group leaders and processes not associated with a terminal.
–f	Generate a **f**ull listing. (See below for significance of columns in a full listing.)
–l	Generate a **l**ong listing. (See below.)
–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*.

555

−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.)

The column headings and the meaning of the columns in a *ps* listing are given below; the letters **f** and **l** indicate the option (**f**ull or **l**ong, 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:

02	Active
04	Running
08	Externally suspended
10	Internally suspended
40	Being aborted
80	A signal is waiting

S (l) The state of the process:

A	Active
R	Running
S	Externally suspended
I	Internally suspended

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. (Always 0 on a Supermax Computer.)
PRI	(l)	The priority of the process (higher numbers mean lower priority).
NI	(l)	Nice value, used in priority computation.
ADDR	(l)	The Address Space Number followed by 3 zeroes followed by the MCU number for the process.
SZ	(l)	The size (in 2048-byte pages) of the process's image in main memory.
WCHAN	(l)	The address of a global event for which the process is sleeping (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>**.

557

FILES

/dev	terminal ("tty") names searcher files
/dev/kmem*	kernel memory
/etc/passwd	UID information supplier
/etc/ps_data	internal data structure
/bin/ps20	started by *ps* when running on an MC68020 processor.

SEE ALSO

getty(1M), kill(1), nice(1).

WARNING

Things can change while *ps* is running; the snap-shot it gives is only true for a split-second, and it may not be accurate by the time you see it. Some data printed for defunct processes is irrelevant.

If no *termlist*, *proclist*, *uidlist*, or *grplist* is specified, *ps* checks *stdin*, *stdout*, and *stderr* in that order, looking for the controlling terminal and will attempt to report on processes associated with the controlling terminal. In this situation, if *stdin*, *stdout*, and *stderr* are all redirected, *ps* will not find a controlling terminal, so there will be no report.

NAME

pwck, grpck – password/group file checkers

SYNOPSIS

/etc/pwck [file]
/etc/grpck [file]

DESCRIPTION

pwck scans the password file and notes any inconsistencies. The checks include validation of the number of fields, login name, user ID, group ID, and whether the login directory and the program-to-use-as-Shell exist. The default password file is **/etc/passwd**.

grpck verifies all entries in the group file. This verification includes a check of the number of fields, group name, group ID, and whether all login names appear in the password file. The default group file is **/etc/group**.

FILES

/etc/group
/etc/passwd

SEE ALSO

group(4), passwd(4).

DIAGNOSTICS

Group entries in **/etc/group** with no login names are flagged.

This page is intentionally left blank

NAME

pwd – working directory name

SYNOPSIS

pwd

DESCRIPTION

pwd prints the path name of the working (current) directory.

SEE ALSO

cd(1).

DIAGNOSTICS

“Cannot open ..” and “Read error in ..” indicate possible file system trouble and should be referred to a UNIX system administrator.

This page is intentionally left blank

NAME

rc0 – run commands performed to stop the operating system

SYNOPSIS

/etc/rc0

DESCRIPTION

This file is executed at each system state change that needs to have the system in an inactive state. It is responsible for those actions that bring the system to a quiescent state, traditionally called "shutdown".

There are three system states that require this procedure. They are state 0 (the system halt state), state 5 (the firmware state), and state 6 (the reboot state). Whenever a change to one of these states occurs, the */etc/rc0* procedure is run. The entry in **/etc/inittab** might read:

```
s0:056:wait:/etc/rc0 >/dev/console 2>&1
                              </dev/console
```

Some of the actions performed by */etc/rc0* are carried out by files in the directory **/etc/shutdown.d**. and files beginning with **K** in **/etc/rc0.d**. These files are executed in ascii order (see FILES below for more information), terminating some system service. The combination of commands in */etc/rc0* and files in **/etc/shutdown.d** and **/etc/rc0.d** determines how the system is shut down.

The recommended sequence for */etc/rc0* is:

Stop System Services and Daemons.

Various system services (such as 3BNET Local Area Network or LP Spooler) are gracefully terminated.

When new services are added that should be terminated when the system is shut down, the appropriate files are installed in **/etc/shutdown.d** and **/etc/rc0.d**.

Terminate Processes

SIGTERM signals are sent to all running processes by *killall*(1M). Processes stop themselves cleanly if sent SIGTERM.

Kill Processes

SIGKILL signals are sent to all remaining processes; no process can resist SIGKILL.

At this point the only processes left are those associated with */etc/rc0* and processes 0 and 1, which are special to the operating system.

Unmount All File Systems

Only the root file system (/) remains mounted.

Depending on which system state the systems end up in (0, 5, or 6), the entries in **/etc/inittab** will direct what happens next. If the **/etc/inittab** has not defined any other actions to be performed as in the case of system state 0, then the operating system will have nothing to do. It should not be possible to get the system's attention. The only thing that can be done is to turn off the power or possibly get the attention of a firmware monitor. The command can be used only by the super-user.

564

FILES

The execution by **/bin/sh** of any files in **/etc/shutdown.d** occurs in ascii sort-sequence order. See rc2(1M) for more information.

SEE ALSO

killall(1M), rc2(1M), shutdown(1M).

NAME

rc2 – run commands performed for multi-user environment

SYNOPSIS

/etc/rc2

DESCRIPTION

This file is executed via an entry in **/etc/inittab** and is responsible for those initializations that bring the system to a ready-to-use state, traditionally state 2, called the "multi-user" state.

The actions performed by **/etc/rc2** are found in files in the directory **/etc/rc.d**. These files are executed by **/bin/sh** in ascii sort – sequence order (see FILES for more information). When functions are added that need to be initialized when the system goes multi-user, an appropriate file should be added in **/etc/rc.d**.

The functions done by **/etc/rc2** command and associated **/etc/rc.d** files include:

Setting and exporting the TIMEZONE variable.

Setting-up and mounting the user (**/usr**) file system.

Cleaning up (remaking) the **/tmp** and **/usr/tmp** directories.

Loading the network interface and ports cards with program data and starting the associated processes.

Starting the *cron* daemon by executing **/etc/cron**.

Cleaning up (deleting) uucp locks status, and temporary files in the **/usr/spool/uucp** directory.

Other functions can be added, as required, to support the addition of hardware and software features.

EXAMPLES

The following are prototypical files found in **/etc/rc.d**.

```
MOUNTFILESYS
          #   Set up and mount file systems

          cd /
          /etc/mountall /etc/fstab
RMTMPFILES
          # clean up /tmp
          rm -rf /tmp
          mkdir /tmp
          chmod 777 /tmp
          chgrp sys /tmp
          chown sys /tmp
uucp
          #   clean-up uucp locks, status, and temporary files

          rm -rf /usr/spool/locks/*
```

The file **/etc/TIMEZONE** is included early in */etc/rc2*, thus establishing the default time zone for all commands that follow.

FILES

Here are some hints about files in **/etc/rc.d**:

The order in which files are executed is important. Since they are executed in ascii sort-sequence order, using the first character of the file name as a sequence indicator will help keep the proper order. Thus, files starting with the following characters would be:

[0-9]	very early
[A-Z]	early
[a-n]	later
[o-z]	last

Files in **/etc/rc.d** might be named:

```
3.mountfs
B.uucp
c.cron
r.lpr
```

Files in **/etc/rc.d** that begin with a dot (.) will not be executed. This feature can be used to hide files that are not to be executed for the time being without removing them.

SEE ALSO

shutdown(1M).

This page is intentionally left blank

NAME

rm, rmdir – remove files or directories

SYNOPSIS

rm [**–fi**] file ...

rm –r [**–fi**] dir ... [file ...]

rmdir [**–p**] [**–s**] dir ...

DESCRIPTION

rm removes the entries for one or more files from a directory. If an entry was the last link to the file, the file is destroyed. Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

If a file has no write permission and the standard input is a terminal, the full set of permissions (in octal) for the file are printed followed by a question mark. This is a prompt for confirmation. If the answer begins with **y** (for yes), the file is deleted, otherwise the file remains.

Note that if the standard input is not a terminal, the command will operate as if the **–f** option is in effect.

rmdir removes the named directories, which must be empty.

Three options apply to *rm*.

–f This option causes the removal of all files (whether write-protected or not) in a directory without prompting the user. In a write-protected directory, however, files are never removed (whatever their permissions are), but no messages are displayed.

If the removal of a write-protected directory was attempted, this option cannot suppress an error message.

–r This option causes the recursive removal of any directories and subdirectories in the argument list. The directory will be emptied of files and removed. Note that the user is normally prompted for removal of any write-protected files which the directory contains. The write-protected files are removed without prompting, however, if the **−f** option is used, or if the standard input is not a terminal and the **−i** option is not used.

If the removal of a non-empty, write-protected directory was attempted, the command will always fail (even if the **−f** option is used), resulting in an error message.

−i With this option in effect, *rm* asks if each file should be deleted and, with the **−r** option if each directory should be examined.

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. Note that it is forbidden to remove the files **"."** and **".."** to avoid the consequences of inadvertently doing something like:

```
rm −r    .*
```

SEE ALSO

unlink(2).

NAME
: rmpkg – remove a software package

SYNOPSIS
: **rmpkg** [device]

DESCRIPTION
: *rmpkg(1)* removes a software package earlier installed on the system with the *newpkg(1)* utility. As for *newpkg* the device may either be a floppy or a streamer. If no device is specified **/dev/flop** is assumed.

SEE ALSO
: newpkg(1)

This page is intentionally left blank

NAME

rsetsioc – initialize terminal or printer

SYNOPSIS

rsetsioc [specialfiles]

DESCRIPTION

rsetsioc sends an initialization sequence to a printer or a terminal. More specifically, it outputs the control sequence to the SIOC, which (assuming a proper configuration table in the SIOC) will result in an initialization sequence being sent to the terminal or printer. Some, but not all, terminals or printers require such a sequence to be sent to them before they will operate properly. Furthermore all attribute values, such as inverse video, underlining etc. will be turned off, so the *rsetsioc* program can be used for resetting terminals that have accidently been set to improper attribute values.

If no specialfile is given as argument to *rsetsioc* the current output device is reset.

BUGS

The *rsetsioc* program may occasionally hang when trying to reset a terminal that has an outstanding read.

This page is intentionally left blank