TM

ICON/UXB Operating System Reference Manual

Volume 1B

ICON
INTERNATIONAL

P.O. Box 340
Orem, Utah 84059
(801) 225-6888

OPERATING SYSTEM REFERENCE MANUAL

ICON/UXB

Supplementary Documents

Volume 1B

This manual reflects system enhancements made at Berkeley and sponsored in part by NSF Grants MCS-7807291, MCS-8005144, and MCS-74-07644-A04; DOE Contract DE-AT03-76SF00034 and Project Agreement DE-AS03-79ER10358; and by Defense Advanced Research Projects Agency (DoD) ARPA Order No. 4031, Monitored by Naval Electronics Systems Command under Contract No. N00039-80-K-0649.

This manual was prepared by the Documentation Group of Icon International, Inc., P.O. Box 340, Orem, UT 84057-0340. A form for reader's comments has been provided at the back of this publication. Comments are welcomed and may be sent to the above address. Users who respond will be entitled to free updates of this manual for one year.

Revision B

Order Number 172-022-002 (Manual Assembly)
Order Number 171-070-002 (Pages Only)

Printed in the U.S.A.

Change Record Page

Manual Part No. 172-022-002

Date	Revision	Description	Pages Affected
Jan. 1987	A	Initial production release	All
Nov. 1987	B	Incorporate additions of commands included in Releases 2.16, 3.0, and 3.1 of the ICON/UXB Operating System and separate Volume 1 into two separate binders	Main cover page, titlepage, copyright page, Table of Contents, Permuted Index, Introduction, addition of and changes to the following manual pages: dstrules(5), printcap(5), rcsfile(5), dump(8), ping(8), sccstorcs(8), slattach(8c), talkd(8c)

TABLE OF CONTENTS

3. C Library Subroutines

SECTION I

ICON/UXB OPERATING SYSTEM PERMUTED INDEX

PERMUTED INDEX

	@: arithmetic on shell variables.	csh(1)
lib2648: subroutines for the HP	2648 graphics terminal.	lib2648(3X)
diff3:	3-way differential file comparison.	diff3(1)
sendbug: mail a system bug report to	4bsd-bugs.	sendbug(1)
	aardvark: yet another exploration game.	aardvark(6)
	abort: generate a fault.	abort(3)
	abort: terminate abruptly with memory image.	abort(3F)
abort: terminate	abruptly with memory image.	abort(3F)
	abs: integer absolute value.	abs(3)
abs: integer	absolute value.	abs(3)
fabs, floor, ceil:	absolute value, floor, ceiling functions.	floor(3M)
	ac: login accounting.	ac(8)
accept:	accept a connection on a socket.	accept(2)
	accept: accept a connection on a socket.	accept(2)
	access: determine accessability of a file.	access(3F)
	access: determine accessibility of file.	access(2)
getgroups: get group	access list.	getgroups(2)
initgroups: initialize group	access list.	initgroups(3X)
setgroups: set group	access list.	setgroups(2)
access: determine	accessability of a file.	access(3F)
access: determine	accessibility of file.	access(2)
ac: login	accounting.	ac(8)
sa, accton: system	accounting.	sa(8)
acct: execution	accounting file.	acct(5)
pac: printer/plotter	accounting information.	pac(8)
acct: turn	accounting on or off.	acct(2)
	acct: execution accounting file.	acct(5)
	acct: turn accounting on or off.	acct(2)
sa,	accton: system accounting.	sa(8)
sin, cos, tan, asin,	acos, atan, atan2: trigonometric functions.	sin(3M)
signal: change the	action for a signal.	signal(3F)
fortune: print a random, hopefully interesting,	adage.	fortune(6)
	adb: debugger.	adb(1)
	addbib: create or extend bibliographic database.	addbib(1)
adduser: procedure for	adding new users.	adduser(8)
swapon: specify	additional device for paging and swapping.	swapon(8)
inet_makeaddr, inet_lnaof, inet_netof: Internet	address manipulation routines. /inet_ntoa,	inet(3n)
loc: return the	address of an object.	loc(3F)
mailaddr: mail	addressing description.	mailaddr(7)
	adduser: procedure for adding new users.	adduser(8)
	adventure: an exploration game.	adventure(6)
battlestar: a tropical	adventure game.	battlestar(6)
flock: apply or remove an	advisory lock on an open file.	flock(2)
yes: be repetitively	affirmative.	yes(1)
basename: strip filename	affixes.	basename(1)
learn: computer	aided instruction about UNIX.	learn(1)
	alarm: execute a subroutine after a specified time.	alarm(3F)
	alarm: schedule signal after specified time.	alarm(3C)
	alias: shell macros.	csh(1)
unalias: remove	aliases.	csh(1)
	aliases: aliases file for sendmail.	aliases(5)
which: locate a program file including	aliases and paths (*csh* only).	which(1)
newaliases: rebuild the data base for the mail	aliases file.	newaliases(1)
aliases:	aliases file for sendmail.	aliases(5)
valloc:	aligned memory allocator.	valloc(3)
malloc, free, realloc, calloc,	alloca: memory allocator.	malloc(3)
malloc, free, realloc, calloc, alloca: memory	allocator.	malloc(3)
valloc: aligned memory	allocator.	valloc(3)
eyacc: modified yacc	allowing much improved error recovery.	eyacc(1)
limit:	alter per-process resource limitations.	csh(1)
renice:	alter priority of running processes.	renice(8)
else:	alternative commands.	csh(1)
lex: generator of lexical	analysis programs.	lex(1)
error:	analyze and disperse compiler error messages.	error(1)
style:	analyze surface characteristics of a document.	style(1)
sigstack: set	and/or get signal stack context.	sigstack(2)
worms:	animate worms on a display terminal.	worms(6)
rain:	animated raindrops display.	rain(6)
bcd: convert to	antique media.	bcd(6)
apply:	apply a command to a set of arguments.	apply(1)
	apply: apply a command to a set of arguments.	apply(1)

flock:	apply or remove an advisory lock on an open file.	flock(2)
	apropos: locate commands by keyword lookup.	apropos(1)
	ar: archive and library maintainer.	ar(1)
	ar: archive (library) file format.	ar(5)
number: convert	Arabic numerals to English.	number(6)
bc:	arbitrary-precision arithmetic language.	bc(1)
graphics/ plot: openpl, erase, label, line, circle,	arc, move, cont, point, linemod, space, closepl:	plot(3X)
ar:	archive and library maintainer.	ar(1)
tar: tape	archive file format.	tar(5)
ar:	archive (library) file format.	ar(5)
tar: tape	archiver.	tar(1)
cpio: copy file	archives in and out.	cpio(1)
ranlib: convert	archives to random libraries.	ranlib(1)
glob: filename expand	argument list.	csh(1)
shift: manipulate	argument list.	csh(1)
varargs: variable	argument list.	varargs(3)
apply: apply a command to a set of	arguments.	apply(1)
echo: echo	arguments.	csh(1)
echo: echo	arguments.	echo(1)
getarg, iargc: return command line	arguments.	getarg(3F)
expr: evaluate	arguments as an expression.	expr(1)
traper: trap	arithmetic errors.	traper(3F)
bc: arbitrary-precision	arithmetic language.	bc(1)
@:	arithmetic on shell variables.	csh(1)
	arithmetic: provide drill in number facts.	arithmetic(6)
biff: be notified if mail	arrives and who it is from.	biff(1)
expr: evaluate arguments	as an expression.	expr(1)
	as: M68020 assembler.	as(1)
slattach: attach serial lines	as network interfaces.	slattach(8C)
gmtime, asctime, timezone: convert date and time to	ASCII. ctime, localtime,	ctime(3)
ascii: map of	ASCII character set.	ascii(7)
od: octal, decimal, hex,	ascii dump.	od(1)
	ascii: map of ASCII character set.	ascii(7)
fdate: return date and time in an	ASCII string.	fdate(3F)
atof, atoi, atol: convert	ASCII to numbers.	atof(3)
ctime, localtime, gmtime,	asctime, timezone: convert date and time to ASCII.	ctime(3)
sin, cos, tan,	asin, acos, atan, atan2: trigonometric functions.	sin(3M)
as: M68020	assembler.	as(1)
a.out:	assembler and link editor output.	a.out(5)
	assert: program verification.	assert(3X)
setbuf, setbuffer, setlinebuf:	assign buffering to a stream.	setbuf(3S)
shutdown: close down the system	at a given time.	shutdown(8)
at: execute commands	at a later time.	at(1)
	at: execute commands at a later time.	at(1)
nice, nohup: run a command	at low priority (*sh* only).	nice(1)
sin, cos, tan, asin, acos,	atan, atan2: trigonometric functions.	sin(3M)
sin, cos, tan, asin, acos, atan,	atan2: trigonometric functions.	sin(3M)
	atof, atoi, atol: convert ASCII to numbers.	atof(3)
atof,	atoi, atol: convert ASCII to numbers.	atof(3)
atof, atoi,	atol: convert ASCII to numbers.	atof(3)
interrupt. sigpause:	atomically release blocked signals and wait for	sigpause(2)
slattach:	attach serial lines as network interfaces.	slattach(8C)
rcs: change RCS file	attributes.	rcs(1)
bugfiler: file bug reports in folders	automatically.	bugfiler(8)
rc: command script for	auto-reboot and daemons.	rc(8)
wait:	await completion of process.	wait(1)
	awk: pattern scanning and processing language.	awk(1)
	backgammon: the game.	backgammon(6)
bg: place job in	background.	csh(1)
wait: wait for	background processes to complete.	csh(1)
banner: print large	banner on printer.	banner(6)
	banner: print large banner on printer.	banner(6)
gettytab: terminal configuration data	base.	gettytab(5)
hosts: host name data	base.	hosts(5)
networks: network name data	base.	networks(5)
phones: remote host phone number data	base.	phones(5)
printcap: printer capability data	base.	printcap(5)
protocols: protocol name data	base.	protocols(5)
services: service name data	base.	services(5)
termcap: terminal capability data	base.	termcap(5)
vgrindefs: vgrind's language definition data	base.	vgrindefs(5)
newaliases: rebuild the data	base for the mail aliases file.	newaliases(1)
ttytype: data	base of terminal types by port.	ttytype(5)
fetch, store, delete, firstkey, nextkey: data	base subroutines. dbminit,	dbm(3X)
vi: screen oriented (visual) display editor	based on ex.	vi(1)

	basename: strip filename affixes.	basename(1)
	battlestar: a tropical adventure game.	battlestar(6)
	bc: arbitrary-precision arithmetic language.	bc(1)
	bcd: convert to antique media.	bcd(6)
bcopy,	bcmp, bzero, ffs: bit and byte string operations.	bstring(3)
operations.	bcopy, bcmp, bzero, ffs: bit and byte string	bstring(3)
cb: C program	beautifier.	cb(1)
j0, j1, jn, y0, y1, yn:	bessel functions.	j0(3M)
	bessel functions: of two kinds for integer orders.	bessel(3F)
changing/ random, srandom, initstate, setstate:	better random number generator; routines for	random(3)
	bg: place job in background.	csh(1)
addbib: create or extend	bibliographic database.	addbib(1)
roffbib: run off	bibliographic database.	roffbib(1)
sortbib: sort	bibliographic database.	sortbib(1)
index for a bibliography, find references in a	bibliography. indxbib, lookbib: build inverted	lookbib(1)
indxbib, lookbib: build inverted index for a	bibliography, find references in a bibliography.	lookbib(1)
from.	biff: be notified if mail arrives and who it is	biff(1)
comsat:	biff server.	comsat(8C)
install: install	binaries.	install(1)
whereis: locate source,	binary, and or manual for program.	whereis(1)
find the printable strings in a object, or other	binary, file. strings:	strings(1)
uuencode,uudecode: encode/decode a	binary file for transmission via mail.	uuencode(1C)
fread, fwrite: buffered	binary input/output.	fread(3S)
bind:	bind a name to a socket.	bind(2)
	bind: bind a name to a socket.	bind(2)
	binmail: send or receive mail among users.	binmail(1)
	binstl: program to install bootloader on disk.	binstl(8)
bcopy, bcmp, bzero, ffs:	bit and byte string operations.	bstring(3)
functions.	bit: and, or, xor, not, rshift, lshift bitwise	bit(3F)
bit: and, or, xor, not, rshift, lshift	bitwise functions.	bit(3F)
	bload: program to load standalone programs.	bload(8)
sync: update the super	block.	sync(8)
update: periodically update the super	block.	update(8)
sigblock:	block signals.	sigblock(2)
sigpause: atomically release	blocked signals and wait for interrupt.	sigpause(2)
sum: sum and count	blocks in a file.	sum(1)
boggle: play the game of	boggle.	boggle(6)
	boggle: play the game of boggle.	boggle(6)
ching: the	book of changes and other cookies.	ching(6)
binstl: program to install	bootloader on disk.	binstl(8)
reboot: UNIX	bootstrapping procedures.	reboot(8)
mille: play Mille	Bournes.	mille(6)
switch: multi-way command	branch.	csh(1)
login,/ sh, for, case, if, while, :, .,	break, continue, cd, eval, exec, exit, export,	sh(1)
	break: exit while/foreach loop.	csh(1)
	breaksw: exit from switch.	csh(1)
fg:	bring job into foreground.	csh(1)
	brk, sbrk: change data segment size.	brk(2)
fread, fwrite:	buffered binary input/output.	fread(3S)
stdio: standard	buffered input/output package.	intro(3S)
setbuf, setbuffer, setlinebuf: assign	buffering to a stream.	setbuf(3S)
generate a dump of the operating system's profile	buffers. kgmon:	kgmon(8)
jove_recover - recover JOVE	buffers after a system/editor crash.	jove_recover(1)
sendbug: mail a system	bug report to 4bsd-bugs.	sendbug(1)
bugfiler: file	bug reports in folders automatically.	bugfiler(8)
automatically.	bugfiler: file bug reports in folders	bugfiler(8)
references in a bibliography. indxbib, lookbib:	build inverted index for a bibliography, find	lookbib(1)
sccstorcs:	build RCS file from SCCS file.	sccstorcs(1)
sccstorcs:	build RCS file from SCCS file.	sccstorcs(8)
mknod:	build special file.	mknod(8)
ntohs: convert values between host and network	byte order. htonl, htons, ntohl,	byteorder(3n)
bcopy, bcmp, bzero, ffs: bit and	byte string operations.	bstring(3)
swab: swap	bytes.	swab(3)
bcopy, bcmp,	bzero, ffs: bit and byte string operations.	bstring(3)
cc:	C compiler.	cc(1)
prec:	C precedence chart.	prec(7)
cb:	C program beautifier.	cb(1)
indent: indent and format	C program source.	indent(1)
lint: a	C program verifier.	lint(1)
xstr: extract strings from	C programs to implement shared strings.	xstr(1)
mkstr: create an error message file by massaging	C source.	mkstr(1)
hypot,	cabs: Euclidean distance.	hypot(3M)
	cal: print calendar.	cal(1)
dc: desk	calculator.	dc(1)
cal: print	calendar.	cal(1)

	calendar: reminder service.	calendar(1)
syscall: indirect system	call.	syscall(2)
gprof: display	call graph profile data.	gprof(1)
getuid, getgid: get user or group ID of the	caller.	getuid(3F)
malloc, free, realloc,	calloc, alloca: memory allocator.	malloc(3)
intro: introduction to system	calls and error numbers.	intro(2)
canfield, cfscores: the solitaire card game	canfield.	canfield(6)
canfield.	canfield, cfscores: the solitaire card game	canfield(6)
printcap: printer	capability data base.	printcap(5)
termcap: terminal	capability data base.	termcap(5)
canfield, cfscores: the solitaire	card game canfield.	canfield(6)
cribbage: the	card game cribbage.	cribbage(6)
cd, eval, exec, exit, export, login,/ sh, for,	case, if, while, : , . , break, continue,	sh(1)
	case: selector in switch.	csh(1)
	cat: catenate and print.	cat(1)
catman: create the	cat files for the manual.	catman(8)
uncompact, ccat: compress and uncompress files, and	cat them. compact,	compact(1)
default:	catchall clause in switch.	csh(1)
cat:	catenate and print.	cat(1)
	catman: create the cat files for the manual.	catman(8)
	cb: C program beautifier.	cb(1)
	cc: C compiler.	cc(1)
compact, uncompact,	ccat: compress and uncompress files, and cat them.	compact(1)
	cd: change directory.	csh(1)
	cd: change working directory.	cd(1)
case, if, while, : , . , break, continue,	cd, eval, exec, exit, export, login, read,/ /for,	sh(1)
fabs, floor,	ceil: absolute value, floor, *ceiling* functions.	floor(3M)
fabs, floor, ceil: absolute value, floor,	ceiling functions.	floor(3M)
canfield,	cfscores: the solitaire card game canfield.	canfield(6)
chdir:	change current working directory.	chdir(2)
brk, sbrk:	change data segment size.	brk(2)
chdir:	change default directory.	chdir(3F)
chsh:	change default login shell.	chsh(1)
cd:	change directory.	csh(1)
chdir:	change directory.	csh(1)
ioinit:	change f77 I/O initialization.	ioinit(3F)
chfn:	change finger entry.	chfn(1)
chgrp:	change group.	chgrp(1)
passwd:	change login password.	passwd(1)
chmod:	change mode.	chmod(1)
chmod:	change mode of a file.	chmod(3F)
chmod:	change mode of file.	chmod(2)
umask:	change or display file creation mask.	csh(1)
chown:	change owner.	chown(8)
chown:	change owner and group of a file.	chown(2)
rcs:	change RCS file attributes.	rcs(1)
chroot:	change root directory.	chroot(2)
signal:	change the action for a signal.	signal(3F)
rename:	change the name of a file.	rename(2)
set:	change value of shell variable.	csh(1)
cd:	change working directory.	cd(1)
ching: the book of	changes and other cookies.	ching(6)
better random number generator; routines for	changing generators. /srandom, initstate, setstate:	random(3)
pipe: create an interprocess communication	channel.	pipe(2)
ungetc: push	character back into input stream.	ungetc(3S)
isspace, ispunct, isprint, iscntrl, isascii:	character classification macros. /isdigit, isalnum,	ctype(3)
eqnchar: special	character definitions for eqn.	eqnchar(7)
getc, fgetc: get a	character from a logical unit.	getc(3F)
index, rindex, lnblnk, len: tell about	character objects.	index(3F)
getc, getchar, fgetc, getw: get	character or word from stream.	getc(3S)
putc, putchar, fputc, putw: put	character or word on a stream.	putc(3S)
ascii: map of ASCII	character set.	ascii(7)
putc, fputc: write a	character to a fortran logical unit.	putc(3F)
style: analyze surface	characteristics of a document.	style(1)
tr: translate	characters.	tr(1)
prec: C precedence	chart.	prec(7)
snake, snscore: display	chase game.	snake(6)
	chdir: change current working directory.	chdir(2)
	chdir: change default directory.	chdir(3F)
	chdir: change directory.	csh(1)
dcheck: file system directory consistency	check.	dcheck(8)
icheck: file system storage consistency	check.	icheck(8)
fsck: file system consistency	check and interactive repair.	fsck(8)
ci:	check in RCS revisions.	ci(1)
checknr:	check nroff/troff files.	checknr(1)

co:	check out RCS revisions.	co(1)
eqn, neqn,	checkeq: typeset mathematics.	eqn(1)
quotacheck: file system quota consistency	checker.	quotacheck(8)
fastboot, fasthalt: reboot/halt the system without	checking the disks.	fastboot(8)
	checknr: check nroff/troff files.	checknr(1)
	chfn: change finger entry.	chfn(1)
	chgrp: change group.	chgrp(1)
	ching: the book of changes and other cookies.	ching(6)
	chmod: change mode.	chmod(1)
	chmod: change mode of a file.	chmod(3F)
	chmod: change mode of file.	chmod(2)
	chown: change owner.	chown(8)
	chown: change owner and group of a file.	chown(2)
	chroot: change root directory.	chroot(2)
	chsh: change default login shell.	chsh(1)
	ci: check in RCS revisions.	ci(1)
closepl:/ plot: openpl, erase, label, line,	circle, arc, move, cont, point, linemod, space,	plot(3X)
ispunct, isprint, iscntrl, isascii: character	classification macros. /isdigit, isalnum, isspace,	ctype(3)
default: catchall	clause in switch.	csh(1)
cleanlpd:	clean line printer daemon environment.	cleanlpd(8)
	cleanlpd: clean line printer daemon environment.	cleanlpd(8)
uuclean: uucp spool directory	clean-up.	uuclean(8C)
	clear: clear terminal screen.	clear(1)
clri:	clear i-node.	clri(8)
clear:	clear terminal screen.	clear(1)
ferror, feof,	clearerr, fileno: stream status inquiries.	ferror(3S)
csh: a shell (command interpreter) with	C-like syntax.	csh(1)
cron:	clock daemon.	cron(8)
	close: delete a descriptor.	close(2)
shutdown:	close down the system at a given time.	shutdown(8)
fclose, fflush:	close or flush a stream.	fclose(3S)
opendir, readdir, telldir, seekdir, rewinddir,	closedir: directory operations.	directory(3)
opendir, readdir, telldir, seekdir, rewinddir,	closedir: directory operations. directory:	directory(3X)
syslog, openlog,	closelog: control system log.	syslog(3)
circle, arc, move, cont, point, linemod, space,	closepl: graphics interface. /erase, label, line,	plot(3X)
	clri: clear i-node.	clri(8)
	cmp: compare two files.	cmp(1)
	co: check out RCS revisions.	co(1)
pi: Pascal interpreter	code translator.	pi(1)
	col: filter reverse line feeds.	col(1)
	colcrt: filter nroff output for CRT previewing.	colcrt(1)
log. dmesg:	collect system diagnostic messages to form error	dmesg(8)
	colrm: remove columns from a *file*.	colrm(1)
colrm: remove	columns from a file.	colrm(1)
files.	comm: select or reject lines common to two sorted	comm(1)
exec: overlay shell with specified	command.	csh(1)
time: time	command.	csh(1)
routines for returning a stream to a remote	command. rcmd, rresvport, ruserok:	rcmd(3X)
rexec: return stream to a remote	command.	rexec(3X)
system: issue a shell	command.	system(3)
system: execute a UNIX	command.	system(3F)
test: condition	command.	test(1)
time: time a	command.	time(1)
nice, nohup: run a	command at low priority (*sh* only).	nice(1)
switch: multi-way	command branch.	csh(1)
uux: unix to unix	command execution.	uux(1C)
rehash: recompute	command hash table.	csh(1)
unhash: discard	command hash table.	csh(1)
hashstat: print	command hashing statistics.	csh(1)
nohup: run	command immune to hangups.	csh(1)
csh: a shell	(command interpreter) with C-like syntax.	csh(1)
whatis: describe what a	command is.	whatis(1)
readonly, set, shift, times, trap, umask, wait:	command language. /exec, exit, export, login, read,	sh(1)
getarg, iargc: return	command line arguments.	getarg(3F)
repeat: execute	command repeatedly.	csh(1)
rc:	command script for auto-reboot and daemons.	rc(8)
onintr: process interrupts in	command scripts.	csh(1)
apply: apply a	command to a set of arguments.	apply(1)
goto:	command transfer.	csh(1)
else: alternative	commands.	csh(1)
intro: introduction to	commands.	intro(1)
introduction to system maintenance and operation	commands. intro:	intro(8)
rcsintro: introduction to RCS	commands.	rcsintro(1)
at: execute	commands at a later time.	at(1)
apropos: locate	commands by keyword lookup.	apropos(1)

while: repeat	commands conditionally.	csh(1)
lastcomm: show last	commands executed in reverse order.	lastcomm(1)
source: read	commands from file.	csh(1)
comm: select or reject lines	common to two sorted files.	comm(1)
socket: create an endpoint for	communication.	socket(2)
pipe: create an interprocess	communication channel.	pipe(2)
ipcs: report inter-process	communication facilities status.	ipcs(1)
talkd: remote user	communication server.	talkd(8C)
users:	compact list of users who are on the system.	users(1)
files, and cat them.	compact, uncompact, ccat: compress and uncompress	compact(1)
diff: differential file and directory	comparator.	diff(1)
rcsdiff:	compare RCS revisions.	rcsdiff(1)
cmp:	compare two files.	cmp(1)
diff3: 3-way differential file	comparison.	diff3(1)
intro: introduction to	compatibility library functions.	intro(3C)
liszt:	compile a Franz Lisp program.	liszt(1)
cc: C	compiler.	cc(1)
f77: Fortran 77	compiler.	f77(1)
pc: Pascal	compiler.	pc(1)
error: analyze and disperse	compiler *error* messages.	error(1)
yacc: yet another	compiler-compiler.	yacc(1)
fp: Functional Programming language	compiler/interpreter.	fp(1)
wait: wait for background processes to	complete.	csh(1)
wait: await	completion of process.	wait(1)
compress, uncompress, zcat:	compress and expand data.	compress(1)
compact, uncompact, ccat:	compress and uncompress files, and cat them.	compact(1)
data.	compress, uncompress, zcat: compress and expand	compress(1)
learn:	computer aided instruction about UNIX.	learn(1)
hangman:	Computer version of the game hangman.	hangman(6)
	comsat: biff server.	comsat(8C)
tac:	concatenate and print files in reverse order.	tac(1)
test:	condition command.	test(1)
endif: terminate	conditional.	csh(1)
if:	conditional statement.	csh(1)
while: repeat commands	conditionally.	csh(1)
gettytab: terminal	configuration data base.	gettytab(5)
uxrc:	configuration file for kernel.	uxrc(8)
ifconfig:	configure network interface parameters.	ifconfig(8C)
	connect: initiate a connection on a socket.	connect(2)
tip, cu:	connect to a remote system.	tip(1C)
dosc:	connect to proc/286 system.	dosc(1)
getpeername: get name of	connected peer.	getpeername(2)
socketpair: create a pair of	connected sockets.	socketpair(2)
shutdown: shut down part of a full-duplex	connection.	shutdown(2)
accept: accept a	connection on a socket.	accept(2)
connect: initiate a	connection on a socket.	connect(2)
listen: listen for	connections on a socket.	listen(2)
dcheck: file system directory	consistency check.	dcheck(8)
icheck: file system storage	consistency check.	icheck(8)
fsck: file system	consistency check and interactive repair.	fsck(8)
quotacheck: file system quota	consistency checker.	quotacheck(8)
show what versions of object modules were used to	construct a file. what:	what(1)
newfs:	construct a new file system.	newfs(8)
mkproto:	construct a prototype file system.	mkproto(8)
deroff: remove nroff, troff, tbl and eqn	constructs.	deroff(1)
setrlimit: control maximum system resource	consumption. getrlimit,	getrlimit(2)
vlimit: control maximum system resource	consumption.	vlimit(3C)
/openpl, erase, label, line, circle, arc, move,	cont, point, linemod, space, closepl: graphics/	plot(3X)
ls: list	contents of directory.	ls(1)
sigstack: set and/or get signal stack	context.	sigstack(2)
sh, for, case, if, while, : , . , break,	continue, cd, eval, exec, exit, export, login,/	sh(1)
	continue: cycle in loop.	csh(1)
fcntl: file	control.	fcntl(2)
ioctl:	control device.	ioctl(2)
init: process	control initialization.	init(8)
getrlimit, setrlimit:	control maximum system resource consumption.	getrlimit(2)
vlimit:	control maximum system resource consumption.	vlimit(3C)
msgctl: message	control operations.	msgctl(2)
semctl: semaphore	control operations.	semctl(2)
shmctl: shared memory	control operations.	shmctl(2)
fcntl: file	control options.	fcntl(5)
lpc: line printer	control program.	lpc(8)
syslog, openlog, closelog:	control system log.	syslog(3)
vhangup: virtually "hangup" the current	control terminal.	vhangup(2)
term:	conventional names for terminals.	term(7)

ecvt, fcvt, gcvt: output	conversion.	ecvt(3)
long, short: integer object	conversion.	long(3F)
printf, fprintf, sprintf: formatted output	conversion.	printf(3S)
scanf, fscanf, sscanf: formatted input	conversion.	scanf(3S)
units:	conversion program.	units(1)
dd:	convert and copy a file.	dd(1)
number:	convert Arabic numerals to English.	number(6)
ranlib:	convert archives to random libraries.	ranlib(1)
atof, atoi, atol:	convert ASCII to numbers.	atof(3)
ctime, localtime, gmtime, asctime, timezone:	convert date and time to ASCII.	ctime(3)
htable:	convert NIC standard format host tables.	htable(8)
bcd:	convert to antique media.	bcd(6)
htonl, htons, ntohl, ntohs:	convert values between host and network byte order.	byteorder(3n)
ching: the book of changes and other	cookies.	ching(6)
fpu: determine presence of the floating point	coprocessor.	fpu(1)
cp:	copy.	cp(1)
rcp: remote file	copy.	rcp(1C)
uucp, uulog: unix to unix	copy.	uucp(1C)
dd: convert and	copy a file.	dd(1)
tcopy:	copy a mag tape.	tcopy(1)
doscopyd: MPS/DOS file	copy daemon.	doscopyd(8)
cpio:	copy file archives in and out.	cpio(1)
fork: create a	copy of this process.	fork(3F)
copy: standalone	copy program.	copy(8)
	copy: standalone copy program.	copy(8)
	core: format of memory image file.	core(5)
gcore: get	core images of running processes.	gcore(1)
functions. sin,	cos, tan, asin, acos, atan, atan2: trigonometric	sin(3M)
sinh,	cosh, tanh: hyperbolic functions.	sinh(3M)
wc: word	count.	wc(1)
sum: sum and	count blocks in a file.	sum(1)
	cp: copy.	cp(1)
	cpio: copy file archives in and out.	cpio(1)
- recover JOVE buffers after a system/editor	crash. jove_recover	jove_recover(1)
	crash: what happens when the system crashes.	crash(8V)
crash: what happens when the system	crashes.	crash(8V)
	creat: create a new file.	creat(2)
fork:	create a copy of this process.	fork(3F)
creat:	create a new file.	creat(2)
open: open a file for reading or writing, or	create a new file.	open(2)
fork:	create a new process.	fork(2)
socketpair:	create a pair of connected sockets.	socketpair(2)
ctags:	create a tags file.	ctags(1)
socket:	create an endpoint for communication.	socket(2)
mkstr:	create an error message file by massaging C source.	mkstr(1)
pipe:	create an interprocess communication channel.	pipe(2)
dosdisk: program to	create and display information for MPS/DOS vdisks.	dosdisk(8)
addbib:	create or extend bibliographic database.	addbib(1)
catman:	create the cat files for the manual.	catman(8)
umask: change or display file	creation mask.	csh(1)
umask: set file	creation mode mask.	umask(2)
cribbage: the card game	cribbage.	cribbage(6)
	cribbage: the card game cribbage.	cribbage(6)
	cron: clock daemon.	cron(8)
lxref: lisp	cross reference program.	lxref(1)
pxref: Pascal	cross-reference program.	pxref(1)
colcrt: filter nroff output for	CRT previewing.	colcrt(1)
more, page: file perusal filter for	crt viewing.	more(1)
	crypt: encode/decode.	crypt(1)
	crypt, setkey, encrypt: DES encryption.	crypt(3)
syntax.	csh: a shell (command interpreter) with C-like	csh(1)
	ctags: create a tags file.	ctags(1)
convert date and time to ASCII.	ctime, localtime, gmtime, asctime, timezone:	ctime(3)
time,	ctime, ltime, gmtime: return system time.	time(3F)
tip,	cu: connect to a remote system.	tip(1C)
vhangup: virtually "hangup" the	current control terminal.	vhangup(2)
gethostid, sethostid: get/set unique identifier of	current host.	gethostid(2)
gethostname, sethostname: get/set name of	current host.	gethostname(2)
hostnm: get name of	current host.	hostnm(3F)
hostid: set or print identifier of	current host system.	hostid(1)
hostname: set or print name of	current host system.	hostname(1)
jobs: print	current job list.	csh(1)
sigsetmask: set	current signal mask.	sigsetmask(2)
whoami: print effective	current user id.	whoami(1)
chdir: change	current working directory.	chdir(2)

getcwd: get pathname of	current working directory.	getcwd(3F)
getwd: get	current working directory pathname.	getwd(3)
motion.	curses: screen functions with "optimal" cursor	curses(3X)
curses: screen functions with "optimal"	cursor motion.	curses(3X)
spline: interpolate smooth	curve.	spline(1G)
continue:	cycle in loop.	csh(1)
cron: clock	daemon.	cron(8)
doscopyd: MPS/DOS file copy	daemon.	doscopyd(8)
dosprint: MPS/DOS spooler	daemon.	dosprint(8)
lpd: line printer	daemon.	lpd(8)
routed: network routing	daemon.	routed(8C)
cleanlpd: clean line printer	daemon environment.	cleanlpd(8)
rc: command script for auto-reboot and	daemons.	rc(8)
ftpd:	DARPA Internet File Transfer Protocol server.	ftpd(8C)
telnetd:	DARPA TELNET protocol server.	telnetd(8C)
tftpd:	DARPA Trivial File Transfer Protocol server.	tftpd(8C)
compress, uncompress, zcat: compress and expand	data.	compress(1)
eval: re-evaluate shell	data.	csh(1)
gprof: display call graph profile	data.	gprof(1)
prof: display profile	data.	prof(1)
ttys: terminal initialization	data.	ttys(5)
gettytab: terminal configuration	data base.	gettytab(5)
hosts: host name	data base.	hosts(5)
networks: network name	data base.	networks(5)
phones: remote host phone number	data base.	phones(5)
printcap: printer capability	data base.	printcap(5)
protocols: protocol name	data base.	protocols(5)
services: service name	data base.	services(5)
termcap: terminal capability	data base.	termcap(5)
vgrindefs: vgrind's language definition	data base.	vgrindefs(5)
newaliases: rebuild the	data base for the mail aliases file.	newaliases(1)
ttytype:	data base of terminal types by port.	ttytype(5)
dbminit, fetch, store, delete, firstkey, nextkey:	data base subroutines.	dbm(3X)
brk, sbrk: change	data segment size.	brk(2)
null:	data sink.	null(4)
types: primitive system	data types.	types(5)
addbib: create or extend bibliographic	database.	addbib(1)
roffbib: run off bibliographic	database.	roffbib(1)
sortbib: sort bibliographic	database.	sortbib(1)
join: relational	database operator.	join(1)
date: print and set the	date.	date(1)
gettimeofday, settimeofday: get/set	date and time.	gettimeofday(2)
time, ftime: get	date and time.	time(3C)
fdate: return	date and time in an ASCII string.	fdate(3F)
localtime, gmtime, asctime, timezone: convert	date and time to ASCII. ctime,	ctime(3)
touch: update	date last modified of a file.	touch(1)
idate, itime: return	date or time in numerical form.	idate(3F)
	date: print and set the date.	date(1)
file.. .PP dstrules:	Daylight savings time and time zone name rule	dstrules(5)
data base subroutines.	dbminit, fetch, store, delete, firstkey, nextkey:	dbm(3X)
	dbx: debugger.	dbx(1)
	dc: desk calculator.	dc(1)
	dcheck: file system directory consistency check.	dcheck(8)
	dd: convert and copy a file.	dd(1)
adb:	debugger.	adb(1)
dbx:	debugger.	dbx(1)
pdx: pascal	debugger.	pdx(1)
od: octal,	decimal, hex, ascii dump.	od(1)
	default: catchall clause in switch.	csh(1)
chdir: change	default directory.	chdir(3F)
chsh: change	default login shell.	chsh(1)
vgrindefs: vgrind's language	definition data base.	vgrindefs(5)
mode.. Standalone mode:	definition of this Sanyo/ICON machine operation	standalone(8)
eqnchar: special character	definitions for eqn.	eqnchar(7)
stty, gtty: set and get terminal state	(defunct).	stty(3C)
close:	delete a descriptor.	close(2)
dbminit, fetch, store,	delete, firstkey, nextkey: data base subroutines.	dbm(3X)
tail:	deliver the last part of a file.	tail(1)
mesg: permit or	deny messages.	mesg(1)
tset: terminal	dependent initialization.	tset(1)
constructs.	deroff: remove nroff, troff, tbl and eqn	deroff(1)
crypt, setkey, encrypt:	DES encryption.	crypt(3)
whatis:	describe what a command is.	whatis(1)
mailaddr: mail addressing	description.	mailaddr(7)
getdiskbyname: get disk	description by its name.	getdisk(3X)

remote: remote host	description file.	remote(5)
close: delete a	descriptor.	close(2)
dup, dup2: duplicate a	descriptor.	dup(2)
getfstype, setfsent, endfsent: get file system	descriptor file entry. /getfsspec, getfsfile,	getfsent(3X)
getdtablesize: get	descriptor table size.	getdtablesize(2)
dc:	desk calculator.	dc(1)
dosprinters:	destinations for spooled output from SLPT printers.	dosprinters(5)
access:	determine accessability of a file.	access(3F)
access:	determine accessibility of file.	access(2)
file:	determine file type.	file(1)
coprocessor. fpu:	determine presence of the floating point	fpu(1)
fold: fold long lines for finite width output	device.	fold(1)
ioctl: control	device.	ioctl(2)
swapon: specify additional	device for paging and swapping.	swapon(8)
	df: disk free.	df(1)
flmin, flmax, ffrac, dflmin, dflmax,	dffrac, inmax: return extreme values.	flmin(3F)
flmin, flmax, ffrac, dflmin,	dflmax, dffrac, inmax: return extreme values.	flmin(3F)
values. flmin, flmax, ffrac,	dflmin, dflmax, dffrac, inmax: return extreme	flmin(3F)
dmesg: collect system	diagnostic messages to form error log.	dmesg(8)
print wordy sentences; thesaurus for	diction. diction,explain:	diction(1)
diction– print wordy sentences; thesaurus for	diction. explain,	explain(1)
diction. explain,	diction– print wordy sentences; thesaurus for	explain(1)
for diction.	diction,explain: print wordy sentences; thesaurus	diction(1)
	diff: differential file and directory comparator.	diff(1)
	diff3: 3-way differential file comparison.	diff3(1)
diff:	differential file and directory comparator.	diff(1)
diff3: 3-way	differential file comparison.	diff3(1)
	dir: format of directories.	dir(5)
dir: format of	directories.	dir(5)
rm, rmdir: remove (unlink) files or	directories.	rm(1)
rmdir, rm: remove (unlink)	directories or files.	rmdir(1)
cd: change working	directory.	cd(1)
chdir: change current working	directory.	chdir(2)
chdir: change default	directory.	chdir(3F)
chroot: change root	directory.	chroot(2)
cd: change	directory.	csh(1)
chdir: change	directory.	csh(1)
getcwd: get pathname of current working	directory.	getcwd(3F)
ls: list contents of	directory.	ls(1)
mkdir: make a	directory.	mkdir(1)
scandir: scan a	directory.	scandir(3)
swapon: specify a swap	directory.	swapon(2)
uuclean: uucp spool	directory clean-up.	uuclean(8C)
diff: differential file and	directory comparator.	diff(1)
dcheck: file system	directory consistency check.	dcheck(8)
unlink: remove	directory entry.	unlink(2)
unlink: remove a	directory entry.	unlink(3F)
mkdir: make a	directory file.	mkdir(2)
rmdir: remove a	directory file.	rmdir(2)
mklost+found: make a lost+found	directory for fsck.	mklost+found(8)
pwd: working	directory name.	pwd(1)
rewinddir, closedir: directory operations.	directory: opendir, readdir, telldir, seekdir,	directory(3X)
readdir, telldir, seekdir, rewinddir, closedir:	directory operations. opendir,	directory(3)
readdir, telldir, seekdir, rewinddir, closedir:	directory operations. directory: opendir,	directory(3X)
getwd: get current working	directory pathname.	getwd(3)
popd: pop shell	directory stack.	csh(1)
pushd: push shell	directory stack.	csh(1)
	dis: an mc68020 disassembler.	dis(1)
dis: an mc68020	disassembler.	dis(1)
quota: display	disc usage and limits.	quota(1)
unhash:	discard command hash table.	csh(1)
unset:	discard shell variables.	csh(1)
binstl: program to install bootloader on	disk.	binstl(8)
synchronize a file's in-core state with that on	disk. fsync:	fsync(2)
getdiskbyname: get	disk description by its name.	getdisk(3X)
dkfmt: standalone	disk formatter.	dkfmt(8)
df:	disk free.	df(1)
park: program to park the hard	disk heads.	park(8)
quota: manipulate	disk quotas.	quota(2)
du: summarize	disk usage.	du(1)
dosdisks: list of MPS/DOS virtual	disks.	dosdisks(5)
reboot/halt the system without checking the	disks. fastboot, fasthalt:	fastboot(8)
mount, umount: mount and	dismount file system.	mount(8)
error: analyze and	disperse compiler error messages.	error(1)
rain: animated raindrops	display.	rain(6)

gprof:	display call graph profile data.	gprof(1)
snake, snscore:	display chase game.	snake(6)
quota:	display disc usage and limits.	quota(1)
vi: screen oriented (visual)	display editor based on ex.	vi(1)
umask: change or	display file creation mask.	csh(1)
whodos:	display information about dosc users.	whodos(1)
dosdisk: program to create and	display information for MPS/DOS vdisks.	dosdisk(8)
prof:	display profile data.	prof(1)
sysline:	display system status on status line of a terminal.	sysline(1)
worms: animate worms on a	display terminal.	worms(6)
jove: an interactive	display-oriented text editor.	jove(1)
hypot, cabs: Euclidean	distance.	hypot(3M)
rdist: remote file	distribution program.	rdist(1)
	dkfmt: standalone disk formatter.	dkfmt(8)
	dmem, kmem: main memory.	mem(4)
error log.	dmesg: collect system diagnostic messages to form	dmesg(8)
	doctor: interact with a psychoanalyst.	doctor(6)
style: analyze surface characteristics of a	document.	style(1)
refer: find and insert literature references in	documents.	refer(1)
w: who is on and what they are	doing.	w(1)
rogue: Exploring The Dungeons of	Doom.	rogue(6)
	dosc: connect to proc/286 system.	dosc(1)
whodos: display information about	dosc users.	whodos(1)
	doscopyd: MPS/DOS file copy daemon.	doscopyd(8)
for MPS/DOS vdisks.	dosdisk: program to create and display information	dosdisk(8)
	dosdisks: list of MPS/DOS virtual disks.	dosdisks(5)
	dosprint: MPS/DOS spooler daemon.	dosprint(8)
SLPT printers.	dosprinters: destinations for spooled output from	dosprinters(5)
shutdown: shut	down part of a full-duplex connection.	shutdown(2)
shutdown: close	down the system at a given time.	shutdown(8)
rand,	drand, irand: return random values.	rand(3F)
graph:	draw a graph.	graph(1G)
arithmetic: provide	drill in number facts.	arithmetic(6)
rule file.. .PP	dstrules: Daylight savings time and time zone name	dstrules(5)
etime,	dtime: return elapsed execution time.	etime(3F)
	du: summarize disk usage.	du(1)
dump: incremental file system	dump.	dump(8)
od: octal, decimal, hex, ascii	dump.	od(1)
rdump: file system	dump across the network.	rdump(8C)
rrestore: restore a file system	dump across the network.	rrestore(8C)
	dump, dumpdates: incremental dump format.	dump(5)
dumpfs:	dump file system information.	dumpfs(8)
dump, dumpdates: incremental	dump format.	dump(5)
	dump: incremental file system dump.	dump(8)
kgmon: generate a	dump of the operating system's profile buffers.	kgmon(8)
dump,	dumpdates: incremental dump format.	dump(5)
	dumpfs: dump file system information.	dumpfs(8)
rogue: Exploring The	Dungeons of Doom.	rogue(6)
	dup, dup2: duplicate a descriptor.	dup(2)
dup,	dup2: duplicate a descriptor.	dup(2)
dup, dup2:	duplicate a descriptor.	dup(2)
echo:	echo arguments.	csh(1)
echo:	echo arguments.	echo(1)
	echo: echo arguments.	csh(1)
	echo: echo arguments.	echo(1)
ping: send ICMP	ECHO_REQUEST packets to network hosts.	ping(8)
	ecvt, fcvt, gcvt: output conversion.	ecvt(3)
	ed: text editor.	ed(1)
end, etext,	edata: last locations in program.	end(3)
ex,	edit: text editor.	ex(1)
vipw:	edit the password file.	vipw(8)
edquota:	edit user quotas.	edquota(8)
ed: text	editor.	ed(1)
ex, edit: text	editor.	ex(1)
jove: an interactive display-oriented text	editor.	jove(1)
ld: link	editor.	ld(1)
sed: stream	editor.	sed(1)
TEACHJOVE - learn how to use the JOVE	editor.	teachjove(1)
vi: screen oriented (visual) display	editor based on ex.	vi(1)
a.out: assembler and link	editor output.	a.out(5)
	edquota: edit user quotas.	edquota(8)
whoami: print	effective current user id.	whoami(1)
setregid: set real and	effective group ID.	setregid(2)
setreuid: set real and	effective user ID's.	setreuid(2)
vfork: spawn new process in a virtual memory	efficient way.	vfork(2)

	efl: Extended Fortran Language.	efl(1)
grep,	egrep, fgrep: search a file for a pattern.	grep(1)
etime, dtime: return	elapsed execution time.	etime(3F)
insque, remque: insert/remove	element from a queue.	insque(3)
soelim:	eliminate .so's from nroff input.	soelim(1)
	else: alternative commands.	csh(1)
setquota:	enable/disable quotas on a file system.	setquota(2)
uuencode: format of an	encoded uuencode file.	uuencode(5)
crypt:	encode/decode.	crypt(1)
mail. uuencode,uudecode:	encode/decode a binary file for transmission via	uuencode(1C)
crypt, setkey,	encrypt: DES encryption.	crypt(3)
crypt, setkey, encrypt: DES	encryption.	crypt(3)
makekey: generate	encryption key.	makekey(8)
	end, etext, edata: last locations in program.	end(3)
logout:	end session.	csh(1)
	end: terminate loop.	csh(1)
/getfsspec, getfsfile, getfstype, setfsent,	endfsent: get file system descriptor file entry.	getfsent(3X)
getgrent, getgrgid, getgrnam, setgrent,	endgrent: get group file entry.	getgrent(3)
gethostbyaddr, gethostbyname, sethostent,	endhostent: get network host entry. gethostent,	gethostent(3n)
	endif: terminate conditional.	csh(1)
getnetent, getnetbyaddr, getnetbyname, setnetent,	endnetent: get network entry.	getnetent(3n)
socket: create an	endpoint for communication.	socket(2)
getprotobynumber, getprotobyname, setprotoent,	endprotoent: get protocol entry. getprotoent,	getprotoent(3n)
getpwent, getpwuid, getpwnam, setpwent,	endpwent: get password file entry.	getpwent(3)
getservbyport, getservbyname, setservent,	endservent: get service entry. getservent,	getservent(3n)
	endsw: terminate switch.	csh(1)
number: convert Arabic numerals to	English.	number(6)
xsend, xget,	enroll: secret mail.	xsend(1)
nlist: get	entries from name list.	nlist(3)
chfn: change finger	entry.	chfn(1)
setfsent, endfsent: get file system descriptor file	entry. getfsent, getfsspec, getfsfile, getfstype,	getfsent(3X)
getgrnam, setgrent, endgrent: get group file	entry. getgrent, getgrgid,	getgrent(3)
sethostent, endhostent: get network host	entry. gethostent, gethostbyaddr, gethostbyname,	gethostent(3n)
getnetbyname, setnetent, endnetent: get network	entry. getnetent, getnetbyaddr,	getnetent(3n)
setprotoent, endprotoent: get protocol	entry. /getprotobynumber, getprotobyname,	getprotoent(3n)
getpwnam, setpwent, endpwent: get password file	entry. getpwent, getpwuid,	getpwent(3)
getservbyname, setservent, endservent: get service	entry. getservent, getservbyport,	getservent(3n)
unlink: remove directory	entry.	unlink(2)
unlink: remove a directory	entry.	unlink(3F)
execl, execv, execle, execlp, execvp, exec, exece,	environ: execute a file.	execl(3)
	environ: user environment.	environ(7)
cleanlpd: clean line printer daemon	environment.	cleanlpd(8)
setenv: set variable in	environment.	csh(1)
environ: user	environment.	environ(7)
printenv: print out the	environment.	printenv(1)
window: window	environment.	window(1)
getenv: value for	environment name.	getenv(3)
unsetenv: remove	environment variables.	csh(1)
getenv: get value of	environment variables.	getenv(3F)
eqnchar: special character definitions for	eqn.	eqnchar(7)
deroff: remove nroff, troff, tbl and	eqn constructs.	deroff(1)
	eqn, neqn, checkeq: typeset mathematics.	eqn(1)
	eqnchar: special character definitions for eqn.	eqnchar(7)
linemod, space, closepl: graphics/ plot: openpl,	erase, label, line, circle, arc, move. cont, point,	plot(3X)
messages.	error: analyze and disperse compiler error	error(1)
dmesg: collect system diagnostic messages to form	error log.	dmesg(8)
mkstr: create an	error message file by massaging C source.	mkstr(1)
error: analyze and disperse compiler	error messages.	error(1)
perror, sys_errlist, sys_nerr: system	error messages.	perror(3)
perror, gerror, ierrno: get system	error messages.	perror(3F)
intro: introduction to system calls and	error numbers.	intro(2)
eyacc: modified yacc allowing much improved	error recovery.	eyacc(1)
spell, spellin, spellout: find spelling	errors.	spell(1)
traper: trap arithmetic	errors.	traper(3F)
end,	etext, edata: last locations in program.	end(3)
	etime, dtime: return elapsed execution time.	etime(3F)
hypot, cabs:	Euclidean distance.	hypot(3M)
/if, while, : , . , break, continue, cd,	eval, exec, exit, export, login, read, readonly,/	sh(1)
	eval: re-evaluate shell data.	csh(1)
expr:	evaluate arguments as an expression.	expr(1)
history: print history	event list.	csh(1)
screen oriented (visual) display editor based on	ex. vi:	vi(1)
	ex, edit: text editor.	ex(1)
lpq: spool queue	examination program.	lpq(1)
execl, execv, execle, execlp, execvp,	exec, exece, environ: execute a file.	execl(3)

/while, : , . , break, continue, cd, eval,	exec, exit, export, login, read, readonly, set,/	sh(1)
	exec: overlay shell with specified command.	csh(1)
execl, execv, execle, execlp, execvp, exec,	exece, environ: execute a file.	execl(3)
environ: execute a file.	execl, execv, execle, execlp, execvp, exec, exece,	execl(3)
execute a file. execl, execv,	execle, execlp, execvp, exec, exece, environ:	execl(3)
file. execl, execv, execle,	execlp, execvp, exec, exece, environ: execute a	execl(3)
sticky:	executable files with persistent text.	sticky(8)
execle, execlp, execvp, exec, exece, environ:	execute a file. execl, execv,	execl(3)
execve:	execute a file.	execve(2)
alarm:	execute a subroutine after a specified time.	alarm(3F)
system:	execute a UNIX command.	system(3F)
repeat:	execute command repeatedly.	csh(1)
at:	execute commands at a later time.	at(1)
lastcomm: show last commands	executed in reverse order.	lastcomm(1)
uux: unix to unix command	execution.	uux(1C)
acct:	execution accounting file.	acct(5)
sleep: suspend	execution for an interval.	sleep(1)
sleep: suspend	execution for an interval.	sleep(3F)
sleep: suspend	execution for interval.	sleep(3)
monitor, monstartup, moncontrol: prepare	execution profile.	monitor(3)
pxp: Pascal	execution profiler.	pxp(1)
rexecd: remote	execution server.	rexecd(8C)
etime, dtime: return elapsed	execution time.	etime(3F)
profil:	execution time profile.	profil(2)
pix: Pascal interpreter and	executor.	pix(1)
environ: execute a file. execl,	execv, execle, execlp, execvp, exec, exece,	execl(3)
	execve: execute a file.	execve(2)
execl, execv, execle, execlp,	execvp, exec, exece, environ: execute a file.	execl(3)
link: make a link to an	existing file.	link(3F)
tunefs: tune up an	existing file system.	tunefs(8)
/ : , . , break, continue, cd, eval, exec,	exit, export, login, read, readonly, set, shift,/	sh(1)
breaksw:	exit from switch.	csh(1)
	exit: leave shell.	csh(1)
	_exit: terminate a process.	exit(2)
pending output.	exit: terminate a process after flushing any	exit(3)
	exit: terminate process with status.	exit(3F)
break:	exit while/foreach loop.	csh(1)
power, square root.	exp, log, log10, pow, sqrt: exponential, logarithm,	exp(3M)
glob: filename	expand argument list.	csh(1)
compress, uncompress, zcat: compress and	expand data.	compress(1)
expand, unexpand:	expand tabs to spaces, and vice versa.	expand(1)
versa.	expand, unexpand: expand tabs to spaces, and vice	expand(1)
for diction.	explain, diction– print wordy sentences; thesaurus	explain(1)
diction. diction,	explain: print wordy sentences; thesaurus for	diction(1)
aardvark: yet another	exploration game.	aardvark(6)
adventure: an	exploration game.	adventure(6)
rogue:	Exploring The Dungeons of Doom.	rogue(6)
frexp, ldexp, modf: split into mantissa and	exponent.	frexp(3)
exp, log, log10, pow, sqrt:	exponential, logarithm, power, square root.	exp(3M)
/ . , break, continue, cd, eval, exec, exit,	export, login, read, readonly, set, shift, times,/	sh(1)
	expr: evaluate arguments as an expression.	expr(1)
expr: evaluate arguments as an	expression.	expr(1)
re_comp, re_exec: regular	expression handler.	regex(3)
addbib: create or	extend bibliographic database.	addbib(1)
efl:	Extended Fortran Language.	efl(1)
strings. xstr:	extract strings from C programs to implement shared	xstr(1)
recovery.	eyacc: modified yacc allowing much improved error	eyacc(1)
	f77: Fortran 77 compiler.	f77(1)
ioinit: change	f77 I/O initialization.	ioinit(3F)
tclose, tread, twrite, trewin, tskipf, tstate:	f77 tape I/O. topen,	topen(3F)
functions.	fabs, floor, ceil: absolute value, floor, ceiling	floor(3M)
signal: simplified software signal	facilities.	signal(3C)
sigvec: software signal	facilities.	sigvec(2)
ipcs: report inter-process communication	facilities status.	ipcs(1)
true,	false: provide truth values.	true(1)
	false, true: provide truth values.	false(1)
checking the disks.	fastboot, fasthalt: reboot/halt the system without	fastboot(8)
the disks. fastboot,	fasthalt: reboot/halt the system without checking	fastboot(8)
abort: generate a	fault.	abort(3)
trpfpe, fpecnt: trap and repair floating point	faults.	trpfpe(3F)
export, login,/ sh, for, case, if, while, : ,	. , break, continue, cd, eval, exec, exit,	sh(1)
exit, export, login,/ sh, for, case, if, while,	: , . , break, continue, cd, eval, exec,	sh(1)
	fclose, fflush: close or flush a stream.	fclose(3S)
	fcntl: file control.	fcntl(2)
	fcntl: file control options.	fcntl(5)

ecvt,	fcvt, gcvt: output conversion.	ecvt(3)
	fdate: return date and time in an ASCII string.	fdate(3F)
fopen, freopen,	fdopen: open a stream.	fopen(3S)
ferror,	feof, clearerr, fileno: stream status inquiries.	ferror(3S)
inquiries.	ferror, feof, clearerr, fileno: stream status	ferror(3S)
subroutines. dbminit,	fetch, store, delete, firstkey, nextkey: data base	dbm(3X)
head: give first	few lines.	head(1)
fclose,	fflush: close or flush a stream.	fclose(3S)
extreme values. flmin, flmax,	ffrac, dflmin, dflmax, dffrac, inmax: return	flmin(3F)
bcopy, bcmp, bzero,	ffs: bit and byte string operations.	bstring(3)
	fg: bring job into foreground.	csh(1)
getc,	fgetc: get a character from a logical unit.	getc(3F)
getc, getchar,	fgetc, getw: get character or word from stream.	getc(3S)
gets,	fgets: get a string from a stream.	gets(3S)
grep, egrep,	fgrep: search a file for a pattern.	grep(1)
locate a program file including aliases and paths	(*csh* only). which:	which(1)
robots:	fight off villainous robots.	robots(6)
access: determine accessibility of	file.	access(2)
access: determine accessability of a	file.	access(3F)
acct: execution accounting	file.	acct(5)
chmod: change mode of	file.	chmod(2)
chmod: change mode of a	file.	chmod(3F)
chown: change owner and group of a	file.	chown(2)
colrm: remove columns from a	file.	colrm(1)
core: format of memory image	file.	core(5)
creat: create a new	file.	creat(2)
source: read commands from	file.	csh(1)
ctags: create a tags	file.	ctags(1)
dd: convert and copy a	file.	dd(1)
Daylight savings time and time zone name rule	file.. .PP dstrules:	dstrules(5)
execlp, execvp, exec, exece, environ: execute a	file. execl, execv, execle,	execl(3)
execve: execute a	file.	execve(2)
flock: apply or remove an advisory lock on an open	file.	flock(2)
fpr: print Fortran	file.	fpr(1)
group: group	file.	group(5)
link: make a hard link to a	file.	link(2)
link: make a link to an existing	file.	link(3F)
mkdir: make a directory	file.	mkdir(2)
mknod: make a special	file.	mknod(2)
mknod: build special	file.	mknod(8)
rebuild the data base for the mail aliases	file. newaliases:	newaliases(1)
open a file for reading or writing, or create a new	file. open:	open(2)
passwd: password	file.	passwd(5)
pr: print	file.	pr(1)
rcsfile: format of RCS	file.	rcsfile(5)
remote: remote host description	file.	remote(5)
rename: change the name of a	file.	rename(2)
rename: rename a	file.	rename(3F)
rev: reverse lines of a	file.	rev(1)
rmdir: remove a directory	file.	rmdir(2)
sccstorcs: build RCS file from SCCS	file.	sccstorcs(1)
sccstorcs: build RCS file from SCCS	file.	sccstorcs(8)
sfdate: set the time/date of a	file.	sfdate(1)
size: size of an object	file.	size(1)
the printable strings in a object, or other binary,	file. strings: find	strings(1)
sum: sum and count blocks in a	file.	sum(1)
symlink: make symbolic link to a	file.	symlink(2)
tail: deliver the last part of a	file.	tail(1)
touch: update date last modified of a	file.	touch(1)
uniq: report repeated lines in a	file.	uniq(1)
uuencode: format of an encoded uuencode	file.	uuencode(5)
vipw: edit the password	file.	vipw(8)
versions of object modules were used to construct a	file. what: show what	what(1)
write, writev: write on a	file.	write(2)
diff: differential	file and directory comparator.	diff(1)
cpio: copy	file archives in and out.	cpio(1)
rcs: change RCS	file attributes.	rcs(1)
bugfiler:	file bug reports in folders automatically.	bugfiler(8)
mkstr: create an error message	file by massaging C source.	mkstr(1)
diff3: 3-way differential	file comparison.	diff3(1)
fcntl:	file control.	fcntl(2)
fcntl:	file control options.	fcntl(5)
rcp: remote	file copy.	rcp(1C)
doscopyd: MPS/DOS	file copy daemon.	doscopyd(8)
umask: change or display	file creation mask.	csh(1)

umask: set	file creation mode mask.	umask(2)
	file: determine file type.	file(1)
rdist: remote	file distribution program.	rdist(1)
setfsent, endfsent: get file system descriptor	file entry. /getfsspec, getfsfile, getfstype,	getfsent(3X)
getgrgid, getgrnam, setgrent, endgrent: get group	file entry. getgrent,	getgrent(3)
getpwnam, setpwent, endpwent: get password	file entry. getpwent, getpwuid,	getpwent(3)
grep, egrep, fgrep: search a	file for a pattern.	grep(1)
uxrc: configuration	file for kernel.	uxrc(8)
open: open a	file for reading or writing, or create a new file.	open(2)
aliases: aliases	file for sendmail.	aliases(5)
uuencode,uudecode: encode/decode a binary	file for transmission via mail.	uuencode(1C)
ar: archive (library)	file format.	ar(5)
tar: tape archive	file format.	tar(5)
sccstorcs: build RCS	file from SCCS file.	sccstorcs(1)
sccstorcs: build RCS	file from SCCS file.	sccstorcs(8)
which: locate a program	file including aliases and paths (*csh* only).	which(1)
fsplit: split a multi-routine Fortran	file into individual files.	fsplit(1)
split: split a	file into pieces.	split(1)
merge: three-way	file merge.	merge(1)
pmerge: pascal	file merger.	pmerge(1)
mktemp: make a unique	file name.	mktemp(3)
fseek, ftell: reposition a	file on a logical unit.	fseek(3F)
more, page:	file perusal filter for crt viewing.	more(1)
stat, lstat, fstat: get	file status.	stat(2)
stat, lstat, fstat: get	file status.	stat(3F)
mkproto: construct a prototype	file system.	mkproto(8)
mount, umount: mount or remove	file system.	mount(2)
mount, umount: mount and dismount	file system.	mount(8)
newfs: construct a new	file system.	newfs(8)
repquota: summarize quotas for a	file system.	repquota(8)
setquota: enable/disable quotas on a	file system.	setquota(2)
tunefs: tune up an existing	file system.	tunefs(8)
repair. fsck:	file system consistency check and interactive	fsck(8)
getfsfile, getfstype, setfsent, endfsent: get	file system descriptor file entry. /getfsspec,	getfsent(3X)
dcheck:	file system directory consistency check.	dcheck(8)
dump: incremental	file system dump.	dump(8)
rdump:	file system dump across the network.	rdump(8C)
rrestore: restore a	file system dump across the network.	rrestore(8C)
hier:	file system hierarchy.	hier(7)
dumpfs: dump	file system information.	dumpfs(8)
quot: summarize	file system ownership.	quot(8)
quotacheck:	file system quota consistency checker.	quotacheck(8)
quotaon, quotaoff: turn	file system quotas on and off.	quotaon(8)
restore: incremental	file system restore.	restore(8)
icheck:	file system storage consistency check.	icheck(8)
mtab: mounted	file system table.	mtab(5)
fs, inode: format of	file system volume.	fs(5)
mkfs: program to make UNIX	file systems.	mkfs(8)
utime: set	file times.	utime(3C)
utimes: set	file times.	utimes(2)
uusend: send a	file to a remote host.	uusend(1C)
truncate: truncate a	file to a specified length.	truncate(2)
kermit: kermit	file transfer.	kermit(1)
ftp:	file transfer program.	ftp(1C)
tftp: trivial	file transfer program.	tftp(1C)
ftpd: DARPA Internet	File Transfer Protocol server.	ftpd(8C)
tftpd: DARPA Trivial	File Transfer Protocol server.	tftpd(8C)
file: determine	file type.	file(1)
basename: strip	filename affixes.	basename(1)
glob:	filename expand argument list.	csh(1)
ferror, feof, clearerr,	fileno: stream status inquiries.	ferror(3S)
checknr: check nroff/troff	files.	checknr(1)
cmp: compare two	files.	cmp(1)
comm: select or reject lines common to two sorted	files.	comm(1)
find: find	files.	find(1)
split a multi-routine Fortran file into individual	files. fsplit:	fsplit(1)
ident: identify	files.	ident(1)
lockf: record locking on	files.	lockf(3C)
makedev: make system special	files.	makedev(8)
mv: move or rename	files.	mv(1)
print log messages and other information about RCS	files. rlog:	rlog(1)
rmdir, rm: remove (unlink) directories or	files.	rmdir(1)
sort: sort or merge	files.	sort(1)
compact, uncompact, ccat: compress and uncompress	files, and cat them.	compact(1)
catman: create the cat	files for the manual.	catman(8)

abort:	generate a fault.	abort(3)
makekey:	generate encryption key.	makekey(8)
ncheck:	generate names from i-numbers.	ncheck(8)
rand, srand: random number	generator.	rand(3C)
lex:	generator of lexical analysis programs.	lex(1)
/srandom, initstate, setstate: better random number	generator; routines for changing generators.	random(3)
random number generator; routines for changing	generators. /srandom, initstate, setstate: better	random(3)
perror,	gerror, ierrno: get system error messages.	perror(3F)
	getarg, iargc: return command line arguments.	getarg(3F)
	getc, fgetc: get a character from a logical unit.	getc(3F)
from stream.	getc, getchar, fgetc, getw: get character or word	getc(3S)
stream. getc,	getchar, fgetc, getw: get character or word from	getc(3S)
	getcwd: get pathname of current working directory.	getcwd(3F)
	getdiskbyname: get disk description by its name.	getdisk(3X)
	getdtablesize: get descriptor table size.	getdtablesize(2)
getgid,	getegid: get group identity.	getgid(2)
	getenv: get value of environment variables.	getenv(3F)
	getenv: value for environment name.	getenv(3)
getuid,	geteuid: get user identity.	getuid(2)
setfsent, endfsent: get file system descriptor/	getfsent, getfsspec, getfsfile, getfstype,	getfsent(3X)
system descriptor file entry. getfsent, getfsspec,	getfsfile, getfstype, setfsent, endfsent: get file	getfsent(3X)
endfsent: get file system descriptor/ getfsent,	getfsspec, getfsfile, getfstype, setfsent,	getfsent(3X)
descriptor file/ getfsent, getfsspec, getfsfile,	getfstype, setfsent, endfsent: get file system	getfsent(3X)
getuid,	getgid: get user or group ID of the caller.	getuid(3F)
	getgid, getegid: get group identity.	getgid(2)
get group file entry.	getgrent, getgrgid, getgrnam, setgrent, endgrent:	getgrent(3)
file entry. getgrent,	getgrgid, getgrnam, setgrent, endgrent: get group	getgrent(3)
getgrent, getgrgid,	getgrnam, setgrent, endgrent: get group file entry.	getgrent(3)
	getgroups: get group access list.	getgroups(2)
endhostent: get network host entry. gethostent,	gethostbyaddr, gethostbyname, sethostent,	gethostent(3n)
host entry. gethostent, gethostbyaddr,	gethostbyname, sethostent, endhostent: get network	gethostent(3n)
sethostent, endhostent: get network host entry.	gethostent, gethostbyaddr, gethostbyname,	gethostent(3n)
current host.	gethostid, sethostid: get/set unique identifier of	gethostid(2)
host.	gethostname, sethostname: get/set name of current	gethostname(2)
timer.	getitimer, setitimer: get/set value of interval	getitimer(2)
	getlog: get user's login name.	getlog(3F)
	getlogin: get login name.	getlogin(3)
get network entry. getnetent,	getnetbyaddr, getnetbyname, setnetent, endnetent:	getnetent(3n)
entry. getnetent, getnetbyaddr,	getnetbyname, setnetent, endnetent: get network	getnetent(3n)
endnetent: get network entry.	getnetent, getnetbyaddr, getnetbyname, setnetent,	getnetent(3n)
	getpagesize: get system page size.	getpagesize(2)
	getpass: read a password.	getpass(3)
	getpeername: get name of connected peer.	getpeername(2)
	getpgrp: get process group.	getpgrp(2)
	getpid: get process id.	getpid(3F)
	getpid, getppid: get process identification.	getpid(2)
getpid,	getppid: get process identification.	getpid(2)
scheduling priority.	getpriority, setpriority: get/set program	getpriority(2)
protocol entry. getprotoent, getprotobynumber,	getprotobyname, setprotoent, endprotoent: get	getprotoent(3n)
endprotoent: get protocol entry. getprotoent,	getprotobynumber, getprotobyname, setprotoent,	getprotoent(3n)
setprotoent, endprotoent: get protocol entry.	getprotoent, getprotobynumber, getprotobyname,	getprotoent(3n)
	getpw: get name from uid.	getpw(3C)
get password file entry.	getpwent, getpwuid, getpwnam, setpwent, endpwent:	getpwent(3)
entry. getpwent, getpwuid,	getpwnam, setpwent, endpwent: get password file	getpwent(3)
password file entry. getpwent,	getpwuid, getpwnam, setpwent, endpwent: get	getpwent(3)
resource consumption.	getrlimit, setrlimit: control maximum system	getrlimit(2)
utilization.	getrusage: get information about resource	getrusage(2)
	gets, fgets: get a string from a stream.	gets(3S)
entry. getservent, getservbyport,	getservbyname, setservent, endservent: get service	getservent(3n)
endservent: get service entry. getservent,	getservbyport, getservbyname, setservent,	getservent(3n)
setservent, endservent: get service entry.	getservent, getservbyport, getservbyname,	getservent(3n)
gettimeofday, settimeofday:	get/set date and time.	gettimeofday(2)
gethostname, sethostname:	get/set name of current host.	gethostname(2)
getpriority, setpriority:	get/set program scheduling priority.	getpriority(2)
gethostid, sethostid:	get/set unique identifier of current host.	gethostid(2)
getitimer, setitimer:	get/set value of interval timer.	getitimer(2)
	getsockname: get socket name.	getsockname(2)
sockets.	getsockopt, setsockopt: get and set options on	getsockopt(2)
	gettable: get NIC format host tables from a host.	gettable(8C)
	gettimeofday, settimeofday: get/set date and time.	gettimeofday(2)
	getty: set terminal mode.	getty(8)
	gettytab: terminal configuration data base.	gettytab(5)
	getuid, geteuid: get user identity.	getuid(2)
	getuid, getgid: get user or group ID of the caller.	getuid(3F)
getc, getchar, fgetc,	getw: get character or word from stream.	getc(3S)

	getwd: get current working directory pathname.	getwd(3)
head:	give first few lines.	head(1)
shutdown: close down the system at a	given time.	shutdown(8)
	glob: filename expand argument list.	csh(1)
ASCII. ctime, localtime,	gmtime, asctime, timezone: convert date and time to	ctime(3)
time, ctime, ltime,	gmtime: return system time.	time(3F)
fish: play	"Go Fish".	fish(6)
setjmp, longjmp: non-local	goto.	setjmp(3)
	goto: command transfer.	csh(1)
	gprof: display call graph profile data.	gprof(1)
graph: draw a	graph.	graph(1G)
	graph: draw a graph.	graph(1G)
gprof: display call	graph profile data.	gprof(1)
plot:	graphics filters.	plot(1G)
arc, move, cont, point, linemod, space, closepl:	graphics interface. /erase, label, line, circle,	plot(3X)
plot:	graphics interface.	plot(5)
lib2648: subroutines for the HP 2648	graphics terminal.	lib2648(3X)
	grep, egrep, fgrep: search a file for a pattern.	grep(1)
vgrind:	grind nice listings of programs.	vgrind(1)
chgrp: change	group.	chgrp(1)
getpgrp: get process	group.	getpgrp(2)
killpg: send signal to a process	group.	killpg(2)
setpgrp: set process	group.	setpgrp(2)
getgroups: get	group access list.	getgroups(2)
initgroups: initialize	group access list.	initgroups(3X)
setgroups: set	group access list.	setgroups(2)
group:	group file.	group(5)
getgrgid, getgrnam, setgrent, endgrent: get	group file entry. getgrent,	getgrent(3)
	group: group file.	group(5)
setregid: set real and effective	group ID.	setregid(2)
setruid, setgid, setegid, setrgid: set user and	group ID. setuid, seteuid,	setuid(3)
getuid, getgid: get user or	group ID of the caller.	getuid(3F)
getgid, getegid: get	group identity.	getgid(2)
groups: show	group memberships.	groups(1)
chown: change owner and	group of a file.	chown(2)
make: maintain program	groups.	make(1)
	groups: show group memberships.	groups(1)
worm: Play the	growing worm game.	worm(6)
stty,	gtty: set and get terminal state (defunct).	stty(3C)
stop:	halt a job or process.	csh(1)
reboot: reboot system or	halt processor.	reboot(2)
	halt: stop the processor.	halt(8)
rmail:	handle remote mail received via uucp.	rmail(1)
re_comp, re_exec: regular expression	handler.	regex(3)
hangman: Computer version of the game	hangman.	hangman(6)
	hangman: Computer version of the game hangman.	hangman(6)
vhangup: virtually	"hangup" the current control terminal.	vhangup(2)
nohup: run command immune to	hangups.	csh(1)
crash: what	happens when the system crashes.	crash(8V)
park: program to park the	hard disk heads.	park(8)
link: make a	hard link to a file.	link(2)
rehash: recompute command	hash table.	csh(1)
unhash: discard command	hash table.	csh(1)
hashstat: print command	hashing statistics.	csh(1)
	hashstat: print command hashing statistics.	csh(1)
leave: remind you when you	have to leave.	leave(1)
park: program to park the hard disk	heads.	park(8)
od: octal, decimal,	hex, ascii dump.	od(1)
	hier: file system hierarchy.	hier(7)
hier: file system	hierarchy.	hier(7)
history: print	history event list.	csh(1)
	history: print history event list.	csh(1)
fortune: print a random,	hopefully interesting, adage.	fortune(6)
sethostid: get/set unique identifier of current	host. gethostid,	gethostid(2)
gethostname, sethostname: get/set name of current	host.	gethostname(2)
gettable: get NIC format host tables from a	host.	gettable(8C)
hostnm: get name of current	host.	hostnm(3F)
uusend: send a file to a remote	host.	uusend(1C)
htonl, htons, ntohl, ntohs: convert values between	host and network byte order.	byteorder(3n)
remote: remote	host description file.	remote(5)
gethostbyname, sethostent, endhostent: get network	host entry. gethostent, gethostbyaddr,	gethostent(3n)
hosts:	host name data base.	hosts(5)
phones: remote	host phone number data base.	phones(5)
ruptime: show	host status of local machines.	ruptime(1C)
hostid: set or print identifier of current	host system.	hostid(1)

hostname: set or print name of current	host system.	hostname(1)
htable: convert NIC standard format	host tables.	htable(8)
gettable: get NIC format	host tables from a host.	gettable(8C)
system.	hostid: set or print identifier of current host	hostid(1)
	hostname: set or print name of current host system.	hostname(1)
	hostnm: get name of current host.	hostnm(3F)
ping: send ICMP ECHO_REQUEST packets to network	hosts.	ping(8)
	hosts: host name data base.	hosts(5)
uptime: show	how long system has been up.	uptime(1)
TEACHJOVE - learn	how to use the JOVE editor.	teachjove(1)
lib2648: subroutines for the	HP 2648 graphics terminal.	lib2648(3X)
	htable: convert NIC standard format host tables.	htable(8)
host and network byte order.	htonl, htons, ntohl, ntohs: convert values between	byteorder(3n)
and network byte order. htonl,	htons, ntohl, ntohs: convert values between host	byteorder(3n)
	hunt: a multi-player multi-terminal game.	hunt(6)
wump: the game of	hunt-the-wumpus.	wump(6)
sinh, cosh, tanh:	hyperbolic functions.	sinh(3M)
	hypot, cabs: Euclidean distance.	hypot(3M)
getarg,	iargc: return command line arguments.	getarg(3F)
	icheck: file system storage consistency check.	icheck(8)
ping: send	ICMP ECHO_REQUEST packets to network hosts.	ping(8)
getpid: get process	id.	getpid(3F)
a message queue, semaphore set or shared memory	id. ipcrm: remove	ipcrm(1)
setregid: set real and effective group	ID.	setregid(2)
setgid, setegid, setrgid: set user and group	ID. setuid, seteuid, setruid,	setuid(3)
whoami: print effective current user	id.	whoami(1)
getuid, getgid: get user or group	ID of the caller.	getuid(3F)
su: substitute user	id temporarily.	su(1)
form.	idate, itime: return date or time in numerical	idate(3F)
	ident: identify files.	ident(1)
getpid, getppid: get process	identification.	getpid(2)
gethostid, sethostid: get/set unique	identifier of current host.	gethostid(2)
hostid: set or print	identifier of current host system.	hostid(1)
ident:	identify files.	ident(1)
getgid, getegid: get group	identity.	getgid(2)
getuid, geteuid: get user	identity.	getuid(2)
setreuid: set real and effective user	ID's.	setreuid(2)
perror, gerror,	ierrno: get system error messages.	perror(3F)
	if: conditional statement.	csh(1)
biff: be notified	if mail arrives and who it is from.	biff(1)
eval, exec, exit, export, login,/ sh, for, case,	if, while, : , . , break, continue, cd,	sh(1)
	ifconfig: configure network interface parameters.	ifconfig(8C)
unifdef: remove	ifdef'ed lines.	unifdef(1)
abort: terminate abruptly with memory	image.	abort(3F)
core: format of memory	image file.	core(5)
gcore: get core	images of running processes.	gcore(1)
notify: request	immediate notification.	csh(1)
nohup: run command	immune to hangups.	csh(1)
implog:	IMP log interpreter.	implog(8C)
implogd:	IMP logger process.	implogd(8C)
xstr: extract strings from C programs to	implement shared strings.	xstr(1)
	implog: IMP log interpreter.	implog(8C)
	implogd: IMP logger process.	implogd(8C)
eyacc: modified yacc allowing much	improved error recovery.	eyacc(1)
which: locate a program file	including aliases and paths (*csh* only).	which(1)
fsync: synchronize a file's	in-core state with that on disk.	fsync(2)
dump, dumpdates:	incremental dump format.	dump(5)
dump:	incremental file system dump.	dump(8)
restore:	incremental file system restore.	restore(8)
indent:	indent and format C program source.	indent(1)
	indent: indent and format C program source.	indent(1)
tgetnum, tgetflag, tgetstr, tgoto, tputs: terminal	independent operation routines. tgetent,	termcap(3X)
ptx: permuted	index.	ptx(1)
bibliography. indxbib, lookbib: build inverted	index for a bibliography, find references in a	lookbib(1)
objects.	index, rindex, lnblnk, len: tell about character	index(3F)
strncat, strcmp, strncmp, strcpy, strncpy, strlen,	index, rindex: string operations. strcat,	string(3)
last:	indicate last logins of users and teletypes.	last(1)
syscall:	indirect system call.	syscall(2)
fsplit: split a multi-routine Fortran file into	individual files.	fsplit(1)
bibliography, find references in a bibliography.	indxbib, lookbib: build inverted index for a	lookbib(1)
inet_lnaof, inet_netof: Internet address/	inet_addr, inet_network, inet_ntoa, inet_makeaddr,	inet(3n)
inet_addr, inet_network, inet_ntoa, inet_makeaddr,	inet_lnaof, inet_netof: Internet address/	inet(3n)
address/ inet_addr, inet_network, inet_ntoa,	inet_makeaddr, inet_lnaof, inet_netof: Internet	inet(3n)
/inet_network, inet_ntoa, inet_makeaddr, inet_lnaof,	inet_netof: Internet address manipulation routines.	inet(3n)
inet_netof: Internet address/ inet_addr,	inet_network, inet_ntoa, inet_makeaddr, inet_lnaof,	inet(3n)

Internet address/ inet_addr, inet_network,	inet_ntoa, inet_makeaddr, inet_lnaof, inet_netof:	inet(3n)
dumpfs: dump file system	information.	dumpfs(8)
mttys: Multi-Link partition	information.	mttys(5)
pac: printer/plotter accounting	information.	pac(8)
whodos: display	information about dosc users.	whodos(1)
rlog: print log messages and other	information about RCS files.	rlog(1)
getrusage: get	information about resource utilization.	getrusage(2)
vtimes: get	information about resource utilization.	vtimes(3C)
fstab: static	information about the filesystems.	fstab(5)
man: find manual	information by keywords; print out the manual.	man(1)
dosdisk: program to create and display	information for MPS/DOS vdisks.	dosdisk(8)
finger: user	information lookup program.	finger(1)
miscellaneous: miscellaneous useful	information pages.	intro(7)
	init: process control initialization.	init(8)
	initgroups: initialize group access list.	initgroups(3X)
init: process control	initialization.	init(8)
ioinit: change f77 I/O	initialization.	ioinit(3F)
tset: terminal dependent	initialization.	tset(1)
ttys: terminal	initialization data.	ttys(5)
initgroups:	initialize group access list.	initgroups(3X)
connect:	initiate a connection on a socket.	connect(2)
popen, pclose:	initiate I/O to/from a process.	popen(3)
generator; routines for changing/ random, srandom,	initstate, setstate: better random number	random(3)
flmin, flmax, ffrac, dflmin, dflmax, dffrac,	inmax: return extreme values.	flmin(3F)
clri: clear	i-node.	clri(8)
fs,	inode: format of file system volume.	fs(5)
read, readv: read	input.	read(2)
soelim: eliminate .so's from nroff	input.	soelim(1)
scanf, fscanf, sscanf: formatted	input conversion.	scanf(3S)
ungetc: push character back into	input stream.	ungetc(3S)
fread, fwrite: buffered binary	input/output.	fread(3S)
stdio: standard buffered	input/output package.	intro(3S)
ferror, feof, clearerr, fileno: stream status	inquiries.	ferror(3S)
refer: find and	insert literature references in documents.	refer(1)
insque, remque:	insert/remove element from a queue.	insque(3)
	insque, remque: insert/remove element from a queue.	insque(3)
install:	install binaries.	install(1)
binstl: program to	install bootloader on disk.	binstl(8)
	install: install binaries.	install(1)
learn: computer aided	instruction about UNIX.	learn(1)
doctor:	interact with a psychoanalyst.	doctor(6)
jove: an	interactive display-oriented text editor.	jove(1)
fsck: file system consistency check and	interactive repair.	fsck(8)
fortune: print a random, hopefully	interesting, adage.	fortune(6)
cont. point, linemod, space, closepl: graphics	interface. /erase, label, line, circle, arc, move,	plot(3X)
plot: graphics	interface.	plot(5)
tty: general terminal	interface.	tty(4)
ifconfig: configure network	interface parameters.	ifconfig(8C)
telnet: user	interface to the TELNET protocol.	telnet(1C)
slattach: attach serial lines as network	interfaces.	slattach(8C)
sendmail: send mail over the	internet.	sendmail(8)
/inet_ntoa, inet_makeaddr, inet_lnaof, inet_netof:	Internet address manipulation routines.	inet(3n)
ftpd: DARPA	Internet File Transfer Protocol server.	ftpd(8C)
spline:	interpolate smooth curve.	spline(1G)
implog: IMP log	interpreter.	implog(8C)
lisp: lisp	interpreter.	lisp(1)
pti: phototypesetter	interpreter.	pti(1)
px: Pascal	interpreter.	px(1)
pix: Pascal	interpreter and executor.	pix(1)
pi: Pascal	interpreter code translator.	pi(1)
csh: a shell (command	interpreter) with C-like syntax.	csh(1)
pipe: create an	interprocess communication channel.	pipe(2)
ipcs: report	inter-process communication facilities status.	ipcs(1)
atomically release blocked signals and wait for	interrupt. sigpause:	sigpause(2)
onintr: process	interrupts in command scripts.	csh(1)
intro:	introduction to commands.	intro(1)
intro:	introduction to compatibility library functions.	intro(3C)
intro:	introduction to FORTRAN library functions.	intro(3F)
intro:	introduction to library functions.	intro(3)
intro:	introduction to mathematical library functions.	intro(3M)
intro:	introduction to miscellaneous library functions.	intro(3X)
intro:	introduction to network library functions.	intro(3n)
rcsintro:	introduction to RCS commands.	rcsintro(1)
intro:	introduction to system calls and error numbers.	intro(2)
commands. intro:	introduction to system maintenance and operation	intro(8)

ncheck: generate names from	i-numbers.	ncheck(8)
in a bibliography. indxbib, lookbib: build	inverted index for a bibliography, find references	lookbib(1)
tread, twrite, trewin, tskipf, tstate: f77 tape	I/O. topen, tclose,	topen(3F)
ioinit: change f77	I/O initialization.	ioinit(3F)
select: synchronous	i/o multiplexing.	select(2)
iostat: report	I/O statistics.	iostat(1)
popen, pclose: initiate	I/O to/from a process.	popen(3)
	ioctl: control device.	ioctl(2)
	ioinit: change f77 I/O initialization.	ioinit(3F)
	iostat: report I/O statistics.	iostat(1)
shared memory id.	ipcrm: remove a message queue, semaphore set or	ipcrm(1)
status.	ipcs: report inter-process communication facilities	ipcs(1)
rand, drand,	irand: return random values.	rand(3F)
isascii:/ isalpha, isupper, islower, isdigit,	isalnum, isspace, ispunct, isprint, iscntrl,	ctype(3)
isspace, ispunct, isprint, iscntrl, isascii:/	isalpha, isupper, islower, isdigit, isalnum,	ctype(3)
isalnum, isspace, ispunct, isprint, iscntrl,	isascii: character classification macros. /isdigit,	ctype(3)
ttynam,	isatty: find name of a terminal port.	ttynam(3F)
ttyname,	isatty, ttyslot: find name of a terminal.	ttyname(3)
/isdigit, isalnum, isspace, ispunct, isprint,	iscntrl, isascii: character classification macros.	ctype(3)
iscntrl, isascii:/ isalpha, isupper, islower,	isdigit, isalnum, isspace, ispunct, isprint,	ctype(3)
isprint, iscntrl, isascii:/ isalpha, isupper,	islower, isdigit, isalnum, isspace, ispunct,	ctype(3)
/islower, isdigit, isalnum, isspace, ispunct,	isprint, iscntrl, isascii: character classification/	ctype(3)
/isupper, islower, isdigit, isalnum, isspace,	ispunct, isprint, iscntrl, isascii: character/	ctype(3)
isalpha, isupper, islower, isdigit, isalnum,	isspace, ispunct, isprint, iscntrl, isascii:/	ctype(3)
system:	issue a shell command.	system(3)
ispunct, isprint, iscntrl, isascii:/ isalpha,	isupper, islower, isdigit, isalnum, isspace,	ctype(3)
idate,	itime: return date or time in numerical form.	idate(3F)
	j0, j1, jn, y0, y1, yn: bessel functions.	j0(3M)
j0,	j1, jn, y0, y1, yn: bessel functions.	j0(3M)
j0, j1,	jn, y0, y1, yn: bessel functions.	j0(3M)
bg: place	job in background.	csh(1)
fg: bring	job into foreground.	csh(1)
jobs: print current	job list.	csh(1)
stop: halt a	job or process.	csh(1)
kill: kill	jobs and processes.	csh(1)
lprm: remove	jobs from the line printer spooling queue.	lprm(1)
	jobs: print current job list.	csh(1)
	join: relational database operator.	join(1)
	jove: an interactive display-oriented text editor.	jove(1)
jove_recover - recover	JOVE buffers after a system/editor crash.	jove_recover(1)
TEACHJOVE - learn how to use the	JOVE editor.	teachjove(1)
system/editor crash.	jove_recover - recover JOVE buffers after a	jove_recover(1)
msgs: system messages and	junk mail program.	msgs(1)
kermit:	kermit file transfer.	kermit(1)
	kermit: kermit file transfer.	kermit(1)
uxrc: configuration file for	kernel.	uxrc(8)
vers: print version number of the	kernel.	vers(1)
makekey: generate encryption	key.	makekey(8)
apropos: locate commands by	keyword lookup.	apropos(1)
man: find manual information by	keywords; print out the manual.	man(1)
profile buffers.	kgmon: generate a dump of the operating system's	kgmon(8)
kill:	kill jobs and processes.	csh(1)
	kill: kill jobs and processes.	csh(1)
	kill: send a signal to a process.	kill(3F)
	kill: send signal to a process.	kill(2)
	kill: terminate a process with extreme prejudice.	kill(1)
	killpg: send signal to a process group.	killpg(2)
bessel functions: of two	kinds for integer orders.	bessel(3F)
dmem,	kmem: main memory.	mem(4)
linemod, space, closepl:/ plot: openpl, erase,	label, line, circle, arc, move, cont, point,	plot(3X)
awk: pattern scanning and processing	language.	awk(1)
bc: arbitrary-precision arithmetic	language.	bc(1)
efl: Extended Fortran	Language.	efl(1)
set, shift, times, trap, umask, wait: command	language. /exit, export, login, read, readonly,	sh(1)
fp: Functional Programming	language compiler/interpreter.	fp(1)
vgrindefs: vgrind's	language definition data base.	vgrindefs(5)
order.	lastcomm: show last commands executed in reverse	lastcomm(1)
	ld: link editor.	ld(1)
frexp,	ldexp, modf: split into mantissa and exponent.	frexp(3)
	learn: computer aided instruction about UNIX.	learn(1)
TEACHJOVE -	learn how to use the JOVE editor.	teachjove(1)
leave: remind you when you have to	leave.	leave(1)
	leave: remind you when you have to leave.	leave(1)
exit:	leave shell.	csh(1)
index, rindex, lnblnk,	len: tell about character objects.	index(3F)

and time to ASCII. ctime,	localtime, gmtime, asctime, timezone: convert date	ctime(3)
(*csh* only). which:	locate a program file including aliases and paths	which(1)
apropos:	locate commands by keyword lookup.	apropos(1)
whereis:	locate source, binary, and or manual for program.	whereis(1)
end, etext, edata: last	locations in program.	end(3)
flock: apply or remove an advisory	lock on an open file.	flock(2)
	lock: reserve a terminal.	lock(1)
	lockf: record locking on files.	lockf(3C)
lockf: record	locking on files.	lockf(3C)
collect system diagnostic messages to form error	log. dmesg:	dmesg(8)
syslog, openlog, closelog: control system	log.	syslog(3)
gamma:	log gamma function.	gamma(3M)
implog: IMP	log interpreter.	implog(8C)
power, square root. exp,	log, log10, pow, sqrt: exponential, logarithm,	exp(3M)
rlog: print	log messages and other information about RCS files.	rlog(1)
syslog:	log systems messages.	syslog(8)
square root. exp, log,	log10, pow, sqrt: exponential, logarithm, power,	exp(3M)
exp, log, log10, pow, sqrt: exponential,	logarithm, power, square root.	exp(3M)
rwho: who's	logged in on local machines.	rwho(1C)
implogd: IMP	logger process.	implogd(8C)
flush: flush output to a	logical unit.	flush(3F)
fseek, ftell: reposition a file on a	logical unit.	fseek(3F)
getc, fgetc: get a character from a	logical unit.	getc(3F)
putc, fputc: write a character to a fortran	logical unit.	putc(3F)
rlogin: remote	login.	rlogin(1C)
ac:	login accounting.	ac(8)
	login: login new user.	csh(1)
getlog: get user's	login name.	getlog(3F)
getlogin: get	login name.	getlogin(3)
login:	login new user.	csh(1)
passwd: change	login password.	passwd(1)
/break, continue, cd, eval, exec, exit, export,	login, read, readonly, set, shift, times, trap,/	sh(1)
utmp, wtmp:	login records.	utmp(5)
rlogind: remote	login server.	rlogind(8C)
chsh: change default	login shell.	chsh(1)
	login: sign on.	login(1)
last: indicate last	logins of users and teletypes.	last(1)
	logout: end session.	csh(1)
setjmp,	longjmp: non-local goto.	setjmp(3)
	look: find lines in a sorted list.	look(1)
find references in a bibliography. indxbib,	lookbib: build inverted index for a bibliography,	lookbib(1)
apropos: locate commands by keyword	lookup.	apropos(1)
finger: user information	lookup program.	finger(1)
break: exit while/foreach	loop.	csh(1)
continue: cycle in	loop.	csh(1)
end: terminate	loop.	csh(1)
foreach:	loop over list of names.	csh(1)
library.	lorder: find ordering relation for an object	lorder(1)
mklost+found: make a	lost+found directory for fsck.	mklost+found(8)
	lpc: line printer control program.	lpc(8)
	lpd: line printer daemon.	lpd(8)
	lpq: spool queue examination program.	lpq(1)
	lpr: off line print.	lpr(1)
queue.	lprm: remove jobs from the line printer spooling	lprm(1)
	ls: list contents of directory.	ls(1)
	lseek: move read/write pointer.	lseek(2)
bit: and, or, xor, not, rshift,	lshift bitwise functions.	bit(3F)
stat,	lstat, fstat: get file status.	stat(2)
stat,	lstat, fstat: get file status.	stat(3F)
time, ctime,	ltime, gmtime: return system time.	time(3F)
	lxref: lisp cross reference program.	lxref(1)
	m4: macro processor.	m4(1)
as:	M68020 assembler.	as(1)
Standalone mode: definition of this Sanyo/ICON	machine operation mode..	standalone(8)
ruptime: show host status of local	machines.	ruptime(1C)
rwho: who's logged in on local	machines.	rwho(1C)
m4:	macro processor.	m4(1)
alias: shell	macros.	csh(1)
isprint, iscntrl, isascii: character classification	macros. /isdigit, isalnum, isspace, ispunct,	ctype(3)
ms: text formatting	macros.	ms(7)
translate version 6 manual macros to version 7	macros. trman:	trman(1)
me:	macros for formatting papers.	me(7)
man:	macros to typeset manual.	man(7)
trman: translate version 6 manual	macros to version 7 macros.	trman(1)
tcopy: copy a	mag tape.	tcopy(1)

mt:	magnetic tape manipulating program.	mt(1)
rmt: remote	magtape protocol module.	rmt(8C)
mail: send and receive	mail.	mail(1)
encode/decode a binary file for transmission via	mail. uuencode,uudecode:	uuencode(1C)
xsend, xget, enroll: secret	mail.	xsend(1)
sendbug:	mail a system bug report to 4bsd-bugs.	sendbug(1)
mailaddr:	mail addressing description.	mailaddr(7)
newaliases: rebuild the data base for the	mail aliases file.	newaliases(1)
binmail: send or receive	mail among users.	binmail(1)
biff: be notified if	mail arrives and who it is from.	biff(1)
from: who is my	mail from?.	from(1)
prmail: print out	mail in the post office.	prmail(1)
tmail: print out	mail messages, most recent first.	tmail(1)
sendmail: send	mail over the internet.	sendmail(8)
msgs: system messages and junk	mail program.	msgs(1)
rmail: handle remote	mail received via uucp.	rmail(1)
	mail: send and receive mail.	mail(1)
	mailaddr: mail addressing description.	mailaddr(7)
dmem, kmem:	main memory.	mem(4)
make:	maintain program groups.	make(1)
ar: archive and library	maintainer.	ar(1)
intro: introduction to system	maintenance and operation commands.	intro(8)
mkdir:	make a directory.	mkdir(1)
mkdir:	make a directory file.	mkdir(2)
link:	make a hard link to a file.	link(2)
link:	make a link to an existing file.	link(3F)
mklost+found:	make a lost+found directory for fsck.	mklost+found(8)
mknod:	make a special file.	mknod(2)
mktemp:	make a unique file name.	mktemp(3)
ln:	make links.	ln(1)
	make: maintain program groups.	make(1)
symlink:	make symbolic link to a file.	symlink(2)
makedev:	make system special files.	makedev(8)
script:	make typescript of terminal session.	script(1)
mkfs: program to	make UNIX file systems.	mkfs(8)
	makedev: make system special files.	makedev(8)
	makekey: generate encryption key.	makekey(8)
allocator.	malloc, free, realloc, calloc, alloca: memory	malloc(3)
the manual.	man: find manual information by keywords; print out	man(1)
	man: macros to typeset manual.	man(7)
shift:	manipulate argument list.	csh(1)
quota:	manipulate disk quotas.	quota(2)
route: manually	manipulate the routing tables.	route(8C)
mt: magnetic tape	manipulating program.	mt(1)
inet_lnaof, inet_netof: Internet address	manipulation routines. /inet_ntoa, inet_makeaddr,	inet(3n)
frexp, ldexp, modf: split into	mantissa and exponent.	frexp(3)
catman: create the cat files for the	manual.	catman(8)
find manual information by keywords; print out the	manual. man:	man(1)
man: macros to typeset	manual.	man(7)
whereis: locate source, binary, and or	manual for program.	whereis(1)
manual. man: find	manual information by keywords; print out the	man(1)
trman: translate version 6	manual macros to version 7 macros.	trman(1)
route:	manually manipulate the routing tables.	route(8C)
umask: change or display file creation	mask.	csh(1)
sigsetmask: set current signal	mask.	sigsetmask(2)
umask: set file creation mode	mask.	umask(2)
mkstr: create an error message file by	massaging C source.	mkstr(1)
intro: introduction to	mathematical library functions.	intro(3M)
eqn, neqn, checkeq: typeset	mathematics.	eqn(1)
getrlimit, setrlimit: control	maximum system resource consumption.	getrlimit(2)
vlimit: control	maximum system resource consumption.	vlimit(3C)
dis: an	mc68020 disassembler.	dis(1)
	me: macros for formatting papers.	me(7)
bcd: convert to antique	media.	bcd(6)
groups: show group	memberships.	groups(1)
dmem, kmem: main	memory.	mem(4)
malloc, free, realloc, calloc, alloca:	memory allocator.	malloc(3)
valloc: aligned	memory allocator.	valloc(3)
shmctl: shared	memory control operations.	shmctl(2)
vfork: spawn new process in a virtual	memory efficient way.	vfork(2)
remove a message queue, semaphore set or shared	memory id. ipcrm:	ipcrm(1)
abort: terminate abruptly with	memory image.	abort(3F)
core: format of	memory image file.	core(5)
shmop: shared	memory operations.	shmop(2)
shmget: get shared	memory segment.	shmget(2)

vmstat: report virtual	memory statistics.	vmstat(1)
merge: three-way file	merge.	merge(1)
sort: sort or	merge files.	sort(1)
rcsmerge:	merge RCS revisions.	rcsmerge(1)
	merge: three-way file merge.	merge(1)
pmerge: pascal file	merger.	pmerge(1)
	mesg: permit or deny messages.	mesg(1)
msgctl:	message control operations.	msgctl(2)
mkstr: create an error	message file by massaging C source.	mkstr(1)
recv, recvfrom, recvmsg: receive a	message from a socket.	recv(2)
send, sendto, sendmsg: send a	message from a socket.	send(2)
msgop:	message operations.	msgop(2)
msgget: get	message queue.	msgget(2)
ipcrm: remove a	message queue, semaphore set or shared memory id.	ipcrm(1)
error: analyze and disperse compiler error	messages.	error(1)
mesg: permit or deny	messages.	mesg(1)
perror, sys_errlist, sys_nerr: system error	messages.	perror(3)
perror, gerror, ierrno: get system error	messages.	perror(3F)
psignal, sys_siglist: system signal	messages.	psignal(3)
syslog: log systems	messages.	syslog(8)
msgs: system	messages and junk mail program.	msgs(1)
rlog: print log	messages and other information about RCS files.	rlog(1)
tmail: print out mail	messages, most recent first.	tmail(1)
dmesg: collect system diagnostic	messages to form error log.	dmesg(8)
mille: play	Mille Bournes.	mille(6)
	mille: play Mille Bournes.	mille(6)
intro: introduction to	miscellaneous library functions.	intro(3X)
pages.	miscellaneous: miscellaneous useful information	intro(7)
miscellaneous:	miscellaneous useful information pages.	intro(7)
	mkdir: make a directory.	mkdir(1)
	mkdir: make a directory file.	mkdir(2)
	mkfs: program to make UNIX file systems.	mkfs(8)
	mklost+found: make a lost+found directory for fsck.	mklost+found(8)
	mknod: build special file.	mknod(8)
	mknod: make a special file.	mknod(2)
	mkproto: construct a prototype file system.	mkproto(8)
source.	mkstr: create an error message file by massaging C	mkstr(1)
	mktemp: make a unique file name.	mktemp(3)
chmod: change	mode.	chmod(1)
getty: set terminal	mode.	getty(8)
definition of this Sanyo/ICON machine operation	mode.. Standalone mode:	standalone(8)
operation mode.. Standalone	mode: definition of this Sanyo/ICON machine	standalone(8)
umask: set file creation	mode mask.	umask(2)
chmod: change	mode of a file.	chmod(3F)
chmod: change	mode of file.	chmod(2)
frexp, ldexp,	modf: split into mantissa and exponent.	frexp(3)
touch: update date last	modified of a file.	touch(1)
recovery. eyacc:	modified yacc allowing much improved error	eyacc(1)
rmt: remote magtape protocol	module.	rmt(8C)
what: show what versions of object	modules were used to construct a file.	what(1)
monitor, monstartup,	moncontrol: prepare execution profile.	monitor(3)
profile.	monitor, monstartup, moncontrol: prepare execution	monitor(3)
	monop: Monopoly game.	monop(6)
monop:	Monopoly game.	monop(6)
monitor,	monstartup, moncontrol: prepare execution profile.	monitor(3)
	more, page: file perusal filter for crt viewing.	more(1)
tmail: print out mail messages,	most recent first.	tmail(1)
curses: screen functions with "optimal" cursor	motion.	curses(3X)
mount, umount:	mount and dismount file system.	mount(8)
mount, umount:	mount or remove file system.	mount(2)
	mount, umount: mount and dismount file system.	mount(8)
	mount, umount: mount or remove file system.	mount(2)
mtab:	mounted file system table.	mtab(5)
plot: openpl, erase, label, line, circle, arc,	move, cont, point, linemod, space, closepl:/	plot(3X)
mv:	move or rename files.	mv(1)
lseek:	move read/write pointer.	lseek(2)
doscopyd:	MPS/DOS file copy daemon.	doscopyd(8)
dosprint:	MPS/DOS spooler daemon.	dosprint(8)
program to create and display information for	MPS/DOS vdisks. dosdisk:	dosdisk(8)
dosdisks: list of	MPS/DOS virtual disks.	dosdisks(5)
	ms: text formatting macros.	ms(7)
	msgctl: message control operations.	msgctl(2)
	msgget: get message queue.	msgget(2)
	msgop: message operations.	msgop(2)
	msgs: system messages and junk mail program.	msgs(1)

	mt: magnetic tape manipulating program.	mt(1)
	mtab: mounted file system table.	mtab(5)
	mttys: Multi-Link partition information.	mttys(5)
eyacc: modified yacc allowing	much improved error recovery.	eyacc(1)
mttys:	Multi-Link partition information.	mttys(5)
hunt: a	multi-player multi-terminal game.	hunt(6)
select: synchronous i/o	multiplexing.	select(2)
fsplit: split a	multi-routine Fortran file into individual files.	fsplit(1)
hunt: a multi-player	multi-terminal game.	hunt(6)
switch:	multi-way command branch.	csh(1)
	mv: move or rename files.	mv(1)
from: who is	my mail from?.	from(1)
getdiskbyname: get disk description by its	name.	getdisk(3X)
getenv: value for environment	name.	getenv(3)
getlog: get user's login	name.	getlog(3F)
getlogin: get login	name.	getlogin(3)
getsockname: get socket	name.	getsockname(2)
mktemp: make a unique file	name.	mktemp(3)
pwd: working directory	name.	pwd(1)
tty: get terminal	name.	tty(1)
hosts: host	name data base.	hosts(5)
networks: network	name data base.	networks(5)
protocols: protocol	name data base.	protocols(5)
services: service	name data base.	services(5)
getpw: get	name from uid.	getpw(3C)
nlist: get entries from	name list.	nlist(3)
nm: print	name list.	nm(1)
symorder: rearrange	name list.	symorder(1)
rename: change the	name of a file.	rename(2)
ttyname, isatty, ttyslot: find	name of a terminal.	ttyname(3)
ttynam, isatty: find	name of a terminal port.	ttynam(3F)
getpeername: get	name of connected peer.	getpeername(2)
gethostname, sethostname: get/set	name of current host.	gethostname(2)
hostnm: get	name of current host.	hostnm(3F)
hostname: set or print	name of current host system.	hostname(1)
.PP dstrules: Daylight savings time and time zone	name rule file..	dstrules(5)
bind: bind a	name to a socket.	bind(2)
foreach: loop over list of	names.	csh(1)
term: conventional	names for terminals.	term(7)
ncheck: generate	names from i-numbers.	ncheck(8)
	ncheck: generate names from i-numbers.	ncheck(8)
eqn,	neqn, checkeq: typeset mathematics.	eqn(1)
	netstat: show network status.	netstat(1)
rdump: file system dump across the	network.	rdump(8C)
rrestore: restore a file system dump across the	network.	rrestore(8C)
ntohl, ntohs: convert values between host and	network byte order. htonl, htons,	byteorder(3n)
getnetbyname, setnetent, endnetent: get	network entry. getnetent, getnetbyaddr,	getnetent(3n)
gethostbyname, sethostent, endhostent: get	network host entry. gethostent, gethostbyaddr,	gethostent(3n)
ping: send ICMP ECHO_REQUEST packets to	network hosts.	ping(8)
ifconfig: configure	network interface parameters.	ifconfig(8C)
slattach: attach serial lines as	network interfaces.	slattach(8C)
intro: introduction to	network library functions.	intro(3n)
networks:	network name data base.	networks(5)
routed:	network routing daemon.	routed(8C)
netstat: show	network status.	netstat(1)
	networks: network name data base.	networks(5)
creat: create a	new file.	creat(2)
open a file for reading or writing, or create a	new file. open:	open(2)
newfs: construct a	new file system.	newfs(8)
fork: create a	new process.	fork(2)
vfork: spawn	new process in a virtual memory efficient way.	vfork(2)
login: login	new user.	csh(1)
adduser: procedure for adding	new users.	adduser(8)
aliases file.	newaliases: rebuild the data base for the mail	newaliases(1)
	newfs: construct a new file system.	newfs(8)
dbminit, fetch, store, delete, firstkey,	nextkey: data base subroutines.	dbm(3X)
gettable: get	NIC format host tables from a host.	gettable(8C)
htable: convert	NIC standard format host tables.	htable(8)
vgrind: grind	nice listings of programs.	vgrind(1)
(*sh* only).	nice, nohup: run a command at low priority	nice(1)
	nice: run low priority process.	csh(1)
	nice: set program priority.	nice(3C)
	nlist: get entries from name list.	nlist(3)
	nm: print name list.	nm(1)
only). nice,	nohup: run a command at low priority (*sh*	nice(1)

	nohup: run command immune to hangups.	csh(1)
setjmp, longjmp:	non-local goto.	setjmp(3)
bit: and, or, xor,	not, rshift, lshift bitwise functions.	bit(3F)
notify: request immediate	notification.	csh(1)
biff: be	notified if mail arrives and who it is from.	biff(1)
	notify: request immediate notification.	csh(1)
soelim: eliminate .so's from	nroff input.	soelim(1)
tbl: format tables for	nroff or troff.	tbl(1)
colcrt: filter	nroff output for CRT previewing.	colcrt(1)
	nroff: text formatting.	nroff(1)
troff,	nroff: text formatting and typesetting.	troff(1)
deroff: remove	nroff, troff, tbl and eqn constructs.	deroff(1)
checknr: check	nroff/troff files.	checknr(1)
network byte order. htonl, htons,	ntohl, ntohs: convert values between host and	byteorder(3n)
order. htonl, htons, ntohl,	ntohs: convert values between host and network byte	byteorder(3n)
	null: data sink.	null(4)
	number: convert Arabic numerals to English.	number(6)
phones: remote host phone	number data base.	phones(5)
arithmetic: provide drill in	number facts.	arithmetic(6)
rand, srand: random	number generator.	rand(3C)
random, srandom, initstate, setstate: better random	number generator; routines for changing generators.	random(3)
vers: print version	number of the kernel.	vers(1)
atof, atoi, atol: convert ASCII to	numbers.	atof(3)
intro: introduction to system calls and error	numbers.	intro(2)
number: convert Arabic	numerals to English.	number(6)
idate, itime: return date or time in	numerical form.	idate(3F)
loc: return the address of an	object.	loc(3F)
long, short: integer	object conversion.	long(3F)
size: size of an	object file.	size(1)
lorder: find ordering relation for an	object library.	lorder(1)
what: show what versions of	object modules were used to construct a file.	what(1)
strings: find the printable strings in a	object, or other binary, file.	strings(1)
index, rindex, lnblnk, len: tell about character	objects.	index(3F)
od:	octal, decimal, hex, ascii dump.	od(1)
	od: octal, decimal, hex, ascii dump.	od(1)
prmail: print out mail in the post	office.	prmail(1)
	onintr: process interrupts in command scripts.	csh(1)
nohup: run a command at low priority (*sh*	only). nice,	nice(1)
program file including aliases and paths (*csh*	only). which: locate a	which(1)
file. open:	open a file for reading or writing, or create a new	open(2)
fopen, freopen, fdopen:	open a stream.	fopen(3S)
flock: apply or remove an advisory lock on an	open file.	flock(2)
a new file.	open: open a file for reading or writing, or create	open(2)
closedir: directory operations.	opendir, readdir, telldir, seekdir, rewinddir,	directory(3)
closedir: directory operations. directory:	opendir, readdir, telldir, seekdir, rewinddir,	directory(3X)
syslog,	openlog, closelog: control system log.	syslog(3)
cont, point, linemod, space, closepl:/ plot:	openpl, erase, label, line, circle, arc, move,	plot(3X)
kgmon: generate a dump of the	operating system's profile buffers.	kgmon(8)
intro: introduction to system maintenance and	operation commands.	intro(8)
mode: definition of this Sanyo/ICON machine	operation mode.. Standalone	standalone(8)
tgetstr, tgoto, tputs: terminal independent	operation routines. tgetent, tgetnum, tgetflag,	termcap(3X)
bcopy, bcmp, bzero, ffs: bit and byte string	operations.	bstring(3)
telldir, seekdir, rewinddir, closedir: directory	operations. opendir, readdir,	directory(3)
telldir, seekdir, rewinddir, closedir: directory	operations. directory: opendir, readdir,	directory(3X)
msgctl: message control	operations.	msgctl(2)
msgop: message	operations.	msgop(2)
semctl: semaphore control	operations.	semctl(2)
semop: semaphore	operations.	semop(2)
shmctl: shared memory control	operations.	shmctl(2)
shmop: shared memory	operations.	shmop(2)
strcpy, strncpy, strlen, index, rindex: string	operations. strcat, strncat, strcmp, strncmp,	string(3)
join: relational database	operator.	join(1)
curses: screen functions with	"optimal" cursor motion.	curses(3X)
fcntl: file control	options.	fcntl(5)
stty: set terminal	options.	stty(1)
getsockopt, setsockopt: get and set	options on sockets.	getsockopt(2)
ntohs: convert values between host and network byte	order. htonl, htons, ntohl,	byteorder(3n)
lastcomm: show last commands executed in reverse	order.	lastcomm(1)
tac: concatenate and print files in reverse	order.	tac(1)
lorder: find	ordering relation for an object library.	lorder(1)
bessel functions: of two kinds for integer	orders.	bessel(3F)
vi: screen	oriented (visual) display editor based on ex.	vi(1)
a.out: assembler and link editor	output.	a.out(5)
terminate a process after flushing any pending	output. exit:	exit(3)
ecvt, fcvt, gcvt:	output conversion.	ecvt(3)

stop: halt a job or	process.	csh(1)
_exit: terminate a	process.	exit(2)
fork: create a new	process.	fork(2)
fork: create a copy of this	process.	fork(3F)
implogd: IMP logger	process.	implogd(8C)
kill: send signal to a	process.	kill(2)
kill: send a signal to a	process.	kill(3F)
popen, pclose: initiate I/O to/from a	process.	popen(3)
wait: await completion of	process.	wait(1)
exit: terminate a	process after flushing any pending output.	exit(3)
init:	process control initialization.	init(8)
getpgrp: get	process group.	getpgrp(2)
killpg: send signal to a	process group.	killpg(2)
setpgrp: set	process group.	setpgrp(2)
getpid: get	process id.	getpid(3F)
getpid, getppid: get	process identification.	getpid(2)
vfork: spawn new	process in a virtual memory efficient way.	vfork(2)
onintr:	process interrupts in command scripts.	csh(1)
ps:	process status.	ps(1)
times: get	process times.	times(3C)
wait, wait3: wait for	process to terminate.	wait(2)
wait: wait for a	process to terminate.	wait(3F)
ptrace:	process trace.	ptrace(2)
kill: terminate a	process with extreme prejudice.	kill(1)
exit: terminate	process with status.	exit(3F)
kill: kill jobs and	processes.	csh(1)
gcore: get core images of running	processes.	gcore(1)
renice: alter priority of running	processes.	renice(8)
wait: wait for background	processes to complete.	csh(1)
awk: pattern scanning and	processing language.	awk(1)
halt: stop the	processor.	halt(8)
m4: macro	processor.	m4(1)
reboot: reboot system or halt	processor.	reboot(2)
	prof: display profile data.	prof(1)
	profil: execution time profile.	profil(2)
monitor, monstartup, moncontrol: prepare execution	profile.	monitor(3)
profil: execution time	profile.	profil(2)
kgmon: generate a dump of the operating system's	profile buffers.	kgmon(8)
gprof: display call graph	profile data.	gprof(1)
prof: display	profile data.	prof(1)
pxp: Pascal execution	profiler.	pxp(1)
copy: standalone copy	program.	copy(8)
end, etext, edata: last locations in	program.	end(3)
finger: user information lookup	program.	finger(1)
ftp: file transfer	program.	ftp(1C)
liszt: compile a Franz Lisp	program.	liszt(1)
lpc: line printer control	program.	lpc(8)
lpq: spool queue examination	program.	lpq(1)
lxref: lisp cross reference	program.	lxref(1)
msgs: system messages and junk mail	program.	msgs(1)
mt: magnetic tape manipulating	program.	mt(1)
pxref: Pascal cross-reference	program.	pxref(1)
rdist: remote file distribution	program.	rdist(1)
tftp: trivial file transfer	program.	tftp(1C)
units: conversion	program.	units(1)
whereis: locate source, binary, and or manual for	program.	whereis(1)
cb: C	program beautifier.	cb(1)
only). which: locate a	program file including aliases and paths (*csh*	which(1)
make: maintain	program groups.	make(1)
nice: set	program priority.	nice(3C)
getpriority, setpriority: get/set	program scheduling priority.	getpriority(2)
indent: indent and format C	program source.	indent(1)
MPS/DOS vdisks. dosdisk:	program to create and display information for	dosdisk(8)
binstl:	program to install bootloader on disk.	binstl(8)
bload:	program to load standalone programs.	bload(8)
mkfs:	program to make UNIX file systems.	mkfs(8)
park:	program to park the hard disk heads.	park(8)
assert:	program verification.	assert(3X)
lint: a C	program verifier.	lint(1)
fp: Functional	Programming language compiler/interpreter.	fp(1)
bload: program to load standalone	programs.	bload(8)
lex: generator of lexical analysis	programs.	lex(1)
struct: structure Fortran	programs.	struct(1)
vgrind: grind nice listings of	programs.	vgrind(1)
xstr: extract strings from C	programs to implement shared strings.	xstr(1)

telnet: user interface to the TELNET	protocol.	telnet(1C)
getprotobyname, setprotoent, endprotoent: get	protocol entry. getprotoent, getprotobynumber,	getprotoent(3n)
rmt: remote magtape	protocol module.	rmt(8C)
protocols:	protocol name data base.	protocols(5)
ftpd: DARPA Internet File Transfer	Protocol server.	ftpd(8C)
telnetd: DARPA TELNET	protocol server.	telnetd(8C)
tftpd: DARPA Trivial File Transfer	Protocol server.	tftpd(8C)
trpt: transliterate	protocol trace.	trpt(8C)
	protocols: protocol name data base.	protocols(5)
mkproto: construct a	prototype file system.	mkproto(8)
arithmetic:	provide drill in number facts.	arithmetic(6)
false, true:	provide truth values.	false(1)
true, false:	provide truth values.	true(1)
	ps: process status.	ps(1)
	psignal, sys_siglist: system signal messages.	psignal(3)
	pstat: print system facts.	pstat(8)
doctor: interact with a	psychoanalyst.	doctor(6)
	pti: phototypesetter interpreter.	pti(1)
	ptrace: process trace.	ptrace(2)
	ptx: permuted index.	ptx(1)
ungetc:	push character back into input stream.	ungetc(3S)
pushd:	push shell directory stack.	csh(1)
	pushd: push shell directory stack.	csh(1)
puts, fputs:	put a string on a stream.	puts(3S)
putc, putchar, fputc, putw:	put character or word on a stream.	putc(3S)
unit.	putc, fputc: write a character to a fortran logical	putc(3F)
on a stream.	putc, putchar, fputc, putw: put character or word	putc(3S)
stream. putc,	putchar, fputc, putw: put character or word on a	putc(3S)
	puts, fputs: put a string on a stream.	puts(3S)
putc, putchar, fputc,	putw: put character or word on a stream.	putc(3S)
	pwd: working directory name.	pwd(1)
	px: Pascal interpreter.	px(1)
	pxp: Pascal execution profiler.	pxp(1)
	pxref: Pascal cross-reference program.	pxref(1)
	qsort: quick sort.	qsort(3F)
	qsort: quicker sort.	qsort(3)
insque, remque: insert/remove element from a	queue.	insque(3)
lprm: remove jobs from the line printer spooling	queue.	lprm(1)
msgget: get message	queue.	msgget(2)
lpq: spool	queue examination program.	lpq(1)
ipcrm: remove a message	queue, semaphore set or shared memory id.	ipcrm(1)
qsort:	quick sort.	qsort(3F)
qsort:	quicker sort.	qsort(3)
	quiz: test your knowledge.	quiz(6)
	quot: summarize file system ownership.	quot(8)
quotacheck: file system	quota consistency checker.	quotacheck(8)
	quota: display disc usage and limits.	quota(1)
	quota: manipulate disk quotas.	quota(2)
	quotacheck: file system quota consistency checker.	quotacheck(8)
quotaon,	quotaoff: turn file system quotas on and off.	quotaon(8)
off.	quotaon, quotaoff: turn file system quotas on and	quotaon(8)
edquota: edit user	quotas.	edquota(8)
quota: manipulate disk	quotas.	quota(2)
repquota: summarize	quotas for a file system.	repquota(8)
setquota: enable/disable	quotas on a file system.	setquota(2)
quotaon, quotaoff: turn file system	quotas on and off.	quotaon(8)
	rain: animated raindrops display.	rain(6)
rain: animated	raindrops display.	rain(6)
	rand, drand, irand: return random values.	rand(3F)
	rand, srand: random number generator.	rand(3C)
fortune: print a	random, hopefully interesting, adage.	fortune(6)
ranlib: convert archives to	random libraries.	ranlib(1)
rand, srand:	random number generator.	rand(3C)
random, srandom, initstate, setstate: better	random number generator; routines for changing/	random(3)
number generator; routines for changing/	random, srandom, initstate, setstate: better random	random(3)
rand, drand, irand: return	random values.	rand(3F)
	ranlib: convert archives to random libraries.	ranlib(1)
	ratfor: rational Fortran dialect.	ratfor(1)
ratfor:	rational Fortran dialect.	ratfor(1)
	rc: command script for auto-reboot and daemons.	rc(8)
stream to a remote command.	rcmd, rresvport, ruserok: routines for returning a	rcmd(3X)
	rcp: remote file copy.	rcp(1C)
	rcs: change RCS file attributes.	rcs(1)
rcsintro: introduction to	RCS commands.	rcsintro(1)
rcsfile: format of	RCS file.	rcsfile(5)

ci: check in RCS	revisions.	ci(1)
co: check out RCS	revisions.	co(1)
rcsdiff: compare RCS	revisions.	rcsdiff(1)
rcsmerge: merge RCS	revisions.	rcsmerge(1)
fseek, ftell,	rewind: reposition a stream.	fseek(3S)
opendir, readdir, telldir, seekdir,	rewinddir, closedir: directory operations.	directory(3)
directory: opendir, readdir, telldir, seekdir,	rewinddir, closedir: directory operations.	directory(3X)
	rexec: return stream to a remote command.	rexec(3X)
	rexecd: remote execution server.	rexecd(8C)
index,	rindex, lnblnk, len: tell about character objects.	index(3F)
strcmp, strncmp, strcpy, strncpy, strlen, index,	rindex: string operations. strcat, strncat,	string(3)
about RCS files.	rlog: print log messages and other information	rlog(1)
	rlogin: remote login.	rlogin(1C)
	rlogind: remote login server.	rlogind(8C)
rmdir,	rm: remove (unlink) directories or files.	rmdir(1)
	rm, rmdir: remove (unlink) files or directories.	rm(1)
	rmail: handle remote mail received via uucp.	rmail(1)
	rmdir: remove a directory file.	rmdir(2)
rm,	rmdir: remove (unlink) files or directories.	rm(1)
	rmdir, rm: remove (unlink) directories or files.	rmdir(1)
	rmt: remote magtape protocol module.	rmt(8C)
robots: fight off villainous	robots.	robots(6)
	robots: fight off villainous robots.	robots(6)
	roffbib: run off bibliographic database.	roffbib(1)
	rogue: Exploring The Dungeons of Doom.	rogue(6)
pow, sqrt: exponential, logarithm, power, square	root. exp, log, log10,	exp(3M)
chroot: change	root directory.	chroot(2)
	route: manually manipulate the routing tables.	route(8C)
	routed: network routing daemon.	routed(8C)
inet_netof: Internet address manipulation	routines. /inet_ntoa, inet_makeaddr, inet_lnaof,	inet(3n)
tgoto, tputs: terminal independent operation	routines. tgetent, tgetnum, tgetflag, tgetstr,	termcap(3X)
setstate: better random number generator;	routines for changing generators. /initstate,	random(3)
command. rcmd, rresvport, ruserok:	routines for returning a stream to a remote	rcmd(3X)
routed: network	routing daemon.	routed(8C)
route: manually manipulate the	routing tables.	route(8C)
network.	rrestore: restore a file system dump across the	rrestore(8C)
to a remote command. rcmd,	rresvport, ruserok: routines for returning a stream	rcmd(3X)
	rsh: remote shell.	rsh(1C)
	rshd: remote shell server.	rshd(8C)
bit: and, or, xor, not,	rshift, lshift bitwise functions.	bit(3F)
dstrules: Daylight savings time and time zone name	rule file.. .PP	dstrules(5)
nice, nohup:	run a command at low priority (*sh* only).	nice(1)
nohup:	run command immune to hangups.	csh(1)
nice:	run low priority process.	csh(1)
roffbib:	run off bibliographic database.	roffbib(1)
gcore: get core images of	running processes.	gcore(1)
renice: alter priority of	running processes.	renice(8)
	ruptime: show host status of local machines.	ruptime(1C)
remote command. rcmd, rresvport,	ruserok: routines for returning a stream to a	rcmd(3X)
	rwho: who's logged in on local machines.	rwho(1C)
	rwhod: system status server.	rwhod(8C)
	sa, accton: system accounting.	sa(8)
Standalone mode: definition of this	Sanyo/ICON machine operation mode..	standalone(8)
.PP dstrules: Daylight	savings time and time zone name rule file..	dstrules(5)
brk,	sbrk: change data segment size.	brk(2)
scandir:	scan a directory.	scandir(3)
	scandir: scan a directory.	scandir(3)
	scanf, fscanf, sscanf: formatted input conversion.	scanf(3S)
awk: pattern	scanning and processing language.	awk(1)
sccstorcs: build RCS file from	SCCS file.	sccstorcs(1)
sccstorcs: build RCS file from	SCCS file.	sccstorcs(8)
	sccstorcs: build RCS file from SCCS file.	sccstorcs(1)
	sccstorcs: build RCS file from SCCS file.	sccstorcs(8)
alarm:	schedule signal after specified time.	alarm(3C)
getpriority, setpriority: get/set program	scheduling priority.	getpriority(2)
clear: clear terminal	screen.	clear(1)
curses:	screen functions with "optimal" cursor motion.	curses(3X)
ex. vi:	screen oriented (visual) display editor based on	vi(1)
rc: command	script for auto-reboot and daemons.	rc(8)
	script: make typescript of terminal session.	script(1)
onintr: process interrupts in command	scripts.	csh(1)
grep, egrep, fgrep:	search a file for a pattern.	grep(1)
xsend, xget, enroll:	secret mail.	xsend(1)
	sed: stream editor.	sed(1)
opendir, readdir, telldir,	seekdir, rewinddir, closedir: directory operations.	directory(3)

directory: opendir, readdir, telldir,	seekdir, rewinddir, closedir: directory operations.	directory(3X)
shmget: get shared memory	segment.	shmget(2)
brk, sbrk: change data	segment size.	brk(2)
comm:	select or reject lines common to two sorted files.	comm(1)
	select: synchronous i/o multiplexing.	select(2)
case:	selector in switch.	csh(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)
uusend:	send a file to a remote host.	uusend(1C)
send, sendto, sendmsg:	send a message from a socket.	send(2)
kill:	send a signal to a process.	kill(3F)
mail:	send and receive mail.	mail(1)
ping:	send ICMP ECHO_REQUEST packets to network hosts.	ping(8)
sendmail:	send mail over the internet.	sendmail(8)
binmail:	send or receive mail among users.	binmail(1)
socket.	send, sendto, sendmsg: send a message from a	send(2)
kill:	send signal to a process.	kill(2)
killpg:	send signal to a process group.	killpg(2)
	sendbug: mail a system bug report to 4bsd-bugs.	sendbug(1)
aliases: aliases file for	sendmail.	aliases(5)
	sendmail: send mail over the internet.	sendmail(8)
send, sendto,	sendmsg: send a message from a socket.	send(2)
send,	sendto, sendmsg: send a message from a socket.	send(2)
reset: reset the teletype bits to a	sensible state.	reset(1)
diction,explain: print wordy	sentences; thesaurus for diction.	diction(1)
explain, diction– print wordy	sentences; thesaurus for diction.	explain(1)
slattach: attach	serial lines as network interfaces.	slattach(8C)
comsat: biff	server.	comsat(8C)
ftpd: DARPA Internet File Transfer Protocol	server.	ftpd(8C)
rexecd: remote execution	server.	rexecd(8C)
rlogind: remote login	server.	rlogind(8C)
rshd: remote shell	server.	rshd(8C)
rwhod: system status	server.	rwhod(8C)
talkd: remote user communication	server.	talkd(8C)
telnetd: DARPA TELNET protocol	server.	telnetd(8C)
tftpd: DARPA Trivial File Transfer Protocol	server.	tftpd(8C)
	services: service name data base.	services(5)
logout: end	session.	csh(1)
script: make typescript of terminal	session.	script(1)
ascii: map of ASCII character	set.	ascii(7)
stty, gtty:	set and get terminal state (defunct).	stty(3C)
sigstack:	set and/or get signal stack context.	sigstack(2)
	set: change value of shell variable.	csh(1)
sigsetmask:	set current signal mask.	sigsetmask(2)
umask:	set file creation mode mask.	umask(2)
utime:	set file times.	utime(3C)
utimes:	set file times.	utimes(2)
setgroups:	set group access list.	setgroups(2)
apply: apply a command to a	set of arguments.	apply(1)
semget: get	set of semaphores.	semget(2)
getsockopt, setsockopt: get and	set options on sockets.	getsockopt(2)
hostid:	set or print identifier of current host system.	hostid(1)
hostname:	set or print name of current host system.	hostname(1)
ipcrm: remove a message queue, semaphore	set or shared memory id.	ipcrm(1)
setpgrp:	set process group.	setpgrp(2)
nice:	set program priority.	nice(3C)
setregid:	set real and effective group ID.	setregid(2)
setreuid:	set real and effective user ID's.	setreuid(2)
eval, exec, exit, export, login, read, readonly,	set, shift, times, trap, umask, wait: command/ /cd,	sh(1)
getty:	set terminal mode.	getty(8)
stty:	set terminal options.	stty(1)
tabs:	set terminal tabs.	tabs(1)
date: print and	set the date.	date(1)
sfdate:	set the time/date of a file.	sfdate(1)
setuid, seteuid, setruid, setgid, setegid, setrgid:	set user and group ID.	setuid(3)
setenv:	set variable in environment.	csh(1)
a stream.	setbuf, setbuffer, setlinebuf: assign buffering to	setbuf(3S)
stream. setbuf,	setbuffer, setlinebuf: assign buffering to a	setbuf(3S)
setuid, seteuid, setruid, setgid,	setegid, setrgid: set user and group ID.	setuid(3)
	setenv: set variable in environment.	csh(1)

user and group ID. setuid,	seteuid, setruid, setgid, setegid, setrgid: set	setuid(3)
entry. getfsent, getfsspec, getfsfile, getfstype,	setfsent, endfsent: get file system descriptor file	getfsent(3X)
setuid, seteuid, setruid,	setgid, setegid, setrgid: set user and group ID.	setuid(3)
getgrent, getgrgid, getgrnam,	setgrent, endgrent: get group file entry.	getgrent(3)
	setgroups: set group access list.	setgroups(2)
gethostent, gethostbyaddr, gethostbyname,	sethostent, endhostent: get network host entry.	gethostent(3n)
host. gethostid,	sethostid: get/set unique identifier of current	gethostid(2)
gethostname,	sethostname: get/set name of current host.	gethostname(2)
getitimer,	setitimer: get/set value of interval timer.	getitimer(2)
	setjmp, longjmp: non-local goto.	setjmp(3)
crypt,	setkey, encrypt: DES encryption.	crypt(3)
setbuf, setbuffer,	setlinebuf: assign buffering to a stream.	setbuf(3S)
getnetent, getnetbyaddr, getnetbyname,	setnetent, endnetent: get network entry.	getnetent(3n)
	setpgrp: set process group.	setpgrp(2)
getpriority,	setpriority: get/set program scheduling priority.	getpriority(2)
getprotoent, getprotobynumber, getprotobyname,	setprotoent, endprotoent: get protocol entry.	getprotoent(3n)
getpwent, getpwuid, getpwnam,	setpwent, endpwent: get password file entry.	getpwent(3)
	setquota: enable/disable quotas on a file system.	setquota(2)
	setregid: set real and effective group ID.	setregid(2)
	setreuid: set real and effective user ID's.	setreuid(2)
setuid, seteuid, setruid, setgid, setegid,	setrgid: set user and group ID.	setuid(3)
consumption. getrlimit,	setrlimit: control maximum system resource	getrlimit(2)
group ID. setuid, seteuid,	setruid, setgid, setegid, setrgid: set user and	setuid(3)
getservent, getservbyport, getservbyname,	setservent, endservent: get service entry.	getservent(3n)
getsockopt,	setsockopt: get and set options on sockets.	getsockopt(2)
for changing/ random, srandom, initstate,	setstate: better random number generator; routines	random(3)
gettimeofday,	settimeofday: get/set date and time.	gettimeofday(2)
set user and group ID.	setuid, seteuid, setruid, setgid, setegid, setrgid:	setuid(3)
	sfdate: set the time/date of a file.	sfdate(1)
continue, cd, eval, exec, exit, export, login,/	sh, for, case, if, while, :, ., break,	sh(1)
shmctl:	shared memory control operations.	shmctl(2)
ipcrm: remove a message queue, semaphore set or	shared memory id.	ipcrm(1)
shmop:	shared memory operations.	shmop(2)
shmget: get	shared memory segment.	shmget(2)
xstr: extract strings from C programs to implement	shared strings.	xstr(1)
chsh: change default login	shell.	chsh(1)
exit: leave	shell.	csh(1)
rsh: remote	shell.	rsh(1C)
system: issue a	shell command.	system(3)
csh: a	shell (command interpreter) with C-like syntax.	csh(1)
eval: re-evaluate	shell data.	csh(1)
popd: pop	shell directory stack.	csh(1)
pushd: push	shell directory stack.	csh(1)
alias:	shell macros.	csh(1)
suspend: suspend a	shell, resuming its superior.	csh(1)
rshd: remote	shell server.	rshd(8C)
set: change value of	shell variable.	csh(1)
@: arithmetic on	shell variables.	csh(1)
unset: discard	shell variables.	csh(1)
exec: overlay	shell with specified command.	csh(1)
	shift: manipulate argument list.	csh(1)
/exec, exit, export, login, read, readonly, set,	shift, times, trap, umask, wait: command language.	sh(1)
	shmctl: shared memory control operations.	shmctl(2)
	shmget: get shared memory segment.	shmget(2)
	shmop: shared memory operations.	shmop(2)
long,	short: integer object conversion.	long(3F)
groups:	show group memberships.	groups(1)
ruptime:	show host status of local machines.	ruptime(1C)
uptime:	show how long system has been up.	uptime(1)
lastcomm:	show last commands executed in reverse order.	lastcomm(1)
netstat:	show network status.	netstat(1)
uusnap:	show snapshot of the UUCP system.	uusnap(8C)
construct a file. what:	show what versions of object modules were used to	what(1)
shutdown:	shut down part of a full-duplex connection.	shutdown(2)
	shutdown: close down the system at a given time.	shutdown(8)
connection.	shutdown: shut down part of a full-duplex	shutdown(2)
	sigblock: block signals.	sigblock(2)
login:	sign on.	login(1)
pause: stop until	signal.	pause(3C)
signal: change the action for a	signal.	signal(3F)
alarm: schedule	signal after specified time.	alarm(3C)
	signal: change the action for a signal.	signal(3F)
signal: simplified software	signal facilities.	signal(3C)
sigvec: software	signal facilities.	sigvec(2)
sigsetmask: set current	signal mask.	sigsetmask(2)

psignal, sys_siglist: system	signal messages.	psignal(3)
	signal: simplified software signal facilities.	signal(3C)
sigstack: set and/or get	signal stack context.	sigstack(2)
kill: send	signal to a process.	kill(2)
kill: send a	signal to a process.	kill(3F)
killpg: send	signal to a process group.	killpg(2)
sigblock: block	signals.	sigblock(2)
sigpause: atomically release blocked	signals and wait for interrupt.	sigpause(2)
wait for interrupt.	sigpause: atomically release blocked signals and	sigpause(2)
	sigsetmask: set current signal mask.	sigsetmask(2)
	sigstack: set and/or get signal stack context.	sigstack(2)
	sigvec: software signal facilities.	sigvec(2)
signal:	simplified software signal facilities.	signal(3C)
tc: photoypesetter	simulator.	tc(1)
trigonometric functions.	sin, cos, tan, asin, acos, atan, atan2:	sin(3M)
	sinh, cosh, tanh: hyperbolic functions.	sinh(3M)
null: data	sink.	null(4)
brk, sbrk: change data segment	size.	brk(2)
getdtablesize: get descriptor table	size.	getdtablesize(2)
getpagesize: get system page	size.	getpagesize(2)
pagesize: print system page	size.	pagesize(1)
size:	size of an object file.	size(1)
	size: size of an object file.	size(1)
interfaces.	slattach: attach serial lines as network	slattach(8C)
	sleep: suspend execution for an interval.	sleep(1)
	sleep: suspend execution for an interval.	sleep(3F)
	sleep: suspend execution for interval.	sleep(3)
dosprinters: destinations for spooled output from	SLPT printers.	dosprinters(5)
spline: interpolate	smooth curve.	spline(1G)
	snake, snscore: display chase game.	snake(6)
uusnap: show	snapshot of the UUCP system.	uusnap(8C)
snake,	snscore: display chase game.	snake(6)
accept: accept a connection on a	socket.	accept(2)
bind: bind a name to a	socket.	bind(2)
connect: initiate a connection on a	socket.	connect(2)
listen: listen for connections on a	socket.	listen(2)
recv, recvfrom, recvmsg: receive a message from a	socket.	recv(2)
send, sendto, sendmsg: send a message from a	socket.	send(2)
	socket: create an endpoint for communication.	socket(2)
getsockname: get	socket name.	getsockname(2)
	socketpair: create a pair of connected sockets.	socketpair(2)
getsockopt, setsockopt: get and set options on	sockets.	getsockopt(2)
socketpair: create a pair of connected	sockets.	socketpair(2)
	soelim: eliminate .so's from nroff input.	soelim(1)
signal: simplified	software signal facilities.	signal(3C)
sigvec:	software signal facilities.	sigvec(2)
canfield, cfscores: the	solitaire card game canfield.	canfield(6)
qsort: quicker	sort.	qsort(3)
qsort: quick	sort.	qsort(3F)
tsort: topological	sort.	tsort(1)
sortbib:	sort bibliographic database.	sortbib(1)
sort:	sort or merge files.	sort(1)
	sort: sort or merge files.	sort(1)
	sortbib: sort bibliographic database.	sortbib(1)
comm: select or reject lines common to two	sorted files.	comm(1)
look: find lines in a	sorted list.	look(1)
soelim: eliminate	.so's from nroff input.	soelim(1)
soelim: eliminate .	so's from nroff input.	soelim(1)
indent: indent and format C program	source.	indent(1)
mkstr: create an error message file by massaging C	source.	mkstr(1)
whereis: locate	source, binary, and or manual for program.	whereis(1)
	source: read commands from file.	csh(1)
line, circle, arc, move, cont, point, linemod,	space, closepl: graphics interface. /erase, label,	plot(3X)
expand, unexpand: expand tabs to	spaces, and vice versa.	expand(1)
way. vfork:	spawn new process in a virtual memory efficient	vfork(2)
exec: overlay shell with	specified command.	csh(1)
truncate: truncate a file to a	specified length.	truncate(2)
alarm: schedule signal after	specified time.	alarm(3C)
alarm: execute a subroutine after a	specified time.	alarm(3F)
swapon:	specify a swap directory.	swapon(2)
swapon:	specify additional device for paging and swapping.	swapon(8)
	spell, spellin, spellout: find spelling errors.	spell(1)
spell,	spellin, spellout: find spelling errors.	spell(1)
spell, spellin, spellout: find	spelling errors.	spell(1)
spell, spellin,	spellout: find spelling errors.	spell(1)

	spline: interpolate smooth curve.	spline(1G)
split:	split a file into pieces.	split(1)
files. fsplit:	split a multi-routine Fortran file into individual	fsplit(1)
frexp, ldexp, modf:	split into mantissa and exponent.	frexp(3)
	split: split a file into pieces.	split(1)
uuclean: uucp	spool directory clean-up.	uuclean(8C)
lpq:	spool queue examination program.	lpq(1)
dosprinters: destinations for	spooled output from SLPT printers.	dosprinters(5)
dosprint: MPS/DOS	spooler daemon.	dosprint(8)
lprm: remove jobs from the line printer	spooling queue.	lprm(1)
printf, fprintf,	sprintf: formatted output conversion.	printf(3S)
exp, log, log10, pow,	sqrt: exponential, logarithm, power, square root.	exp(3M)
log10, pow, sqrt: exponential, logarithm, power,	square root. exp, log,	exp(3M)
rand,	srand: random number generator.	rand(3C)
generator; routines for changing/ random,	srandom, initstate, setstate: better random number	random(3)
scanf, fscanf,	sscanf: formatted input conversion.	scanf(3S)
	stab: symbol table types.	stab(5)
popd: pop shell directory	stack.	csh(1)
pushd: push shell directory	stack.	csh(1)
sigstack: set and/or get signal	stack context.	sigstack(2)
copy:	standalone copy program.	copy(8)
dkfmt:	standalone disk formatter.	dkfmt(8)
machine operation mode..	Standalone mode: definition of this Sanyo/ICON	standalone(8)
bload: program to load	standalone programs.	bload(8)
stdio:	standard buffered input/output package.	intro(3S)
htable: convert NIC	standard format host tables.	htable(8)
	stat, lstat, fstat: get file status.	stat(2)
	stat, lstat, fstat: get file status.	stat(3F)
reset: reset the teletype bits to a sensible	state.	reset(1)
stty, gtty: set and get terminal	state (defunct).	stty(3C)
fsync: synchronize a file's in-core	state with that on disk.	fsync(2)
if: conditional	statement.	csh(1)
fstab:	static information about the filesystems.	fstab(5)
hashstat: print command hashing	statistics.	csh(1)
iostat: report I/O	statistics.	iostat(1)
vmstat: report virtual memory	statistics.	vmstat(1)
exit: terminate process with	status.	exit(3F)
ipcs: report inter-process communication facilities	status.	ipcs(1)
netstat: show network	status.	netstat(1)
ps: process	status.	ps(1)
stat, lstat, fstat: get file	status.	stat(2)
stat, lstat, fstat: get file	status.	stat(3F)
ferror, feof, clearerr, fileno: stream	status inquiries.	ferror(3S)
sysline: display system status on	status line of a terminal.	sysline(1)
ruptime: show host	status of local machines.	ruptime(1C)
sysline: display system	status on status line of a terminal.	sysline(1)
rwhod: system	status server.	rwhod(8C)
	stdio: standard buffered input/output package.	intro(3S)
	sticky: executable files with persistent text.	sticky(8)
	stop: halt a job or process.	csh(1)
halt:	stop the processor.	halt(8)
pause:	stop until signal.	pause(3C)
icheck: file system	storage consistency check.	icheck(8)
subroutines. dbminit, fetch,	store, delete, firstkey, nextkey: data base	dbm(3X)
strlen, index, rindex: string operations.	strcat, strncat, strcmp, strncmp, strcpy, strncpy,	string(3)
rindex: string operations. strcat, strncat,	strcmp, strncmp, strcpy, strncpy, strlen, index,	string(3)
operations. strcat, strncat, strcmp, strncmp,	strcpy, strncpy, strlen, index, rindex: string	string(3)
fclose, fflush: close or flush a	stream.	fclose(3S)
fopen, freopen, fdopen: open a	stream.	fopen(3S)
fseek, ftell, rewind: reposition a	stream.	fseek(3S)
getchar, fgetc, getw: get character or word from	stream. getc,	getc(3S)
gets, fgets: get a string from a	stream.	gets(3S)
putchar, fputc, putw: put character or word on a	stream. putc,	putc(3S)
puts, fputs: put a string on a	stream.	puts(3S)
setbuffer, setlinebuf: assign buffering to a	stream. setbuf,	setbuf(3S)
ungetc: push character back into input	stream.	ungetc(3S)
sed:	stream editor.	sed(1)
ferror, feof, clearerr, fileno:	stream status inquiries.	ferror(3S)
rcmd, rresvport, ruserok: routines for returning a	stream to a remote command.	rcmd(3X)
rexec: return	stream to a remote command.	rexec(3X)
fdate: return date and time in an ASCII	string.	fdate(3F)
gets, fgets: get a	string from a stream.	gets(3S)
puts, fputs: put a	string on a stream.	puts(3S)
bcopy, bcmp, bzero, ffs: bit and byte	string operations.	bstring(3)
strncmp, strcpy, strncpy, strlen, index, rindex:	string operations. strcat, strncat, strcmp,	string(3)

extract strings from C programs to implement shared	**strings. xstr:**	**xstr(1)**
other binary, file.	**strings: find the printable strings in a object, or**	**strings(1)**
strings. xstr: extract	**strings from C programs to implement shared**	**xstr(1)**
strings: find the printable	**strings in a object, or other binary, file.**	**strings(1)**
basename:	**strip filename affixes.**	**basename(1)**
	strip: remove symbols and relocation bits.	**strip(1)**
strcat, strncat, strcmp, strncmp, strcpy, strncpy,	**strlen, index, rindex: string operations.**	**string(3)**
index, rindex: string operations. strcat,	**strncat, strcmp, strncmp, strcpy, strncpy, strlen,**	**string(3)**
string operations. strcat, strncat, strcmp,	**strncmp, strcpy, strncpy, strlen, index, rindex:**	**string(3)**
strcat, strncat, strcmp, strncmp, strcpy,	**strncpy, strlen, index, rindex: string operations.**	**string(3)**
	struct: structure Fortran programs.	**struct(1)**
struct:	**structure Fortran programs.**	**struct(1)**
	stty, gtty: set and get terminal state (defunct).	**stty(3C)**
	stty: set terminal options.	**stty(1)**
document.	**style: analyze surface characteristics of a**	**style(1)**
	su: substitute user id temporarily.	**su(1)**
alarm: execute a	**subroutine after a specified time.**	**alarm(3F)**
fetch, store, delete, firstkey, nextkey: data base	**subroutines. dbminit,**	**dbm(3X)**
lib2648:	subroutines for the HP 2648 graphics terminal.	lib2648(3X)
su:	substitute user id temporarily.	su(1)
sum:	sum and count blocks in a file.	sum(1)
	sum: sum and count blocks in a file.	sum(1)
du:	summarize disk usage.	du(1)
quot:	summarize file system ownership.	quot(8)
repquota:	summarize quotas for a file system.	repquota(8)
sync: update the	super block.	sync(8)
update: periodically update the	super block.	update(8)
sync: update	super-block.	sync(2)
suspend: suspend a shell, resuming its	superior.	csh(1)
style: analyze	surface characteristics of a document.	style(1)
suspend:	suspend a shell, resuming its superior.	csh(1)
sleep:	suspend execution for an interval.	sleep(1)
sleep:	suspend execution for an interval.	sleep(3F)
sleep:	suspend execution for interval.	sleep(3)
	suspend: suspend a shell, resuming its superior.	csh(1)
	swab: swap bytes.	swab(3)
swab:	swap bytes.	swab(3)
swapon: specify a	swap directory.	swapon(2)
	swapon: specify a swap directory.	swapon(2)
swapping.	swapon: specify additional device for paging and	swapon(8)
swapon: specify additional device for paging and	swapping.	swapon(8)
breaksw: exit from	switch.	csh(1)
case: selector in	switch.	csh(1)
default: catchall clause in	switch.	csh(1)
endsw: terminate	switch.	csh(1)
	switch: multi-way command branch.	csh(1)
stab:	symbol table types.	stab(5)
readlink: read value of a	symbolic link.	readlink(2)
symlink: make	symbolic link to a file.	symlink(2)
strip: remove	symbols and relocation bits.	strip(1)
	symlink: make symbolic link to a file.	symlink(2)
	symorder: rearrange name list.	symorder(1)
	sync: update super-block.	sync(2)
	sync: update the super block.	sync(8)
disk. fsync:	synchronize a file's in-core state with that on	fsync(2)
select:	synchronous i/o multiplexing.	select(2)
csh: a shell (command interpreter) with C-like	syntax.	csh(1)
	syscall: indirect system call.	syscall(2)
perror,	sys_errlist, sys_nerr: system error messages.	perror(3)
terminal.	sysline: display system status on status line of a	sysline(1)
	syslog: log systems messages.	syslog(8)
	syslog, openlog, closelog: control system log.	syslog(3)
perror, sys_errlist,	sys_nerr: system error messages.	perror(3)
psignal,	sys_siglist: system signal messages.	psignal(3)
jove_recover - recover JOVE buffers after a	system/editor crash.	jove_recover(1)
mkfs: program to make UNIX file	systems.	mkfs(8)
syslog: log	systems messages.	syslog(8)
kgmon: generate a dump of the operating	system's profile buffers.	kgmon(8)
rehash: recompute command hash	table.	csh(1)
unhash: discard command hash	table.	csh(1)
mtab: mounted file system	table.	mtab(5)
getdtablesize: get descriptor	table size.	getdtablesize(2)
stab: symbol	table types.	stab(5)
htable: convert NIC standard format host	tables.	htable(8)
route: manually manipulate the routing	tables.	route(8C)

@: arithmetic on shell	**variables.**	csh(1)
unset: discard shell	**variables.**	csh(1)
unsetenv: remove environment	**variables.**	csh(1)
getenv: get value of environment	**variables.**	getenv(3F)
to create and display information for MPS/DOS	**vdisks. dosdisk: program**	dosdisk(8)
assert: program	**verification.**	assert(3X)
lint: a C program	**verifier.**	lint(1)
	vers: print version number of the kernel.	vers(1)
expand, unexpand: expand tabs to spaces, and vice	**versa.**	expand(1)
trman: translate	**version 6 manual macros to version 7 macros.**	trman(1)
trman: translate version 6 manual macros to	**version 7 macros.**	trman(1)
vers: print	version number of the kernel.	vers(1)
hangman: Computer	version of the game hangman.	hangman(6)
file. what: show what	versions of object modules were used to construct a	what(1)
efficient way.	vfork: spawn new process in a virtual memory	vfork(2)
	vgrind: grind nice listings of programs.	vgrind(1)
	vgrindefs: vgrind's language definition data base.	vgrindefs(5)
vgrindefs:	vgrind's language definition data base.	vgrindefs(5)
terminal.	vhangup: virtually "hangup" the current control	vhangup(2)
on ex.	vi: screen oriented (visual) display editor based	vi(1)
encode/decode a binary file for transmission	via mail. uuencode,uudecode:	uuencode(1C)
rmail: handle remote mail received	via uucp.	rmail(1)
expand, unexpand: expand tabs to spaces, and	vice versa.	expand(1)
more, page: file perusal filter for crt	viewing.	more(1)
robots: fight off	villainous robots.	robots(6)
	vipw: edit the password file.	vipw(8)
dosdisks: list of MPS/DOS	virtual disks.	dosdisks(5)
vfork: spawn new process in a	virtual memory efficient way.	vfork(2)
vmstat: report	virtual memory statistics.	vmstat(1)
vhangup:	virtually "hangup" the current control terminal.	vhangup(2)
vi: screen oriented	(visual) display editor based on ex.	vi(1)
consumption.	vlimit: control maximum system resource	vlimit(3C)
	vmstat: report virtual memory statistics.	vmstat(1)
fs, inode: format of file system	volume.	fs(5)
	vtimes: get information about resource utilization.	vtimes(3C)
	w: who is on and what they are doing.	w(1)
	wait: await completion of process.	wait(1)
read, readonly, set, shift, times, trap, umask,	wait: command language. /exec, exit, export, login,	sh(1)
wait:	wait for a process to terminate.	wait(3F)
wait:	wait for background processes to complete.	csh(1)
sigpause: atomically release blocked signals and	wait for interrupt.	sigpause(2)
wait, wait3:	wait for process to terminate.	wait(2)
	wait: wait for a process to terminate.	wait(3F)
	wait: wait for background processes to complete.	csh(1)
	wait, wait3: wait for process to terminate.	wait(2)
wait,	wait3: wait for process to terminate.	wait(2)
	wall: write to all users.	wall(1)
	wc: word count.	wc(1)
what: show what versions of object modules	were used to construct a file.	what(1)
whatis: describe	what a command is.	whatis(1)
crash:	what happens when the system crashes.	crash(8V)
used to construct a file.	what: show what versions of object modules were	what(1)
w: who is on and	what they are doing.	w(1)
construct a file. what: show	what versions of object modules were used to	what(1)
	whatis: describe what a command is.	whatis(1)
crash: what happens	when the system crashes.	crash(8V)
leave: remind you	when you have to leave.	leave(1)
program.	whereis: locate source, binary, and or manual for	whereis(1)
paths (*csh* only).	which: locate a program file including aliases and	which(1)
exec, exit, export, login,/ sh, for, case, if,	while, : , . , break, continue, cd, eval,	sh(1)
	while: repeat commands conditionally.	csh(1)
break: exit	while/foreach loop.	csh(1)
users: compact list of users	who are on the system.	users(1)
from:	who is my mail from?.	from(1)
w:	who is on and what they are doing.	w(1)
who:	who is on the system.	who(1)
biff: be notified if mail arrives and	who it is from.	biff(1)
	who: who is on the system.	who(1)
	whoami: print effective current user id.	whoami(1)
	whodos: display information about dosc users.	whodos(1)
rwho:	who's logged in on local machines.	rwho(1C)
fold: fold long lines for finite	width output device.	fold(1)
window:	window environment.	window(1)
	window: window environment.	window(1)
fastboot, fasthalt: reboot/halt the system	without checking the disks.	fastboot(8)

ICON/UXB OPERATING SYSTEM C LIBRARY SUBROUTINES

NAME

intro – introduction to library functions

DESCRIPTION

This section describes functions that may be found in various libraries. The library functions are those other than the functions which directly invoke UNIX system primitives, described in section 2. This section has the libraries physically grouped together. This is a departure from older versions of the UNIX Programmer's Reference Manual, which did not group functions by library. The functions described in this section are grouped into various libraries:

(3) and (3S)
: The straight "3" functions are the standard C library functions. The C library also includes all the functions described in section 2. The 3S functions comprise the standard I/O library. Together with the (3N), (3X), and (3C) routines, these functions constitute library *libc*, which is automatically loaded by the C compiler *cc*(1), the Pascal compiler *pc*(1), and the Fortran compiler *f77*(1). The link editor *ld*(1) searches this library under the '–lc' option. Declarations for some of these functions may be obtained from include files indicated on the appropriate pages.

(3F) The 3F functions are all functions callable from FORTRAN. These functions perform the same jobs as do the straight "3" functions.

(3M) These functions constitute the math library, *libm*. They are automatically loaded as needed by the Pascal compiler *pc*(1) and the Fortran compiler *f77*(1). The link editor searches this library under the '–lm' option. Declarations for these functions may be obtained from the include file <*math.h*>.

(3N) These functions constitute the internet network library,

(3S) These functions constitute the 'standard I/O package', see *intro*(3S). These functions are in the library *libc* already mentioned. Declarations for these functions may be obtained from the include file <*stdio.h*>.

(3X) Various specialized libraries have not been given distinctive captions. Files in which such libraries are found are named on appropriate pages.

(3C) Routines included for compatibility with other systems. In particular, a number of system call interfaces provided in previous releases of 4BSD have been included for source code compatibility. The manual page entry for each compatibility routine indicates the proper interface to use.

FILES

/lib/libc.a
/usr/lib/libm.a
/usr/lib/libc_p.a
/usr/lib/libm_p.a

SEE ALSO

intro(3C), intro(3S), intro(3F), intro(3M), intro(3N), nm(1), ld(1), cc(1), f77(1), intro(2)

DIAGNOSTICS

Functions in the math library (3M) may return conventional values when the function is undefined for the given arguments or when the value is not representable. In these cases the external variable *errno* (see *intro*(2)) is set to the value EDOM (domain error) or ERANGE (range error). The values of EDOM and ERANGE are defined in the include file <*math.h*>.

LIST OF FUNCTIONS

Name	*Appears on Page*	*Description*
abort	abort.3	generate a fault
abort	abort.3f	terminate abruptly with memory image
abs	abs.3	integer absolute value
access	access.3f	determine accessability of a file
acos	sin.3m	trigonometric functions
alarm	alarm.3c	schedule signal after specified time
alarm	alarm.3f	execute a subroutine after a specified time
alloca	malloc.3	memory allocator
arc	plot.3x	graphics interface
asctime	ctime.3	convert date and time to ASCII
asin	sin.3m	trigonometric functions
assert	assert.3x	program verification
atan	sin.3m	trigonometric functions
atan2	sin.3m	trigonometric functions
atof	atof.3	convert ASCII to numbers
atoi	atof.3	convert ASCII to numbers
atol	atof.3	convert ASCII to numbers
bcmp	bstring.3	bit and byte string operations
bcopy	bstring.3	bit and byte string operations
bessel	bessel.3f	of two kinds for integer orders
bit	bit.3f	and, or, xor, not, rshift, lshift bitwise functions
bzero	bstring.3	bit and byte string operations
cabs	hypot.3m	Euclidean distance
calloc	malloc.3	memory allocator
ceil	floor.3m	absolute value, floor, ceiling functions
chdir	chdir.3f	change default directory
chmod	chmod.3f	change mode of a file
circle	plot.3x	graphics interface
clearerr	ferror.3s	stream status inquiries
closedir	directory.3	directory operations
closelog	syslog.3	control system log
closepl	plot.3x	graphics interface
cont	plot.3x	graphics interface
cos	sin.3m	trigonometric functions
cosh	sinh.3m	hyperbolic functions
crypt	crypt.3	DES encryption
ctime	ctime.3	convert date and time to ASCII
ctime	time.3f	return system time
curses	curses.3x	screen functions with "optimal" cursor motion
dbminit	dbm.3x	data base subroutines
delete	dbm.3x	data base subroutines
dffrac	flmin.3f	return extreme values
dflmax	flmin.3f	return extreme values
dflmax	range.3f	return extreme values
dflmin	flmin.3f	return extreme values
dflmin	range.3f	return extreme values
drand	rand.3f	return random values
dtime	etime.3f	return elapsed execution time

ecvt	ecvt.3	output conversion
edata	end.3	last locations in program
encrypt	crypt.3	DES encryption
end	end.3	last locations in program
endfsent	getfsent.3x	get file system descriptor file entry
endgrent	getgrent.3	get group file entry
endhostent	gethostent.3n	get network host entry
endnetent	getnetent.3n	get network entry
endprotoent	getprotoent.3n	get protocol entry
endpwent	getpwent.3	get password file entry
endservent	getservent.3n	get service entry
environ	execl.3	execute a file
erase	plot.3x	graphics interface
etext	end.3	last locations in program
etime	etime.3f	return elapsed execution time
exec	execl.3	execute a file
exece	execl.3	execute a file
execl	execl.3	execute a file
execle	execl.3	execute a file
execlp	execl.3	execute a file
exect	execl.3	execute a file
execv	execl.3	execute a file
execvp	execl.3	execute a file
exit	exit.3	terminate a process after flushing any pending output
exit	exit.3f	terminate process with status
exp	exp.3m	exponential, logarithm, power, square root
fabs	floor.3m	absolute value, floor, ceiling functions
fclose	fclose.3s	close or flush a stream
fcvt	ecvt.3	output conversion
fdate	fdate.3f	return date and time in an ASCII string
feof	ferror.3s	stream status inquiries
ferror	ferror.3s	stream status inquiries
fetch	dbm.3x	data base subroutines
fflush	fclose.3s	close or flush a stream
ffrac	flmin.3f	return extreme values
ffs	bstring.3	bit and byte string operations
fgetc	getc.3f	get a character from a logical unit
fgetc	getc.3s	get character or word from stream
fgets	gets.3s	get a string from a stream
fileno	ferror.3s	stream status inquiries
firstkey	dbm.3x	data base subroutines
flmax	flmin.3f	return extreme values
flmax	range.3f	return extreme values
flmin	flmin.3f	return extreme values
flmin	range.3f	return extreme values
floor	floor.3m	absolute value, floor, ceiling functions
flush	flush.3f	flush output to a logical unit
fork	fork.3f	create a copy of this process
fpecnt	trpfpe.3f	trap and repair floating point faults
fprintf	printf.3s	formatted output conversion
fputc	putc.3f	write a character to a fortran logical unit

fputc	putc.3s	put character or word on a stream
fputs	puts.3s	put a string on a stream
fread	fread.3s	buffered binary input/output
free	malloc.3	memory allocator
frexp	frexp.3	split into mantissa and exponent
fscanf	scanf.3s	formatted input conversion
fseek	fseek.3f	reposition a file on a logical unit
fseek	fseek.3s	reposition a stream
fstat	stat.3f	get file status
ftell	fseek.3f	reposition a file on a logical unit
ftell	fseek.3s	reposition a stream
ftime	time.3c	get date and time
fwrite	fread.3s	buffered binary input/output
gamma	gamma.3m	log gamma function
gcvt	ecvt.3	output conversion
gerror	perror.3f	get system error messages
getarg	getarg.3f	return command line arguments
getc	getc.3f	get a character from a logical unit
getc	getc.3s	get character or word from stream
getchar	getc.3s	get character or word from stream
getcwd	getcwd.3f	get pathname of current working directory
getdiskbyname	getdisk.3x	get disk description by its name
getenv	getenv.3	value for environment name
getenv	getenv.3f	get value of environment variables
getfsent	getfsent.3x	get file system descriptor file entry
getfsfile	getfsent.3x	get file system descriptor file entry
getfsspec	getfsent.3x	get file system descriptor file entry
getfstype	getfsent.3x	get file system descriptor file entry
getgid	getuid.3f	get user or group ID of the caller
getgrent	getgrent.3	get group file entry
getgrgid	getgrent.3	get group file entry
getgrnam	getgrent.3	get group file entry
gethostbyaddr	gethostent.3n	get network host entry
gethostbyname	gethostent.3n	get network host entry
gethostent	gethostent.3n	get network host entry
getlog	getlog.3f	get user's login name
getlogin	getlogin.3	get login name
getnetbyaddr	getnetent.3n	get network entry
getnetbyname	getnetent.3n	get network entry
getnetent	getnetent.3n	get network entry
getpass	getpass.3	read a password
getpid	getpid.3f	get process id
getprotobyname	getprotoent.3n	get protocol entry
getprotobynumber	getprotoent.3n	get protocol entry
getprotoent	getprotoent.3n	get protocol entry
getpw	getpw.3	get name from uid
getpwent	getpwent.3	get password file entry
getpwnam	getpwent.3	get password file entry
getpwuid	getpwent.3	get password file entry
gets	gets.3s	get a string from a stream
getservbyname	getservent.3n	get service entry

getservbyport	getservent.3n	get service entry
getservent	getservent.3n	get service entry
getuid	getuid.3f	get user or group ID of the caller
getw	getc.3s	get character or word from stream
getwd	getwd.3	get current working directory pathname
gmtime	ctime.3	convert date and time to ASCII
gmtime	time.3f	return system time
gtty	stty.3c	set and get terminal state (defunct)
hostnm	hostnm.3f	get name of current host
htonl	byteorder.3n	convert values between host and network byte order
htons	byteorder.3n	convert values between host and network byte order
hypot	hypot.3m	Euclidean distance
iargc	getarg.3f	return command line arguments
idate	idate.3f	return date or time in numerical form
ierrno	perror.3f	get system error messages
index	index.3f	tell about character objects
index	string.3	string operations
inet_addr	inet.3n	Internet address manipulation routines
inet_lnaof	inet.3n	Internet address manipulation routines
inet_makeaddr	inet.3n	Internet address manipulation routines
inet_netof	inet.3n	Internet address manipulation routines
inet_network	inet.3n	Internet address manipulation routines
initgroups	initgroups.3x	initialize group access list
initstate	random.3	better random number generator
inmax	flmin.3f	return extreme values
inmax	range.3f	return extreme values
insque	insque.3	insert/remove element from a queue
ioinit	ioinit.3f	change f77 I/O initialization
irand	rand.3f	return random values
isalnum	ctype.3	character classification macros
isalpha	ctype.3	character classification macros
isascii	ctype.3	character classification macros
isatty	ttynam.3f	find name of a terminal port
isatty	ttyname.3	find name of a terminal
iscntrl	ctype.3	character classification macros
isdigit	ctype.3	character classification macros
islower	ctype.3	character classification macros
isprint	ctype.3	character classification macros
ispunct	ctype.3	character classification macros
isspace	ctype.3	character classification macros
isupper	ctype.3	character classification macros
itime	idate.3f	return date or time in numerical form
j0	j0.3m	bessel functions
j1	j0.3m	bessel functions
jn	j0.3m	bessel functions
kill	kill.3f	send a signal to a process
label	plot.3x	graphics interface
ldexp	frexp.3	split into mantissa and exponent
len	index.3f	tell about character objects
lib2648	lib2648.3x	subroutines for the HP 2648 graphics terminal
line	plot.3x	graphics interface

linemod	plot.3x	graphics interface
link	link.3f	make a link to an existing file
lnblnk	index.3f	tell about character objects
loc	loc.3f	return the address of an object
localtime	ctime.3	convert date and time to ASCII
log	exp.3m	exponential, logarithm, power, square root
log10	exp.3m	exponential, logarithm, power, square root
long	long.3f	integer object conversion
longjmp	setjmp.3	non-local goto
lstat	stat.3f	get file status
ltime	time.3f	return system time
malloc	malloc.3	memory allocator
mktemp	mktemp.3	make a unique file name
modf	frexp.3	split into mantissa and exponent
moncontrol	monitor.3	prepare execution profile
monitor	monitor.3	prepare execution profile
monstartup	monitor.3	prepare execution profile
move	plot.3x	graphics interface
nextkey	dbm.3x	data base subroutines
nice	nice.3c	set program priority
nlist	nlist.3	get entries from name list
ntohl	byteorder.3n	convert values between host and network byte order
ntohs	byteorder.3n	convert values between host and network byte order
opendir	directory.3	directory operations
openlog	syslog.3	control system log
pause	pause.3c	stop until signal
pclose	popen.3	initiate I/O to/from a process
perror	perror.3	system error messages
perror	perror.3f	get system error messages
plot: openpl	plot.3x	graphics interface
point	plot.3x	graphics interface
popen	popen.3	initiate I/O to/from a process
pow	exp.3m	exponential, logarithm, power, square root
printf	printf.3s	formatted output conversion
psignal	psignal.3	system signal messages
putc	putc.3f	write a character to a fortran logical unit
putc	putc.3s	put character or word on a stream
putchar	putc.3s	put character or word on a stream
puts	puts.3s	put a string on a stream
putw	putc.3s	put character or word on a stream
qsort	qsort.3	quicker sort
qsort	qsort.3f	quick sort
rand	rand.3c	random number generator
rand	rand.3f	return random values
random	random.3	better random number generator
rcmd	rcmd.3x	routines for returning a stream to a remote command
re_comp	regex.3	regular expression handler
re_exec	regex.3	regular expression handler
readdir	directory.3	directory operations
realloc	malloc.3	memory allocator
remque	insque.3	insert/remove element from a queue

rename	rename.3f	rename a file
rewind	fseek.3s	reposition a stream
rewinddir	directory.3	directory operations
rexec	rexec.3x	return stream to a remote command
rindex	index.3f	tell about character objects
rindex	string.3	string operations
rresvport	rcmd.3x	routines for returning a stream to a remote command
ruserok	rcmd.3x	routines for returning a stream to a remote command
scandir	scandir.3	scan a directory
scanf	scanf.3s	formatted input conversion
seekdir	directory.3	directory operations
setbuf	setbuf.3s	assign buffering to a stream
setbuffer	setbuf.3s	assign buffering to a stream
setegid	setuid.3	set user and group ID
seteuid	setuid.3	set user and group ID
setfsent	getfsent.3x	get file system descriptor file entry
setgid	setuid.3	set user and group ID
setgrent	getgrent.3	get group file entry
sethostent	gethostent.3n	get network host entry
setjmp	setjmp.3	non-local goto
setkey	crypt.3	DES encryption
setlinebuf	setbuf.3s	assign buffering to a stream
setnetent	getnetent.3n	get network entry
setprotoent	getprotoent.3n	get protocol entry
setpwent	getpwent.3	get password file entry
setrgid	setuid.3	set user and group ID
setruid	setuid.3	set user and group ID
setservent	getservent.3n	get service entry
setstate	random.3	better random number generator
setuid	setuid.3	set user and group ID
short	long.3f	integer object conversion
signal	signal.3	simplified software signal facilities
signal	signal.3f	change the action for a signal
sin	sin.3m	trigonometric functions
sinh	sinh.3m	hyperbolic functions
sleep	sleep.3	suspend execution for interval
sleep	sleep.3f	suspend execution for an interval
space	plot.3x	graphics interface
sprintf	printf.3s	formatted output conversion
sqrt	exp.3m	exponential, logarithm, power, square root
srand	rand.3c	random number generator
srandom	random.3	better random number generator
sscanf	scanf.3s	formatted input conversion
stat	stat.3f	get file status
stdio	intro.3s	standard buffered input/output package
store	dbm.3x	data base subroutines
strcat	string.3	string operations
strcmp	string.3	string operations
strcpy	string.3	string operations
strlen	string.3	string operations
strncat	string.3	string operations

strncmp	string.3	string operations
strncpy	string.3	string operations
stty	stty.3c	set and get terminal state (defunct)
swab	swab.3	swap bytes
sys_errlist	perror.3	system error messages
sys_nerr	perror.3	system error messages
sys_siglist	psignal.3	system signal messages
syslog	syslog.3	control system log
system	system.3	issue a shell command
system	system.3f	execute a UNIX command
tan	sin.3m	trigonometric functions
tanh	sinh.3m	hyperbolic functions
tclose	topen.3f	f77 tape I/O
telldir	directory.3	directory operations
tgetent	termcap.3x	terminal independent operation routines
tgetflag	termcap.3x	terminal independent operation routines
tgetnum	termcap.3x	terminal independent operation routines
tgetstr	termcap.3x	terminal independent operation routines
tgoto	termcap.3x	terminal independent operation routines
time	time.3c	get date and time
time	time.3f	return system time
times	times.3c	get process times
timezone	ctime.3	convert date and time to ASCII
topen	topen.3f	f77 tape I/O
tputs	termcap.3x	terminal independent operation routines
traper	traper.3f	trap arithmetic errors
trapov	trapov.3f	trap and repair floating point overflow
tread	topen.3f	f77 tape I/O
trewin	topen.3f	f77 tape I/O
trpfpe	trpfpe.3f	trap and repair floating point faults
tskipf	topen.3f	f77 tape I/O
tstate	topen.3f	f77 tape I/O
ttynam	ttynam.3f	find name of a terminal port
ttyname	ttyname.3	find name of a terminal
ttyslot	ttyname.3	find name of a terminal
twrite	topen.3f	f77 tape I/O
ungetc	ungetc.3s	push character back into input stream
unlink	unlink.3f	remove a directory entry
utime	utime.3c	set file times
valloc	valloc.3	aligned memory allocator
varargs	varargs.3	variable argument list
vlimit	vlimit.3c	control maximum system resource consumption
vtimes	vtimes.3c	get information about resource utilization
wait	wait.3f	wait for a process to terminate
y0	j0.3m	bessel functions
y1	j0.3m	bessel functions
yn	j0.3m	bessel functions

NAME

abort – generate a fault

DESCRIPTION

Abort executes an instruction which is illegal in user mode. This causes a signal that normally terminates the process with a core dump, which may be used for debugging.

SEE ALSO

adb(1), sigvec(2), exit(2)

DIAGNOSTICS

Usually 'IOT trap – core dumped' from the shell.

BUGS

The abort() function does not flush standard I/O buffers. Use *fflush*(3S).

NAME
abs – integer absolute value

SYNOPSIS
```
abs(i)
int i;
```

DESCRIPTION
Abs returns the absolute value of its integer operand.

SEE ALSO
floor(3M) for *fabs*

BUGS
Applying the *abs* function to the most negative integer generates a result which is the most negative integer. That is,

abs(0x80000000)

returns 0x80000000 as a result.

NAME

atof, atoi, atol – convert ASCII to numbers

SYNOPSIS

double atof(nptr)
char *nptr;

atoi(nptr)
char *nptr;

long atol(nptr)
char *nptr;

DESCRIPTION

These functions convert a string pointed to by *nptr* to floating, integer, and long integer representation respectively. The first unrecognized character ends the string.

Atof recognizes an optional string of spaces, then an optional sign, then a string of digits optionally containing a decimal point, then an optional 'e' or 'E' followed by an optionally signed integer.

Atoi and *atol* recognize an optional string of spaces, then an optional sign, then a string of digits.

SEE ALSO

scanf(3S)

BUGS

There are no provisions for overflow.

NAME

bcopy, bcmp, bzero, ffs – bit and byte string operations

SYNOPSIS

bcopy(b1, b2, length)
char *b1, *b2;
int length;

bcmp(b1, b2, length)
char *b1, *b2;
int length;

bzero(b, length)
char *b;
int length;

ffs(i)
int i;

DESCRIPTION

The functions *bcopy*, *bcmp*, and *bzero* operate on variable length strings of bytes. They do not check for null bytes as the routines in *string*(3) do.

Bcopy copies *length* bytes from string *b1* to the string *b2*.

Bcmp compares byte string *b1* against byte string *b2*, returning zero if they are identical, non-zero otherwise. Both strings are assumed to be *length* bytes long.

Bzero places *length* 0 bytes in the string *b1*.

Ffs find the first bit set in the argument passed it and returns the index of that bit. Bits are numbered starting at 1. A return value of –1 indicates the value passed is zero.

BUGS

The *bcmp* and *bcopy* routines take parameters backwards from *strcmp* and *strcpy*.

NAME

crypt, setkey, encrypt – DES encryption

SYNOPSIS

char *crypt(key, salt)
char *key, *salt;

setkey(key)
char *key;

encrypt(block, edflag)
char *block;

DESCRIPTION

Crypt is the password encryption routine. It is based on the NBS Data Encryption Standard, with variations intended (among other things) to frustrate use of hardware implementations of the DES for key search.

The first argument to *crypt* is normally a user's typed password. The second is a 2-character string chosen from the set [a-zA-Z0-9./]. The *salt* string is used to perturb the DES algorithm in one of 4096 different ways, after which the password is used as the key to encrypt repeatedly a constant string. The returned value points to the encrypted password, in the same alphabet as the salt. The first two characters are the salt itself.

The other entries provide (rather primitive) access to the actual DES algorithm. The argument of *setkey* is a character array of length 64 containing only the characters with numerical value 0 and 1. If this string is divided into groups of 8, the low-order bit in each group is ignored, leading to a 56-bit key which is set into the machine.

The argument to the *encrypt* entry is likewise a character array of length 64 containing 0's and 1's. The argument array is modified in place to a similar array representing the bits of the argument after having been subjected to the DES algorithm using the key set by *setkey*. If *edflag* is 0, the argument is encrypted; if non-zero, it is decrypted.

SEE ALSO

passwd(1), passwd(5), login(1), getpass(3)

BUGS

The return value points to static data whose content is overwritten by each call.

NAME

ctime, localtime, gmtime, asctime, timezone – convert date and time to ASCII

SYNOPSIS

```
char *ctime(clock)
long *clock;

#include <sys/time.h>

struct tm *localtime(clock)
long *clock;

struct tm *gmtime(clock)
long *clock;

char *asctime(tm)
struct tm *tm;

char *timezone(zone, dst)
```

DESCRIPTION

Ctime converts a time pointed to by *clock* such as returned by *time*(2) into ASCII and returns a pointer to a 26-character string in the following form. All the fields have constant width.

```
Sun Sep 16 01:03:52 1973\n\0
```

Localtime and *gmtime* return pointers to structures containing the broken-down time. *Localtime* corrects for the time zone and possible daylight savings time; *gmtime* converts directly to GMT, which is the time UNIX uses. *Asctime* converts a broken-down time to ASCII and returns a pointer to a 26-character string.

The structure declaration from the include file is:

```
struct tm {
        int     tm_sec;
        int     tm_min;
        int     tm_hour;
        int     tm_mday;
        int     tm_mon;
        int     tm_year;
        int     tm_wday;
        int     tm_yday;
        int     tm_isdst;
};
```

These quantities give the time on a 24-hour clock, day of month (1-31), month of year (0-11), day of week (Sunday = 0), year – 1900, day of year (0-365), and a flag that is nonzero if daylight saving time is in effect.

When local time is called for, the program consults the system to determine the time zone and whether the U.S.A., Australian, Eastern European, Middle European, or Western European daylight saving time adjustment is appropriate. The program knows about various peculiarities in time conversion over the past 10-20 years; if necessary, this understanding can be extended.

Timezone returns the name of the time zone associated with its first argument, which is measured in minutes westward from Greenwich. If the second argument is 0, the standard name is used, otherwise the Daylight Saving version. If the required name does not appear in a table built into the routine, the difference from GMT is produced; e.g. in Afghanistan *timezone(-*

*(60*4+30), 0)* is appropriate because it is 4:30 ahead of GMT and the string **GMT+4:30** is produced.

SEE ALSO

gettimeofday(2), time(3)

BUGS

The return values point to static data whose content is overwritten by each call.

NAME

isalpha, isupper, islower, isdigit, isalnum, isspace, ispunct, isprint, iscntrl, isascii – character classification macros

SYNOPSIS

#include <ctype.h>

isalpha(c)

. . .

DESCRIPTION

These macros classify ASCII-coded integer values by table lookup. Each is a predicate returning nonzero for true, zero for false. *Isascii* is defined on all integer values; the rest are defined only where *isascii* is true and on the single non-ASCII value EOF (see *stdio*(3S)).

isalpha	*c* is a letter
isupper	*c* is an upper case letter
islower	*c* is a lower case letter
isdigit	*c* is a digit
isalnum	*c* is an alphanumeric character
isspace	*c* is a space, tab, carriage return, newline, or formfeed
ispunct	*c* is a punctuation character (neither control nor alphanumeric)
isprint	*c* is a printing character, code 040(8) (space) through 0176 (tilde)
iscntrl	*c* is a delete character (0177) or ordinary control character (less than 040).
isascii	*c* is an ASCII character, code less than 0200

SEE ALSO

ascii(7)

NAME

opendir, readdir, telldir, seekdir, rewinddir, closedir – directory operations

SYNOPSIS

#include <sys/dir.h>

DIR *opendir(filename)
char *filename;

struct direct *readdir(dirp)
DIR *dirp;

long telldir(dirp)
DIR *dirp;

seekdir(dirp, loc)
DIR *dirp;
long loc;

rewinddir(dirp)
DIR *dirp;

closedir(dirp)
DIR *dirp;

DESCRIPTION

Opendir opens the directory named by *filename* and associates a *directory stream* with it. *Opendir* returns a pointer to be used to identify the *directory stream* in subsequent operations. The pointer **NULL** is returned if *filename* cannot be accessed, or if it cannot *malloc*(3) enough memory to hold the whole thing.

Readdir returns a pointer to the next directory entry. It returns **NULL** upon reaching the end of the directory or detecting an invalid *seekdir* operation.

Telldir returns the current location associated with the named *directory stream.*

Seekdir sets the position of the next *readdir* operation on the *directory stream.* The new position reverts to the one associated with the *directory stream* when the *telldir* operation was performed. Values returned by *telldir* are good only for the lifetime of the DIR pointer from which they are derived. If the directory is closed and then reopened, the *telldir* value may be invalidated due to undetected directory compaction. It is safe to use a previous *telldir* value immediately after a call to *opendir* and before any calls to *readdir.*

Rewinddir resets the position of the named *directory stream* to the beginning of the directory.

Closedir closes the named *directory stream* and frees the structure associated with the DIR pointer.

Sample code which searchs a directory for entry "name" is:

```
len = strlen(name);
dirp = opendir(".");
for (dp = readdir(dirp); dp != NULL; dp = readdir(dirp))
        if (dp->d_namlen == len && !strcmp(dp->d_name, name)) {
                closedir(dirp);
                return FOUND;
        }
closedir(dirp);
return NOT_FOUND;
```

SEE ALSO

open(2), close(2), read(2), lseek(2), dir(5)

NAME

ecvt, fcvt, gcvt – output conversion

SYNOPSIS

char *ecvt(value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;

char *fcvt(value, ndigit, decpt, sign)
double value;
int ndigit, *decpt, *sign;

char *gcvt(value, ndigit, buf)
double value;
char *buf;

DESCRIPTION

Ecvt converts the *value* to a null-terminated string of *ndigit* ASCII digits and returns a pointer thereto. The position of the decimal point relative to the beginning of the string is stored indirectly through *decpt* (negative means to the left of the returned digits). If the sign of the result is negative, the word pointed to by *sign* is non-zero, otherwise it is zero. The low-order digit is rounded.

Fcvt is identical to *ecvt,* except that the correct digit has been rounded for Fortran F-format output of the number of digits specified by *ndigits.*

Gcvt converts the *value* to a null-terminated ASCII string in *buf* and returns a pointer to *buf.* It attempts to produce *ndigit* significant digits in Fortran F format if possible, otherwise E format, ready for printing. Trailing zeros may be suppressed.

SEE ALSO

printf(3)

BUGS

The return values point to static data whose content is overwritten by each call.

NAME

end, etext, edata – last locations in program

SYNOPSIS

extern end;
extern etext;
extern edata;

DESCRIPTION

These names refer neither to routines nor to locations with interesting contents. The address of *etext* is the first address above the program text, *edata* above the initialized data region, and *end* above the uninitialized data region.

When execution begins, the program break coincides with *end*, but it is reset by the routines *brk*(2), *malloc*(3), standard input/output (*stdio*(3)), the profile (**–p**) option of *cc*(1), etc. The current value of the program break is reliably returned by 'sbrk(0)', see *brk*(2).

SEE ALSO

brk(2), malloc(3)

NAME

execl, execv, execle, execlp, execvp, exec, exece, environ – execute a file

SYNOPSIS

execl(name, arg0, arg1, ..., argn, 0)
char *name, *arg0, *arg1, ..., *argn;

execv(name, argv)
char *name, *argv[];

execle(name, arg0, arg1, ..., argn, 0, envp)
char *name, *arg0, *arg1, ..., *argn, *envp[];

extern char **environ;

DESCRIPTION

These routines provide various interfaces to the *execve* system call. Refer to *execve*(2) for a description of their properties; only brief descriptions are provided here.

Exec in all its forms overlays the calling process with the named file, then transfers to the entry point of the core image of the file. There can be no return from a successful exec; the calling core image is lost.

The *name* argument is a pointer to the name of the file to be executed. The pointers *arg*[*0*], *arg*[*1*] ... address null-terminated strings. Conventionally *arg*[*0*] is the name of the file.

Two interfaces are available. *execl* is useful when a known file with known arguments is being called; the arguments to *execl* are the character strings constituting the file and the arguments; the first argument is conventionally the same as the file name (or its last component). A 0 argument must end the argument list.

The *execv* version is useful when the number of arguments is unknown in advance; the arguments to *execv* are the name of the file to be executed and a vector of strings containing the arguments. The last argument string must be followed by a 0 pointer.

When a C program is executed, it is called as follows:

```
main(argc, argv, envp)
int argc;
char **argv, **envp;
```

where *argc* is the argument count and *argv* is an array of character pointers to the arguments themselves. As indicated, *argc* is conventionally at least one and the first member of the array points to a string containing the name of the file.

Argv is directly usable in another *execv* because *argv*[*argc*] is 0.

Envp is a pointer to an array of strings that constitute the *environment* of the process. Each string consists of a name, an "=", and a null-terminated value. The array of pointers is terminated by a null pointer. The shell *sh*(1) passes an environment entry for each global shell variable defined when the program is called. See *environ*(7) for some conventionally used names. The C run-time start-off routine places a copy of *envp* in the global cell *environ*, which is used by *execv* and *execl* to pass the environment to any subprograms executed by the current program.

Execlp and *execvp* are called with the same arguments as *execl* and *execv*, but duplicate the shell's actions in searching for an executable file in a list of directories. The directory list is obtained from the environment.

FILES

/bin/sh shell, invoked if command file found by *execlp* or *execvp*

SEE ALSO

execve(2), fork(2), environ(7), csh(1)

DIAGNOSTICS

If the file cannot be found, if it is not executable, if it does not start with a valid magic number (see *a.out*(5)), if maximum memory is exceeded, or if the arguments require too much space, a return constitutes the diagnostic; the return value is -1. Even for the super-user, at least one of the execute-permission bits must be set for a file to be executed.

BUGS

If *execvp* is called to execute a file that turns out to be a shell command file, and if it is impossible to execute the shell, the values of *argv[0]* and *argv[-1]* will be modified before return.

NAME
exit – terminate a process after flushing any pending output

SYNOPSIS
exit(status)
int status;

DESCRIPTION
Exit terminates a process after calling the Standard I/O library function *_cleanup* to flush any buffered output. *Exit* never returns.

SEE ALSO
exit(2), intro(3S)

NAME

frexp, ldexp, modf – split into mantissa and exponent

SYNOPSIS

double frexp(value, eptr)
double value;
int *eptr;

double ldexp(value, exp)
double value;

double modf(value, iptr)
double value, *iptr;

DESCRIPTION

Frexp returns the mantissa of a double *value* as a double quantity, *x*, of magnitude less than 1 and stores an integer *n* such that $value = x*2^n$ indirectly through *eptr*.

Ldexp returns the quantity $value * 2^{exp}$.

Modf returns the positive fractional part of *value* and stores the integer part indirectly through *iptr*.

NAME

getenv – value for environment name

SYNOPSIS

char *getenv(name)
char *name;

DESCRIPTION

Getenv searches the environment list (see *environ*(7)) for a string of the form *name*=*value* and returns a pointer to the string *value* if such a string is present, otherwise *getenv* returns the value 0 (NULL).

SEE ALSO

environ(7), execve(2)

NAME

getgrent, getgrgid, getgrnam, setgrent, endgrent – get group file entry

SYNOPSIS

#include <grp.h>

struct group *getgrent()

struct group *getgrgid(gid)
int gid;

struct group *getgrnam(name)
char *name;

setgrent()

endgrent()

DESCRIPTION

Getgrent, getgrgid and *getgrnam* each return pointers to an object with the following structure containing the broken-out fields of a line in the group file.

```
/*      grp.h   4.1     83/05/03        */

struct  group { /* see getgrent(3) */
        char    *gr_name;
        char    *gr_passwd;
        int     gr_gid;
        char    **gr_mem;
};

struct group *getgrent(), *getgrgid(), *getgrnam();
```

The members of this structure are:

gr_name	The name of the group.
gr_passwd	The encrypted password of the group.
gr_gid	The numerical group-ID.
gr_mem	Null-terminated vector of pointers to the individual member names.

Getgrent simply reads the next line while *getgrgid* and *getgrnam* search until a matching *gid* or *name* is found (or until EOF is encountered). Each routine picks up where the others leave off so successive calls may be used to search the entire file.

A call to *setgrent* has the effect of rewinding the group file to allow repeated searches. *Endgrent* may be called to close the group file when processing is complete.

FILES

/etc/group

SEE ALSO

getlogin(3), getpwent(3), group(5)

DIAGNOSTICS

A null pointer (0) is returned on EOF or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved.

NAME

getlogin – get login name

SYNOPSIS

char *getlogin()

DESCRIPTION

Getlogin returns a pointer to the login name as found in */etc/utmp*. It may be used in conjunction with *getpwnam* to locate the correct password file entry when the same userid is shared by several login names.

If *getlogin* is called within a process that is not attached to a typewriter, it returns NULL. The correct procedure for determining the login name is to first call *getlogin* and if it fails, to call *getpw*(*getuid*()).

FILES

/etc/utmp

SEE ALSO

getpwent(3), getgrent(3), utmp(5), getpw(3)

DIAGNOSTICS

Returns NULL (0) if name not found.

BUGS

The return values point to static data whose content is overwritten by each call.

NAME
getpass – read a password

SYNOPSIS
char *getpass(prompt)
char *prompt;

DESCRIPTION
Getpass reads a password from the file */dev/tty*, or if that cannot be opened, from the standard input, after prompting with the null-terminated string *prompt* and disabling echoing. A pointer is returned to a null-terminated string of at most 8 characters.

FILES
/dev/tty

SEE ALSO
crypt(3)

BUGS
The return value points to static data whose content is overwritten by each call.

NAME

getpwent, getpwuid, getpwnam, setpwent, endpwent – get password file entry

SYNOPSIS

#include <pwd.h>

struct passwd *getpwent()

struct passwd *getpwuid(uid)
int uid;

struct passwd *getpwnam(name)
char *name;

int setpwent()

int endpwent()

DESCRIPTION

Getpwent, getpwuid and *getpwnam* each return a pointer to an object with the following structure containing the broken-out fields of a line in the password file.

```
/*      pwd.h  4.1     83/05/03        */

struct  passwd { /* see getpwent(3) */
        char    *pw_name;
        char    *pw_passwd;
        int     pw_uid;
        int     pw_gid;
        int     pw_quota;
        char    *pw_comment;
        char    *pw_gecos;
        char    *pw_dir;
        char    *pw_shell;
};

struct passwd *getpwent(), *getpwuid(), *getpwnam();
```

The fields *pw_quota* and *pw_comment* are unused; the others have meanings described in *passwd*(5).

Getpwent reads the next line (opening the file if necessary); *setpwent* rewinds the file; *endpwent* closes it.

Getpwuid and *getpwnam* search from the beginning until a matching *uid* or *name* is found (or until EOF is encountered).

FILES

/etc/passwd

SEE ALSO

getlogin(3), getgrent(3), passwd(5)

DIAGNOSTICS

Null pointer (0) returned on EOF or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved.

NAME

getwd – get current working directory pathname

SYNOPSIS

char *getwd(pathname)
char *pathname;

DESCRIPTION

Getwd copies the absolute pathname of the current working directory to *pathname* and returns a pointer to the result.

LIMITATIONS

Maximum pathname length is MAXPATHLEN characters (1024).

DIAGNOSTICS

Getwd returns zero and places a message in *pathname* if an error occurs.

BUGS

Getwd may fail to return to the current directory if an error occurs.

NAME

insque, remque – insert/remove element from a queue

SYNOPSIS

```
struct qelem {
        struct qelem *q_forw;
        struct qelem *q_back;
        char   q_data[];
};

insque(elem, pred)
struct qelem *elem, *pred;

remque(elem)
struct qelem *elem;
```

DESCRIPTION

Insque and *remque* manipulate queues built from doubly linked lists. Each element in the queue must in the form of "struct qelem". *Insque* inserts *elem* in a queue imediately after *pred*; *remque* removes an entry *elem* from a queue.

NAME

malloc, free, realloc, calloc, alloca – memory allocator

SYNOPSIS

char *malloc(size)
unsigned size;

free(ptr)
char *ptr;

char *realloc(ptr, size)
char *ptr;
unsigned size;

char *calloc(nelem, elsize)
unsigned nelem, elsize;

char *alloca(size)
int size;

DESCRIPTION

Malloc and *free* provide a simple general-purpose memory allocation package. *Malloc* returns a pointer to a block of at least *size* bytes beginning on a word boundary.

The argument to *free* is a pointer to a block previously allocated by *malloc*; this space is made available for further allocation, but its contents are left undisturbed.

Needless to say, grave disorder will result if the space assigned by *malloc* is overrun or if some random number is handed to *free*.

Malloc maintains multiple lists of free blocks according to size, allocating space from the appropriate list. It calls *sbrk* (see *brk*(2)) to get more memory from the system when there is no suitable space already free.

Realloc changes the size of the block pointed to by *ptr* to *size* bytes and returns a pointer to the (possibly moved) block. The contents will be unchanged up to the lesser of the new and old sizes.

In order to be compatible with older versions, *realloc* also works if *ptr* points to a block freed since the last call of *malloc, realloc* or *calloc*; sequences of *free, malloc* and *realloc* were previously used to attempt storage compaction. This procedure is no longer recommended.

Calloc allocates space for an array of *nelem* elements of size *elsize*. The space is initialized to zeros.

Alloca allocates *size* bytes of space in the stack frame of the caller. This temporary space is automatically freed on return.

Each of the allocation routines returns a pointer to space suitably aligned (after possible pointer coercion) for storage of any type of object.

DIAGNOSTICS

Malloc, realloc and *calloc* return a null pointer (0) if there is no available memory or if the arena has been detectably corrupted by storing outside the bounds of a block. *Malloc* may be recompiled to check the arena very stringently on every transaction; those sites with a source code license may check the source code to see how this can be done.

BUGS

When *realloc* returns 0, the block pointed to by *ptr* may be destroyed.

Alloca is machine dependent; it's use is discouraged.

NAME

mktemp – make a unique file name

SYNOPSIS

char *mktemp(template)
char *template;

DESCRIPTION

Mktemp replaces *template* by a unique file name, and returns the address of the template. The template should look like a file name with six trailing X's, which will be replaced with the current process id and a unique letter.

SEE ALSO

getpid(2)

NAME

monitor, monstartup, moncontrol – prepare execution profile

SYNOPSIS

monitor(lowpc, highpc, buffer, bufsize, nfunc)
int (*lowpc)(), (*highpc)();
short buffer[];

monstartup(lowpc, highpc)
int (*lowpc)(), (*highpc)();

moncontrol(mode)

DESCRIPTION

There are two different forms of monitoring available: An executable program created by:

```
cc –p . . .
```

automatically includes calls for the *prof*(1) monitor and includes an initial call to its start-up routine *monstartup* with default parameters; *monitor* need not be called explicitly except to gain fine control over profil buffer allocation. An executable program created by:

```
cc –pg . . .
```

automatically includes calls for the *gprof*(1) monitor.

Monstartup is a high level interface to *profil*(2). *Lowpc* and *highpc* specify the address range that is to be sampled; the lowest address sampled is that of *lowpc* and the highest is just below *highpc*. *Monstartup* allocates space using *sbrk*(2) and passes it to *monitor* (see below) to record a histogram of periodically sampled values of the program counter, and of counts of calls of certain functions, in the buffer. Only calls of functions compiled with the profiling option **–p** of *cc*(1) are recorded.

To profile the entire program, it is sufficient to use

```
extern etext();
. . .
monstartup((int) 2, etext);
```

Etext lies just above all the program text, see *end*(3).

To stop execution monitoring and write the results on the file *mon.out,* use

```
monitor(0);
```

then *prof*(1) can be used to examine the results.

Moncontrol is used to selectively control profiling within a program. This works with either *prof*(1) or *gprof*(1) type profiling. When the program starts, profiling begins. To stop the collection of histogram ticks and call counts use *moncontrol*(0); to resume the collection of histogram ticks and call counts use *moncontrol*(1). This allows the cost of particular operations to be measured. Note that an output file will be produced upon program exit irregardless of the state of *moncontrol.*

Monitor is a low level interface to *profil*(2). *Lowpc* and *highpc* are the addresses of two functions; *buffer* is the address of a (user supplied) array of *bufsize* short integers. At most *nfunc* call counts can be kept. For the results to be significant, especially where there are small, heavily used routines, it is suggested that the buffer be no more than a few times smaller than the range of locations sampled. *Monitor* divides the buffer into space to record the histogram of program counter samples over the range *lowpc* to *highpc*, and space to record call counts of functions compiled with the **−p** option to *cc*(1).

To profile the entire program, it is sufficient to use

```
        extern etext();
        . . .
        monitor((int) 2, etext, buf, bufsize, nfunc);
```

FILES

mon.out

SEE ALSO

cc(1), prof(1), gprof(1), profil(2), sbrk(2)

NAME

nlist – get entries from name list

SYNOPSIS

#include <nlist.h>

nlist(filename, nl)
char *filename;
struct nlist nl[];

DESCRIPTION

Nlist examines the name list in the given executable output file and selectively extracts a list of values. The name list consists of an array of structures containing names, types and values. The list is terminated with a null name. Each name is looked up in the name list of the file. If the name is found, the type and value of the name are inserted in the next two fields. If the name is not found, both entries are set to 0. See *a.out*(5) for the structure declaration.

This subroutine is useful for examining the system name list kept in the file **/vmunix**. In this way programs can obtain system addresses that are up to date.

SEE ALSO

a.out(5)

DIAGNOSTICS

All type entries are set to 0 if the file cannot be found or if it is not a valid namelist.

NAME

perror, sys_errlist, sys_nerr – system error messages

SYNOPSIS

perror(s)
char *s;

int sys_nerr;
char *sys_errlist[];

DESCRIPTION

Perror produces a short error message on the standard error file describing the last error encountered during a call to the system from a C program. First the argument string *s* is printed, then a colon, then the message and a new-line. Most usefully, the argument string is the name of the program which incurred the error. The error number is taken from the external variable *errno* (see *intro*(2)), which is set when errors occur but not cleared when non-erroneous calls are made.

To simplify variant formatting of messages, the vector of message strings *sys_errlist* is provided; *errno* can be used as an index in this table to get the message string without the new-line. *Sys_nerr* is the number of messages provided for in the table; it should be checked because new error codes may be added to the system before they are added to the table.

SEE ALSO

intro(2), psignal(3)

NAME

popen, pclose – initiate I/O to/from a process

SYNOPSIS

#include <stdio.h>

FILE *popen(command, type)
char *command, *type;

pclose(stream)
FILE *stream;

DESCRIPTION

The arguments to *popen* are pointers to null-terminated strings containing respectively a shell command line and an I/O mode, either "r" for reading or "w" for writing. It creates a pipe between the calling process and the command to be executed. The value returned is a stream pointer that can be used (as appropriate) to write to the standard input of the command or read from its standard output.

A stream opened by *popen* should be closed by *pclose*, which waits for the associated process to terminate and returns the exit status of the command.

Because open files are shared, a type "r" command may be used as an input filter, and a type "w" as an output filter.

SEE ALSO

pipe(2), fopen(3S), fclose(3S), system(3), wait(2), sh(1)

DIAGNOSTICS

Popen returns a null pointer if files or processes cannot be created, or the shell cannot be accessed.

Pclose returns –1 if *stream* is not associated with a 'popened' command.

BUGS

Buffered reading before opening an input filter may leave the standard input of that filter mispositioned. Similar problems with an output filter may be forestalled by careful buffer flushing, for instance, with *fflush,* see *fclose*(3).

Popen always calls *sh*, never calls *csh*.

NAME

psignal, sys_siglist – system signal messages

SYNOPSIS

psignal(sig, s)
unsigned sig;
char *s;

char *sys_siglist[];

DESCRIPTION

Psignal produces a short message on the standard error file describing the indicated signal. First the argument string *s* is printed, then a colon, then the name of the signal and a newline. Most usefully, the argument string is the name of the program which incurred the signal. The signal number should be from among those found in <*signal.h*>.

To simplify variant formatting of signal names, the vector of message strings *sys_siglist* is provided; the signal number can be used as an index in this table to get the signal name without the newline. The define NSIG defined in <*signal.h*> is the number of messages provided for in the table; it should be checked because new signals may be added to the system before they are added to the table.

SEE ALSO

sigvec(2), perror(3)

NAME

qsort – quicker sort

SYNOPSIS

qsort(base, nel, width, compar)
char *base;
int (*compar)();

DESCRIPTION

Qsort is an implementation of the quicker-sort algorithm. The first argument is a pointer to the base of the data; the second is the number of elements; the third is the width of an element in bytes; the last is the name of the comparison routine to be called with two arguments which are pointers to the elements being compared. The routine must return an integer less than, equal to, or greater than 0 according as the first argument is to be considered less than, equal to, or greater than the second.

SEE ALSO

sort(1)

NAME

random, srandom, initstate, setstate – better random number generator; routines for changing generators

SYNOPSIS

long random()

srandom(seed)
int seed;

char *initstate(seed, state, n)
unsigned seed;
char *state;
int n;

char *setstate(state)
char *state;

DESCRIPTION

Random uses a non-linear additive feedback random number generator employing a default table of size 31 long integers to return successive pseudo-random numbers in the range from 0 to $2^{31}-1$. The period of this random number generator is very large, approximately $16*(2^{31}-1)$.

Random/srandom have (almost) the same calling sequence and initialization properties as *rand/srand.* The difference is that *rand*(3) produces a much less random sequence -- in fact, the low dozen bits generated by rand go through a cyclic pattern. All the bits generated by *random* are usable. For example, "random()&01" will produce a random binary value.

Unlike *srand*, *srandom* does not return the old seed; the reason for this is that the amount of state information used is much more than a single word. (Two other routines are provided to deal with restarting/changing random number generators). Like *rand*(3), however, *random* will by default produce a sequence of numbers that can be duplicated by calling *srandom* with *1* as the seed.

The *initstate* routine allows a state array, passed in as an argument, to be initialized for future use. The size of the state array (in bytes) is used by *initstate* to decide how sophisticated a random number generator it should use -- the more state, the better the random numbers will be. (Current "optimal" values for the amount of state information are 8, 32, 64, 128, and 256 bytes; other amounts will be rounded down to the nearest known amount. Using less than 8 bytes will cause an error). The seed for the initialization (which specifies a starting point for the random number sequence, and provides for restarting at the same point) is also an argument. *Initstate* returns a pointer to the previous state information array.

Once a state has been initialized, the *setstate* routine provides for rapid switching between states. *Setstate returns a pointer to the* argument state array is used for further random number generation until the next call to *initstate* or *setstate.*

Once a state array has been initialized, it may be restarted at a different point either by calling *initstate* (with the desired seed, the state array, and its size) or by calling both *setstate* (with the state array) and *srandom* (with the desired seed). The advantage of calling both *setstate* and *srandom* is that the size of the state array does not have to be remembered after it is initialized.

With 256 bytes of state information, the period of the random number generator is greater than 2^{69}, which should be sufficient for most purposes.

AUTHOR
Earl T. Cohen

DIAGNOSTICS
If *initstate* is called with less than 8 bytes of state information, or if *setstate* detects that the state information has been garbled, error messages are printed on the standard error output.

SEE ALSO
rand(3)

BUGS
About 2/3 the speed of *rand*(3C).

NAME

re_comp, re_exec – regular expression handler

SYNOPSIS

char *re_comp(s)
char *s;

re_exec(s)
char *s;

DESCRIPTION

Re_comp compiles a string into an internal form suitable for pattern matching. *Re_exec* checks the argument string against the last string passed to *re_comp*.

Re_comp returns 0 if the string *s* was compiled successfully; otherwise a string containing an error message is returned. If *re_comp* is passed 0 or a null string, it returns without changing the currently compiled regular expression.

Re_exec returns 1 if the string *s* matches the last compiled regular expression, 0 if the string *s* failed to match the last compiled regular expression, and –1 if the compiled regular expression was invalid (indicating an internal error).

The strings passed to both *re_comp* and *re_exec* may have trailing or embedded newline characters; they are terminated by nulls. The regular expressions recognized are described in the manual entry for *ed*(1), given the above difference.

SEE ALSO

ed(1), ex(1), egrep(1), fgrep(1), grep(1)

DIAGNOSTICS

Re_exec returns –1 for an internal error.

Re_comp returns one of the following strings if an error occurs:

> *No previous regular expression,*
> *Regular expression too long,*
> *unmatched \(,*
> *missing],*
> *too many \(\) pairs,*
> *unmatched \).*

NAME

scandir – scan a directory

SYNOPSIS

```
#include <sys/types.h>
#include <sys/dir.h>

scandir(dirname, namelist, select, compar)
char *dirname;
struct direct *(*namelist[]);
int (*select)();
int (*compar)();

alphasort(d1, d2)
struct direct **d1, **d2;
```

DESCRIPTION

Scandir reads the directory *dirname* and builds an array of pointers to directory entries using *malloc*(3). It returns the number of entries in the array and a pointer to the array through *namelist*.

The *select* parameter is a pointer to a user supplied subroutine which is called by *scandir* to select which entries are to be included in the array. The select routine is passed a pointer to a directory entry and should return a non-zero value if the directory entry is to be included in the array. If *select* is null, then all the directory entries will be included.

The *compar* parameter is a pointer to a user supplied subroutine which is passed to *qsort*(3) to sort the completed array. If this pointer is null, the array is not sorted. *Alphasort* is a routine which can be used for the *compar* parameter to sort the array alphabetically.

The memory allocated for the array can be deallocated with *free* (see *malloc*(3)) by freeing each pointer in the array and the array itself.

SEE ALSO

directory(3), malloc(3), qsort(3), dir(5)

DIAGNOSTICS

Returns -1 if the directory cannot be opened for reading or if *malloc*(3) cannot allocate enough memory to hold all the data structures.

NAME

setjmp, longjmp – non-local goto

SYNOPSIS

#include <setjmp.h>

setjmp(env)
jmp_buf env;

longjmp(env, val)
jmp_buf env;

_setjmp(env)
jmp_buf env;

_longjmp(env, val)
jmp_buf env;

DESCRIPTION

These routines are useful for dealing with errors and interrupts encountered in a low-level subroutine of a program.

Setjmp saves its stack environment in *env* for later use by *longjmp.* It returns value 0.

Longjmp restores the environment saved by the last call of *setjmp.* It then returns in such a way that execution continues as if the call of *setjmp* had just returned the value *val* to the function that invoked *setjmp,* which must not itself have returned in the interim. All accessible data have values as of the time *longjmp* was called.

Setjmp and *longjmp* save and restore the signal mask *sigmask*(2), while *_setjmp* and *_longjmp* manipulate only the C stack and registers.

SEE ALSO

sigvec(2), sigstack(2), signal(3)

BUGS

Setjmp does not save current notion of whether the process is executing on the signal stack. The result is that a longjmp to some place on the signal stack leaves the signal stack state incorrect.

NAME

setuid, seteuid, setruid, setgid, setegid, setrgid – set user and group ID

SYNOPSIS

setuid(uid)
seteuid(euid)
setruid(ruid)

setgid(gid)
setegid(egid)
setrgid(rgid)

DESCRIPTION

Setuid (*setgid*) sets both the real and effective user ID (group ID) of the current process to as specified.

Seteuid (*setegid*) sets the effective user ID (group ID) of the current process.

Setruid (*setruid*) sets the real user ID (group ID) of the current process.

These calls are only permitted to the super-user or if the argument is the real or effective ID.

SEE ALSO

setreuid(2), setregid(2), getuid(2), getgid(2)

DIAGNOSTICS

Zero is returned if the user (group) ID is set; –1 is returned otherwise.

NAME

sleep – suspend execution for interval

SYNOPSIS

sleep(seconds)
unsigned seconds;

DESCRIPTION

The current process is suspended from execution for the number of seconds specified by the argument. The actual suspension time may be up to 1 second less than that requested, because scheduled wakeups occur at fixed 1-second intervals, and an arbitrary amount longer because of other activity in the system.

The routine is implemented by setting an interval timer and pausing until it occurs. The previous state of this timer is saved and restored. If the sleep time exceeds the time to the expiration of the previous timer, the process sleeps only until the signal would have occurred, and the signal is sent 1 second later.

SEE ALSO

setitimer(2), sigpause(2)

BUGS

An interface with finer resolution is needed.

NAME

strcat, strncat, strcmp, strncmp, strcpy, strncpy, strlen, index, rindex – string operations

SYNOPSIS

```
#include <strings.h>

char *strcat(s1, s2)
char *s1, *s2;

char *strncat(s1, s2, n)
char *s1, *s2;

strcmp(s1, s2)
char *s1, *s2;

strncmp(s1, s2, n)
char *s1, *s2;

char *strcpy(s1, s2)
char *s1, *s2;

char *strncpy(s1, s2, n)
char *s1, *s2;

strlen(s)
char *s;

char *index(s, c)
char *s, c;

char *rindex(s, c)
char *s, c;
```

DESCRIPTION

These functions operate on null-terminated strings. They do not check for overflow of any receiving string.

Strcat appends a copy of string *s2* to the end of string *s1*. *Strncat* copies at most *n* characters. Both return a pointer to the null-terminated result.

Strcmp compares its arguments and returns an integer greater than, equal to, or less than 0, according as *s1* is lexicographically greater than, equal to, or less than *s2*. *Strncmp* makes the same comparison but looks at at most *n* characters.

Strcpy copies string *s2* to *s1*, stopping after the null character has been moved. *Strncpy* copies exactly *n* characters, truncating or null-padding *s2*; the target may not be null-terminated if the length of *s2* is *n* or more. Both return *s1*.

Strlen returns the number of non-null characters in *s*.

Index (*rindex*) returns a pointer to the first (last) occurrence of character *c* in string *s*, or zero if *c* does not occur in the string.

NAME

swab – swap bytes

SYNOPSIS

swab(from, to, nbytes)
char *from, *to;

DESCRIPTION

Swab copies *nbytes* bytes pointed to by *from* to the position pointed to by *to,* exchanging adjacent even and odd bytes. It is useful for carrying binary data between PDP11's and other machines. *Nbytes* should be even.

NAME

syslog, openlog, closelog – control system log

SYNOPSIS

#include <syslog.h>

openlog(ident, logstat)
char *ident;

syslog(priority, message, parameters ...)
char *message;

closelog()

DESCRIPTION

Syslog arranges to write the *message* onto the system log maintained by *syslog*(8). The message is tagged with *priority*. The message looks like a *printf*(3) string except that **%m** is replaced by the current error message (collected from *errno*). A trailing newline is added if needed. This message will be read by *syslog*(8) and output to the system console or files as appropriate.

If special processing is needed, *openlog* can be called to initialize the log file. Parameters are *ident* which is prepended to every message, and *logstat* which is a bit field indicating special status; current values are:

LOG_PID	log the process id with each message: useful for identifying instantiations of daemons.

Openlog returns zero on success. If it cannot open the file */dev/log,* it writes on */dev/console* instead and returns -1.

Closelog can be used to close the log file.

EXAMPLES

```
syslog(LOG_SALERT, "who: internal error 23");

openlog("serverftp", LOG_PID);
syslog(LOG_INFO, "Connection from host %d", CallingHost);
```

SEE ALSO

syslog(8)

NAME

system – issue a shell command

SYNOPSIS

system(string)
char *string;

DESCRIPTION

System causes the *string* to be given to *sh*(1) as input as if the string had been typed as a command at a terminal. The current process waits until the shell has completed, then returns the exit status of the shell.

SEE ALSO

popen(3S), execve(2), wait(2)

DIAGNOSTICS

Exit status 127 indicates the shell couldn't be executed.

NAME

ttyname, isatty, ttyslot – find name of a terminal

SYNOPSIS

char *ttyname(filedes)

isatty(filedes)

ttyslot()

DESCRIPTION

Ttyname returns a pointer to the null-terminated path name of the terminal device associated with file descriptor *filedes* (this is a system file descriptor and has nothing to do with the standard I/O FILE typedef).

Isatty returns 1 if *filedes* is associated with a terminal device, 0 otherwise.

Ttyslot returns the number of the entry in the *ttys*(5) file for the control terminal of the current process.

FILES

/dev/*
/etc/ttys

SEE ALSO

ioctl(2), ttys(5)

DIAGNOSTICS

Ttyname returns a null pointer (0) if *filedes* does not describe a terminal device in directory '/dev'.

Ttyslot returns 0 if '/etc/ttys' is inaccessible or if it cannot determine the control terminal.

BUGS

The return value points to static data whose content is overwritten by each call.

NAME

valloc – aligned memory allocator

SYNOPSIS

char *valloc(size)
unsigned size;

DESCRIPTION

Valloc allocates *size* bytes aligned on a page boundary. It is implemented by calling *malloc*(3) with a slightly larger request, saving the true beginning of the block allocated, and returning a properly aligned pointer.

DIAGNOSTICS

Valloc returns a null pointer (0) if there is no available memory or if the arena has been detectably corrupted by storing outside the bounds of a block.

BUGS

Vfree isn't implemented.

NAME

varargs – variable argument list

SYNOPSIS

#include <varargs.h>

function(**va_alist**)
va_dcl
va_list *pvar*;
va_start(*pvar*);
f = va_arg(*pvar*, *type*);
va_end(*pvar*);

DESCRIPTION

This set of macros provides a means of writing portable procedures that accept variable argument lists. Routines having variable argument lists (such as *printf*(3)) that do not use varargs are inherently nonportable, since different machines use different argument passing conventions.

va_alist is used in a function header to declare a variable argument list.

va_dcl is a declaration for **va_alist**. Note that there is no semicolon after **va_dcl.**

va_list is a type which can be used for the variable *pvar*, which is used to traverse the list. One such variable must always be declared.

va_start(pvar) is called to initialize *pvar* to the beginning of the list.

va_arg(*pvar*, *type*) will return the next argument in the list pointed to by *pvar*. *Type* is the type the argument is expected to be. Different types can be mixed, but it is up to the routine to know what type of argument is expected, since it cannot be determined at runtime.

va_end(*pvar*) is used to finish up.

Multiple traversals, each bracketed by **va_start** ... **va_end,** are possible.

EXAMPLE

```
#include <varargs.h>
execl(va_alist)
va_dcl
{
   va_list ap;
   char *file;
   char *args[100];
   int argno = 0;

   va_start(ap);
   file = va_arg(ap, char *);
   while (args[argno++] = va_arg(ap, char *))
   ;
   va_end(ap);
   return execv(file, args);
}
```

BUGS

It is up to the calling routine to determine how many arguments there are, since it is not possible to determine this from the stack frame. For example, *execl* passes a 0 to signal the end of the list. *Printf* can tell how many arguments are supposed to be there by the format.

SECTION 3C

ICON/UXB OPERATING SYSTEM COMPATIBILITY LIBRARY SUBROUTINES

ICON®

NAME

intro – introduction to compatibility library functions

DESCRIPTION

These functions constitute the compatibility library portion of *libc*. They are automatically loaded as needed by the C compiler *cc*(1). The link editor searches this library under the "–lc" option. Use of these routines should, for the most part, be avoided. Manual entries for the functions in this library describe the proper routine to use.

LIST OF FUNCTIONS

Name	*Appears on Page*	*Description*
alarm	alarm.3c	schedule signal after specified time
ftime	time.3c	get date and time
getpw	getpw.3c	get name from uid
gtty	stty.3c	set and get terminal state (defunct)
nice	nice.3c	set program priority
pause	pause.3c	stop until signal
rand	rand.3c	random number generator
signal	signal.3c	simplified software signal facilities
srand	rand.3c	random number generator
stty	stty.3c	set and get terminal state (defunct)
time	time.3c	get date and time
times	times.3c	get process times
utime	utime.3c	set file times
vlimit	vlimit.3c	control maximum system resource consumption
vtimes	vtimes.3c	get information about resource utilization

NAME

alarm – schedule signal after specified time

SYNOPSIS

alarm(seconds)
unsigned seconds;

DESCRIPTION

This interface is obsoleted by setitimer(2).

Alarm causes signal SIGALRM, see *signal*(3C), to be sent to the invoking process in a number of seconds given by the argument. Unless caught or ignored, the signal terminates the process.

Alarm requests are not stacked; successive calls reset the alarm clock. If the argument is 0, any alarm request is canceled. Because of scheduling delays, resumption of execution of when the signal is caught may be delayed an arbitrary amount. The longest specifiable delay time is 2147483647 seconds.

The return value is the amount of time previously remaining in the alarm clock.

SEE ALSO

sigpause(2), sigvec(2), signal(3C), sleep(3)

NAME

getpw – get name from uid

SYNOPSIS

getpw(uid, buf)
char *buf;

DESCRIPTION

Getpw is obsoleted by getpwuid(3).

Getpw searches the password file for the (numerical) *uid*, and fills in *buf* with the corresponding line; it returns non-zero if *uid* could not be found. The line is null-terminated.

FILES

/etc/passwd

SEE ALSO

getpwent(3), passwd(5)

DIAGNOSTICS

Non-zero return on error.

NAME

lockf – record locking on files

SYNOPSIS

# include <unistd.h>

lockf (fildes, function, size) long size; int fildes, function;

DESCRIPTION

The *lockf* call will allow sections of a file to be locked (advisory write locks). (Mandatory or enforcement mode record locks are not currently available.) Locking calls from other processes which attempt to lock the locked file section will either return an error value or be put to sleep until the resource becomes unlocked. All the locks for a process are removed when the process terminates. [See *fcntl*(2) for more information about record locking.]

Fildes is an open file descriptor. The file descriptor must have O_WRONLY or O_RDWR permission in order to establish lock with this function call.

Function is a control value which specifies the action to be taken. The permissible values for *function* are defined in **<unistd.h>** as follows:

```
#define      F_ULOCK    0      /* Unlock a previously locked section */
#define      F_LOCK     1      /* Lock a section for exclusive use */
#define      F_TLOCK    2      /* Test and lock a section for exclusive use */
#define      F_TEST     3      /* Test section for other processes locks */
```

All other values of *function* are reserved for future extensions and will result in an error return if not implemented.

F_TEST is used to detect if a lock by another process is present on the specified section. F_LOCK and F_TLOCK both lock a section of a file if the section is available. F_UNLOCK removes locks from a section of the file.

Size is the number of contiguous bytes to be locked or unlocked. The resource to be locked starts at the current offset in the file and extends forward for a positive size and backward for a negative size. If *size* is zero, the section from the current offset through the largest file offset is locked (i.e., from the current offset through the present or any future end-of-file). An area need not be allocated to the file in order to be locked, as such locks may exist past the end-of-file.

The sections locked with F_LOCK or F_TLOCK may, in whole or in part, contain or be contained by a previously locked section for the same process. When this occurs, or if adjacent sections occur, the sections are combined into a single section. If the request requires that a new element be added to the table of active locks and this table is already full, an error is returned, and the new section is not locked.

F_LOCK and F_TLOCK requests differ only by the action taken if the resource is not available. F_LOCK will cause the calling process to sleep until the resource is available. F_TLOCK will cause the function to return a –1 and set *errno* to [EACCESS] error if the section is already locked by another process.

F_ULOCK requests may, in whole or in part, release one or more locked sections controlled by the process. When sections are not fully released, the remaining sections are still locked by the process. Releasing the center section of a locked section requires an additional element in the table of active locks. If this table is full, an [EDEADLK] error is returned and the requested section is not released.

A potential for deadlock occurs if a process controlling a locked resource is put to sleep by accessing another process's locked resource. Thus calls to *lock* or *fcntl* scan for a deadlock prior to sleeping on a locked resource. An error return is made if sleeping on the locked resource would cause a deadlock.

Sleeping on a resource is interrupted with any signal. The *alarm*(2) command may be used to provide a timeout facility in applications which require this facility.

ERRORS

The *lockf* utility will fail if one or more of the following are true:

[EBADF]
Fildes is not a valid open descriptor.

[EACCESS]
Cmd is F_TLOCK or F_TEST and the section is already locked by another process.

[EDEADLK]
Cmd is F_LOCK or F_TLOCK and a deadlock would occur. Also the *cmd* is either of the above or F_ULOCK and the number of entries in the lock table would exceed the number allocated on the system.

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned and *errno* is set to indicate the error.

CAVEATS

Unexpected results may occur in processes that do buffering in the user address space. The process may later read/write data which is/was locked. The standard I/O package is the most common source of unexpected buffering.

SEE ALSO

close(2), creat(2), fcntl(2), intro(2), open(2), read(2), write(2).

NAME

nice – set program priority

SYNOPSIS

nice(incr)

DESCRIPTION

This interface is obsoleted by setpriority(2).

The scheduling priority of the process is augmented by *incr*. Positive priorities get less service than normal. Priority 10 is recommended to users who wish to execute long-running programs without flak from the administration.

Negative increments are ignored except on behalf of the super-user. The priority is limited to the range −20 (most urgent) to 20 (least).

The priority of a process is passed to a child process by *fork*(2). For a privileged process to return to normal priority from an unknown state, *nice* should be called successively with arguments −40 (goes to priority −20 because of truncation), 20 (to get to 0), then 0 (to maintain compatibility with previous versions of this call).

SEE ALSO

nice(1), setpriority(2), fork(2), renice(8)

NAME
pause – stop until signal

SYNOPSIS
pause()

DESCRIPTION
Pause never returns normally. It is used to give up control while waiting for a signal from *kill*(2) or an interval timer, see *setitimer*(2). Upon termination of a signal handler started during a *pause,* the *pause* call will return.

RETURN VALUE
Always returns –1.

ERRORS
Pause always returns:

[EINTR]	The call was interrupted.

SEE ALSO
kill(2), select(2), sigpause(2)

NAME

rand, srand – random number generator

SYNOPSIS

srand(seed)
int seed;

rand()

DESCRIPTION

The newer random(3) should be used in new applications; rand remains for compatibilty.

Rand uses a multiplicative congruential random number generator with period 2^{32} to return successive pseudo-random numbers in the range from 0 to $2^{31}-1$.

The generator is reinitialized by calling *srand* with 1 as argument. It can be set to a random starting point by calling *srand* with whatever you like as argument.

SEE ALSO

random(3)

NAME

signal – simplified software signal facilities

SYNOPSIS

#include <signal.h>

(*signal(sig, func))()
void (*func)();

DESCRIPTION

Signal is a simplified interface to the more general *sigvec*(2) facility.

A signal is generated by some abnormal event, initiated by a user at a terminal (quit, interrupt, stop), by a program error (bus error, etc.), by request of another program (kill), or when a process is stopped because it wishes to access its control terminal while in the background (see *tty*(4)). Signals are optionally generated when a process resumes after being stopped, when the status of child processes changes, or when input is ready at the control terminal. Most signals cause termination of the receiving process if no action is taken; some signals instead cause the process receiving them to be stopped, or are simply discarded if the process has not requested otherwise. Except for the SIGKILL and SIGSTOP signals, the *signal* call allows signals either to be ignored or to cause an interrupt to a specified location. The following is a list of all signals with names as in the include file <*signal.h*>:

SIGHUP	1	hangup
SIGINT	2	interrupt
SIGQUIT	3*	quit
SIGILL	4*	illegal instruction
SIGTRAP	5*	trace trap
SIGIOT	6*	IOT instruction
SIGEMT	7*	EMT instruction
SIGFPE	8*	floating point exception
SIGKILL	9	kill (cannot be caught or ignored)
SIGBUS	10*	bus error
SIGSEGV	11*	segmentation violation
SIGSYS	12*	bad argument to system call
SIGPIPE	13	write on a pipe with no one to read it
SIGALRM	14	alarm clock
SIGTERM	15	software termination signal
SIGURG	16•	urgent condition present on socket
SIGSTOP	17†	stop (cannot be caught or ignored)
SIGTSTP	18†	stop signal generated from keyboard
SIGCONT	19•	continue after stop
SIGCHLD	20•	child status has changed
SIGTTIN	21†	background read attempted from control terminal
SIGTTOU	22†	background write attempted to control terminal
SIGIO	23•	i/o is possible on a descriptor (see *fcntl*(2))
SIGXCPU	24	cpu time limit exceeded (see *setrlimit*(2))
SIGXFSZ	25	file size limit exceeded (see *setrlimit*(2))
SIGVTALRM	26	virtual time alarm (see *setitimer*(2))
SIGPROF	27	profiling timer alarm (see *setitimer*(2))

The starred signals in the list above cause a core image if not caught or ignored.

If *func* is SIG_DFL, the default action for signal *sig* is reinstated; this default is termination (with a core image for starred signals) except for signals marked with • or †. Signals marked with • are discarded if the action is SIG_DFL; signals marked with † cause the process to stop. If *func* is SIG_IGN the signal is subsequently ignored and pending instances of the signal are discarded. Otherwise, when the signal occurs further occurences of the signal are automatically blocked and *func* is called.

A return from the function unblocks the handled signal and continues the process at the point it was interrupted. **Unlike previous signal facilities, the handler *func* remains installed after a signal has been delivered.**

If a caught signal occurs during certain system calls, causing the call to terminate prematurely, the call is automatically restarted. In particular this can occur during a *read* or *write*(2) on a slow device (such as a terminal; but not a file) and during a *wait*(2).

The value of *signal* is the previous (or initial) value of *func* for the particular signal.

After a *fork*(2) or *vfork*(2) the child inherits all signals. *Execve*(2) resets all caught signals to the default action; ignored signals remain ignored.

RETURN VALUE

The previous action is returned on a successful call. Otherwise, -1 is returned and *errno* is set to indicate the error.

ERRORS

Signal will fail and no action will take place if one of the following occur:

[EINVAL]	*Sig* is not a valid signal number.
[EINVAL]	An attempt is made to ignore or supply a handler for SIGKILL or SIGSTOP.
[EINVAL]	An attempt is made to ignore SIGCONT (by default SIGCONT is ignored).

SEE ALSO

kill(1), ptrace(2), kill(2), sigvec(2), sigblock(2), sigsetmask(2), sigpause(2), sigstack(2), setjmp(3), tty(4)

NAME

stty, gtty – set and get terminal state (defunct)

SYNOPSIS

```
#include <sgtty.h>

stty(fd, buf)
int fd;
struct sgttyb *buf;

gtty(fd, buf)
int fd;
struct sgttyb *buf;
```

DESCRIPTION

This interface is obsoleted by ioctl(2).

Stty sets the state of the terminal associated with *fd*. *Gtty* retrieves the state of the terminal associated with *fd*. To set the state of a terminal the call must have write permission.

The *stty* call is actually "ioctl(fd, TIOCSETP, buf)", while the *gtty* call is "ioctl(fd, TIOCGETP, buf)". See *ioctl*(2) and *tty*(4) for an explanation.

DIAGNOSTICS

If the call is successful 0 is returned, otherwise –1 is returned and the global variable *errno* contains the reason for the failure.

SEE ALSO

ioctl(2), tty(4)

NAME

time, ftime – get date and time

SYNOPSIS

long time(0)

long time(tloc)
long *tloc;

#include <sys/types.h>
#include <sys/timeb.h>
ftime(tp)
struct timeb *tp;

DESCRIPTION

These interfaces are obsoleted by gettimeofday(2).

Time returns the time since 00:00:00 GMT, Jan. 1, 1970, measured in seconds.

If *tloc* is nonnull, the return value is also stored in the place to which *tloc* points.

The *ftime* entry fills in a structure pointed to by its argument, as defined by <*sys/timeb.h*>:

```
/*      @(#)timeb.h 1.1 83/07/18 SMI; from UCB 4.2 81/02/19*/

/*
 * Structure returned by ftime system call
 */
struct timeb
{
        time_t    time;
        unsigned short millitm;
        short     timezone;
        short     dstflag;
};
```

The structure contains the time since the epoch in seconds, up to 1000 milliseconds of more-precise interval, the local time zone (measured in minutes of time westward from Greenwich), and a flag that, if nonzero, indicates that Daylight Saving time applies locally during the appropriate part of the year.

SEE ALSO

date(1), gettimeofday(2), settimeofday(2), ctime(3)

NAME

times – get process times

SYNOPSIS

```
#include <sys/types.h>
#include <sys/times.h>

times(buffer)
struct tms *buffer;
```

DESCRIPTION

This interface is obsoleted by getrusage(2).

Times returns time-accounting information for the current process and for the terminated child processes of the current process. All times are in 1/HZ seconds, where HZ is 60.

This is the structure returned by *times*:

```
/*      @(#)times.h 1.1 83/07/18 SMI; from UCB 4.2 81/02/19     */

/*
 * Structure returned by times()
 */
struct tms {
        time_t tms_utime;               /* user time */
        time_t tms_stime;               /* system time */
        time_t tms_cutime;              /* user time, children */
        time_t tms_cstime;              /* system time, children */
};
```

The children times are the sum of the children's process times and their children's times.

SEE ALSO

time(1), getrusage(2), wait3(2), time(3)

NAME

utime – set file times

SYNOPSIS

```
#include <sys/types.h>

utime(file, timep)
char *file;
time_t timep[2];
```

DESCRIPTION

This interface is obsoleted by utimes(2).

The *utime* call uses the 'accessed' and 'updated' times in that order from the *timep* vector to set the corresponding recorded times for *file*.

The caller must be the owner of the file or the super-user. The 'inode-changed' time of the file is set to the current time.

SEE ALSO

utimes(2), stat(2)

NAME

vlimit – control maximum system resource consumption

SYNOPSIS

#include <sys/vlimit.h>

vlimit(resource, value)

DESCRIPTION

This facility is superseded by getrlimit(2).

Limits the consumption by the current process and each process it creates to not individually exceed *value* on the specified *resource*. If *value* is specified as –1, then the current limit is returned and the limit is unchanged. The resources which are currently controllable are:

LIM_NORAISE	A pseudo-limit; if set non-zero then the limits may not be raised. Only the super-user may remove the *noraise* restriction.
LIM_CPU	the maximum number of cpu-seconds to be used by each process
LIM_FSIZE	the largest single file which can be created
LIM_DATA	the maximum growth of the data+stack region via *sbrk*(2) beyond the end of the program text
LIM_STACK	the maximum size of the automatically-extended stack region
LIM_CORE	the size of the largest core dump that will be created.
LIM_MAXRSS	a soft limit for the amount of physical memory (in bytes) to be given to the program. If memory is tight, the system will prefer to take memory from processes which are exceeding their declared LIM_MAXRSS.

Because this information is stored in the per-process information this system call must be executed directly by the shell if it is to affect all future processes created by the shell; *limit* is thus a built-in command to *csh*(1).

The system refuses to extend the data or stack space when the limits would be exceeded in the normal way; a *break* call fails if the data space limit is reached, or the process is killed when the stack limit is reached (since the stack cannot be extended, there is no way to send a signal!).

A file i/o operation which would create a file which is too large will cause a signal SIGXFSZ to be generated, this normally terminates the process, but may be caught. When the cpu time limit is exceeded, a signal SIGXCPU is sent to the offending process; to allow it time to process the signal it is given 5 seconds grace by raising the cpu time limit.

SEE ALSO

csh(1)

BUGS

If LIM_NORAISE is set, then no grace should be given when the cpu time limit is exceeded.

There should be *limit* and *unlimit* commands in *sh*(1) as well as in *csh.*

This call is peculiar to this version of UNIX. The options and specifications of this system call and even the call itself are subject to change. It may be extended or replaced by other facilities in future versions of the system.

NAME

vtimes – get information about resource utilization

SYNOPSIS

vtimes(par_vm, ch_vm)
struct vtimes *par_vm, *ch_vm;

DESCRIPTION

This facility is superseded by getrusage(2).

Vtimes returns accounting information for the current process and for the terminated child processes of the current process. Either *par_vm* or *ch_vm* or both may be 0, in which case only the information for the pointers which are non-zero is returned.

After the call, each buffer contains information as defined by the contents of the include file */usr/include/sys/vtimes.h:*

```
struct vtimes {
        int      vm_utime;          /* user time (*HZ) */
        int      vm_stime;          /* system time (*HZ) */
        /* divide next two by utime+stime to get averages */
        unsigned vm_idsrss;         /* integral of d+s rss */
        unsigned vm_ixrss;          /* integral of text rss */
        int      vm_maxrss;         /* maximum rss */
        int      vm_majflt;         /* major page faults */
        int      vm_minflt;         /* minor page faults */
        int      vm_nswap;          /* number of swaps */
        int      vm_inblk;          /* block reads */
        int      vm_oublk;          /* block writes */
};
```

The *vm_utime* and *vm_stime* fields give the user and system time respectively in 60ths of a second (or 50ths if that is the frequency of wall current in your locality.) The *vm_idrss* and *vm_ixrss* measure memory usage. They are computed by integrating the number of memory pages in use each over cpu time. They are reported as though computed discretely, adding the current memory usage (in 512 byte pages) each time the clock ticks. If a process used 5 core pages over 1 cpu-second for its data and stack, then *vm_idsrss* would have the value 5*60, where *vm_utime+vm_stime* would be the 60. *Vm_idsrss* integrates data and stack segment usage, while *vm_ixrss* integrates text segment usage. *Vm_maxrss* reports the maximum instantaneous sum of the text+data+stack core-resident page count.

The *vm_majflt* field gives the number of page faults which resulted in disk activity; the *vm_minflt* field gives the number of page faults incurred in simulation of reference bits; *vm_nswap* is the number of swaps which occurred. The number of file system input/output events are reported in *vm_inblk* and *vm_oublk* These numbers account only for real i/o; data supplied by the caching mechanism is charged only to the first process to read or write the data.

SEE ALSO

time(2), wait3(2)

BUGS

This call is peculiar to this version of UNIX. The options and specifications of this system call are subject to change. It may be extended to include additional information in future versions of the system.

SECTION 3F

ICON/UXB OPERATING SYSTEM FORTRAN LIBRARY

ICON®

NAME

intro – introduction to FORTRAN library functions

DESCRIPTION

This section describes those functions that are in the FORTRAN run time library. The functions listed here provide an interface from *f77* programs to the system in the same manner as the C library does for C programs. They are automatically loaded as needed by the Fortran compiler *f77*(1).

Most of these functions are in libU77.a. Some are in libF77.a or libI77.a. A few intrinsic functions are described for the sake of completeness.

For efficiency, the SCCS ID strings are not normally included in the *a.out* file. To include them, simply declare

```
external f77lid
```

in any *f77* module.

LIST OF FUNCTIONS

Name	*Appears on Page*	*Description*
abort	abort.3f	terminate abruptly with memory image
access	access.3f	determine accessability of a file
alarm	alarm.3f	execute a subroutine after a specified time
bessel	bessel.3f	of two kinds for integer orders
bit	bit.3f	and, or, xor, not, rshift, lshift bitwise functions
chdir	chdir.3f	change default directory
chmod	chmod.3f	change mode of a file
ctime	time.3f	return system time
dffrac	flmin.3f	return extreme values
dflmax	flmin.3f	return extreme values
dflmin	flmin.3f	return extreme values
drand	rand.3f	return random values
dtime	etime.3f	return elapsed execution time
etime	etime.3f	return elapsed execution time
exit	exit.3f	terminate process with status
fdate	fdate.3f	return date and time in an ASCII string
ffrac	flmin.3f	return extreme values
fgetc	getc.3f	get a character from a logical unit
flmax	flmin.3f	return extreme values
flmin	flmin.3f	return extreme values
flush	flush.3f	flush output to a logical unit
fork	fork.3f	create a copy of this process
fpecnt	trpfpe.3f	trap and repair floating point faults
fputc	putc.3f	write a character to a fortran logical unit
fseek	fseek.3f	reposition a file on a logical unit
fstat	stat.3f	get file status
ftell	fseek.3f	reposition a file on a logical unit
gerror	perror.3f	get system error messages
getarg	getarg.3f	return command line arguments
getc	getc.3f	get a character from a logical unit

getcwd	getcwd.3f	get pathname of current working directory
getenv	getenv.3f	get value of environment variables
getgid	getuid.3f	get user or group ID of the caller
getlog	getlog.3f	get user's login name
getpid	getpid.3f	get process id
getuid	getuid.3f	get user or group ID of the caller
gmtime	time.3f	return system time
hostnm	hostnm.3f	get name of current host
iargc	getarg.3f	return command line arguments
idate	idate.3f	return date or time in numerical form
ierrno	perror.3f	get system error messages
index	index.3f	tell about character objects
inmax	flmin.3f	return extreme values
intro	intro.3f	introduction to FORTRAN library functions
ioinit	ioinit.3f	change f77 I/O initialization
irand	rand.3f	return random values
isatty	ttynam.3f	find name of a terminal port
itime	idate.3f	return date or time in numerical form
kill	kill.3f	send a signal to a process
len	index.3f	tell about character objects
link	link.3f	make a link to an existing file
lnblnk	index.3f	tell about character objects
loc	loc.3f	return the address of an object
long	long.3f	integer object conversion
lstat	stat.3f	get file status
ltime	time.3f	return system time
perror	perror.3f	get system error messages
putc	putc.3f	write a character to a fortran logical unit
qsort	qsort.3f	quick sort
rand	rand.3f	return random values
rename	rename.3f	rename a file
rindex	index.3f	tell about character objects
short	long.3f	integer object conversion
signal	signal.3f	change the action for a signal
sleep	sleep.3f	suspend execution for an interval
stat	stat.3f	get file status
system	system.3f	execute a UNIX command
tclose	topen.3f	f77 tape I/O
time	time.3f	return system time
topen	topen.3f	f77 tape I/O
traper	traper.3f	trap arithmetic errors
trapov	trapov.3f	trap and repair floating point overflow
tread	topen.3f	f77 tape I/O
trewin	topen.3f	f77 tape I/O
trpfpe	trpfpe.3f	trap and repair floating point faults
tskipf	topen.3f	f77 tape I/O
tstate	topen.3f	f77 tape I/O
ttynam	ttynam.3f	find name of a terminal port
twrite	topen.3f	f77 tape I/O
unlink	unlink.3f	remove a directory entry
wait	wait.3f	wait for a process to terminate

NAME
abort – terminate abruptly with memory image

SYNOPSIS
subroutine abort (string)
character*(*) string

DESCRIPTION
Abort cleans up the I/O buffers and then aborts producing a *core* file in the current directory. If *string* is given, it is written to logical unit 0 preceeded by "abort:".

FILES
/usr/lib/libF77.a

SEE ALSO
abort(3)

NAME

access – determine accessability of a file

SYNOPSIS

integer function access (name, mode)
character*(*) name, mode

DESCRIPTION

Access checks the given file, *name,* for accessability with respect to the caller according to *mode. Mode* may include in any order and in any combination one or more of:

r	test for read permission
w	test for write permission
x	test for execute permission
(blank)	test for existence

An error code is returned if either argument is illegal, or if the file can not be accessed in all of the specified modes. 0 is returned if the specified access would be successful.

FILES

/usr/lib/libU77.a

SEE ALSO

access(2), perror(3F)

BUGS

Pathnames can be no longer than MAXPATHLEN as defined in <*sys/param.h*>.

NAME

alarm – execute a subroutine after a specified time

SYNOPSIS

integer function alarm (time, proc)
integer time
external proc

DESCRIPTION

This routine arranges for subroutine *proc* to be called after *time* seconds. If *time* is "0", the alarm is turned off and no routine will be called. The returned value will be the time remaining on the last alarm.

FILES

/usr/lib/libU77.a

SEE ALSO

alarm(3C), sleep(3F), signal(3F)

BUGS

Alarm and *sleep* interact. If *sleep* is called after *alarm*, the *alarm* process will never be called. SIGALRM will occur at the lesser of the remaining *alarm* time or the *sleep* time.

NAME

bessel functions – of two kinds for integer orders

SYNOPSIS

function besj0 (x)

function besj1 (x)

function besjn (n, x)

function besy0 (x)

function besy1 (x)

function besyn (n, x)

double precision function dbesj0 (x)
double precision x

double precision function dbesj1 (x)
double precision x

double precision function dbesjn (n, x)
double precision x

double precision function dbesy0 (x)
double precision x

double precision function dbesy1 (x)
double precision x

double precision function dbesyn (n, x)
double precision x

DESCRIPTION

These functions calculate Bessel functions of the first and second kinds for real arguments and integer orders.

DIAGNOSTICS

Negative arguments cause *besy0, besy1,* and *besyn* to return a huge negative value. The system error code will be set to EDOM (33).

FILES

/usr/lib/libF77.a

SEE ALSO

j0(3M), perror(3F)

NAME

bit – and, or, xor, not, rshift, lshift bitwise functions

SYNOPSIS

(intrinsic) function and (word1, word2)

(intrinsic) function or (word1, word2)

(intrinsic) function xor (word1, word2)

(intrinsic) function not (word)

(intrinsic) function rshift (word, nbits)

(intrinsic) function lshift (word, nbits)

DESCRIPTION

These bitwise functions are built into the compiler and return the data type of their argument(s). It is recommended that their arguments be **integer** values; inappropriate manipulation of **real** objects may cause unexpected results.

The bitwise combinatorial functions return the bitwise "and" (**and**), "or" (**or**), or "exclusive or" (**xor**) of two operands. **Not** returns the bitwise complement of its operand.

Lshift, or *rshift* with a negative *nbits*, is a logical left shift with no end around carry. *Rshift*, or *lshift* with a negative *nbits*, is an arithmatic right shift with sign extension. No test is made for a reasonable value of *nbits*.

FILES

These functions are generated in-line by the f77 compiler.

NAME

chdir – change default directory

SYNOPSIS

integer function chdir (dirname)
character*(*) dirname

DESCRIPTION

The default directory for creating and locating files will be changed to *dirname*. Zero is returned if successful; an error code otherwise.

FILES

/usr/lib/libU77.a

SEE ALSO

chdir(2), cd(1), perror(3F)

BUGS

Pathnames can be no longer than MAXPATHLEN as defined in <*sys/param.h*>.

Use of this function may cause **inquire** by unit to fail.

NAME

chmod – change mode of a file

SYNOPSIS

integer function chmod (name, mode)
character*(*) name, mode

DESCRIPTION

This function changes the filesystem *mode* of file *name*. *Mode* can be any specification recognized by *chmod*(1). *Name* must be a single pathname.

The normal returned value is 0. Any other value will be a system error number.

FILES

/usr/lib/libU77.a	
/bin/chmod	exec'ed to change the mode.

SEE ALSO

chmod(1)

BUGS

Pathnames can be no longer than MAXPATHLEN as defined in <*sys/param.h*>.

NAME

etime, dtime – return elapsed execution time

SYNOPSIS

function etime (tarray)
real tarray(2)

function dtime (tarray)
real tarray(2)

DESCRIPTION

These two routines return elapsed runtime in seconds for the calling process. *Dtime* returns the elapsed time since the last call to *dtime,* or the start of execution on the first call.

The argument array returns user time in the first element and system time in the second element. The function value is the sum of user and system time.

The resolution of all timing is 1/HZ sec. where HZ is currently 60.

FILES

/usr/lib/libU77.a

SEE ALSO

times(2)

NAME

exit – terminate process with status

SYNOPSIS

subroutine exit (status)
integer status

DESCRIPTION

Exit flushes and closes all the process's files, and notifies the parent process if it is executing a *wait*. The low-order 8 bits of *status* are available to the parent process. (Therefore *status* should be in the range 0 – 255)

This call will never return.

The C function *exit* may cause cleanup actions before the final 'sys exit'.

FILES

/usr/lib/libF77.a

SEE ALSO

exit(2), fork(2), fork(3F), wait(2), wait(3F)

NAME

fdate – return date and time in an ASCII string

SYNOPSIS

subroutine fdate (string)
character*(*) string

character*(*) function fdate()

DESCRIPTION

Fdate returns the current date and time as a 24 character string in the format described under *ctime*(3). Neither 'newline' nor NULL will be included.

Fdate can be called either as a function or as a subroutine. If called as a function, the calling routine must define its type and length. For example:

```
character*24   fdate
external       fdate

write(*,*) fdate()
```

FILES

/usr/lib/libU77.a

SEE ALSO

ctime(3), time(3F), itime(3F), idate(3F), ltime(3F)

NAME

flmin, flmax, ffrac, dflmin, dflmax, dffrac, inmax – return extreme values

SYNOPSIS

function flmin()

function flmax()

function ffrac()

double precision function dflmin()

double precision function dflmax()

double precision function dffrac()

function inmax()

DESCRIPTION

Functions *flmin* and *flmax* return the minimum and maximum positive floating point values respectively. Functions *dflmin* and *dflmax* return the minimum and maximum positive double precision floating point values. Function *inmax* returns the maximum positive integer value.

The functions *ffrac* and *dffrac* return the fractional accuracy of single and double precision floating point numbers respectively. These are the smallest numbers that can be added to 1.0 without being lost.

These functions can be used by programs that must scale algorithms to the numerical range of the processor.

FILES

/usr/lib/libF77.a

NAME
flush – flush output to a logical unit

SYNOPSIS
subroutine flush (lunit)

DESCRIPTION
Flush causes the contents of the buffer for logical unit *lunit* to be flushed to the associated file. This is most useful for logical units 0 and 6 when they are both associated with the control terminal.

FILES
/usr/lib/libI77.a

SEE ALSO
fclose(3S)

NAME
fork – create a copy of this process

SYNOPSIS
integer function fork()

DESCRIPTION
Fork creates a copy of the calling process. The only distinction between the 2 processes is that the value returned to one of them (referred to as the 'parent' process) will be the process id if the copy. The copy is usually referred to as the 'child' process. The value returned to the 'child' process will be zero.

All logical units open for writing are flushed before the fork to avoid duplication of the contents of I/O buffers in the external file(s).

If the returned value is negative, it indicates an error and will be the negation of the system error code. See perror(3F).

A corresponding *exec* routine has not been provided because there is no satisfactory way to retain open logical units across the exec. However, the usual function of *fork/exec* can be performed using *system*(3F).

FILES
/usr/lib/libU77.a

SEE ALSO
fork(2), wait(3F), kill(3F), system(3F), perror(3F)

NAME

fseek, ftell – reposition a file on a logical unit

SYNOPSIS

integer function fseek (lunit, offset, from)
integer offset, from

integer function ftell (lunit)

DESCRIPTION

lunit must refer to an open logical unit. *offset* is an offset in bytes relative to the position specified by *from*. Valid values for *from* are:

0 meaning 'beginning of the file'
1 meaning 'the current position'
2 meaning 'the end of the file'

The value returned by *fseek* will be 0 if successful, a system error code otherwise. (See perror(3F))

Ftell returns the current position of the file associated with the specified logical unit. The value is an offset, in bytes, from the beginning of the file. If the value returned is negative, it indicates an error and will be the negation of the system error code. (See perror(3F))

FILES

/usr/lib/libU77.a

SEE ALSO

fseek(3S), perror(3F)

NAME

getarg, iargc – return command line arguments

SYNOPSIS

subroutine getarg (k, arg)
character*(*) arg

function iargc ()

DESCRIPTION

A call to *getarg* will return the **k***th* command line argument in character string *arg*. The 0*th* argument is the command name.

Iargc returns the index of the last command line argument.

FILES

/usr/lib/libU77.a

SEE ALSO

getenv(3F), execve(2)

NAME

getc, fgetc – get a character from a logical unit

SYNOPSIS

integer function getc (char)
character char

integer function fgetc (lunit, char)
character char

DESCRIPTION

These routines return the next character from a file associated with a fortran logical unit, bypassing normal fortran I/O. *Getc* reads from logical unit 5, normally connected to the control terminal input.

The value of each function is a system status code. Zero indicates no error occured on the read; −1 indicates end of file was detected. A positive value will be either a UNIX system error code or an f77 I/O error code. See perror(3F).

FILES

/usr/lib/libU77.a

SEE ALSO

getc(3S), intro(2), perror(3F)

NAME

getcwd – get pathname of current working directory

SYNOPSIS

integer function getcwd (dirname)
character*(*) dirname

DESCRIPTION

The pathname of the default directory for creating and locating files will be returned in *dirname*. The value of the function will be zero if successful; an error code otherwise.

FILES

/usr/lib/libU77.a

SEE ALSO

chdir(3F), perror(3F)

BUGS

Pathnames can be no longer than MAXPATHLEN as defined in <*sys/param.h*>.

NAME

getenv – get value of environment variables

SYNOPSIS

subroutine getenv (ename, evalue)
character*(*) ename, evalue

DESCRIPTION

Getenv searches the environment list (see *environ*(7)) for a string of the form *ename*=*value* and returns *value* in *evalue* if such a string is present, otherwise fills *evalue* with blanks.

FILES

/usr/lib/libU77.a

SEE ALSO

environ(7), execve(2)

NAME
getpid – get process id

SYNOPSIS
integer function getpid()

DESCRIPTION
Getpid returns the process ID number of the current process.

FILES
/usr/lib/libU77.a

SEE ALSO
getpid(2)

NAME

getlog – get user's login name

SYNOPSIS

subroutine getlog (name)
character*(*) name

character*(*) function getlog()

DESCRIPTION

Getlog will return the user's login name or all blanks if the process is running detached from a terminal.

FILES

/usr/lib/libU77.a

SEE ALSO

getlogin(3)

NAME

getuid, getgid – get user or group ID of the caller

SYNOPSIS

integer function getuid()

integer function getgid()

DESCRIPTION

These functions return the real user or group ID of the user of the process.

FILES

/usr/lib/libU77.a

SEE ALSO

getuid(2)

NAME
hostnm – get name of current host

SYNOPSIS
integer function hostnm (name)
character*(*) name

DESCRIPTION
This function puts the name of the current host into character string *name*. The return value should be 0; any other value indicates an error.

FILES
/usr/lib/libU77.a

SEE ALSO
gethostname(2)

NAME
idate, itime – return date or time in numerical form

SYNOPSIS
subroutine idate (iarray)
integer iarray(3)

subroutine itime (iarray)
integer iarray(3)

DESCRIPTION
Idate returns the current date in *iarray*. The order is: day, mon, year. Month will be in the range 1-12. Year will be $\geq$ 1969.

Itime returns the current time in *iarray*. The order is: hour, minute, second.

FILES
/usr/lib/libU77.a

SEE ALSO
ctime(3F), fdate(3F)

NAME

index, rindex, lnblnk, len – tell about character objects

SYNOPSIS

(intrinsic) function index (string, substr)
character*(*) string, substr

integer function rindex (string, substr)
character*(*) string, substr

function lnblnk (string)
character*(*) string

(intrinsic) function len (string)
character*(*) string

DESCRIPTION

Index (rindex) returns the index of the first (last) occurrence of the substring *substr* in *string*, or zero if it does not occur. *Index* is an f77 intrinsic function; *rindex* is a library routine.

Lnblnk returns the index of the last non-blank character in *string*. This is useful since all f77 character objects are fixed length, blank padded. Intrinsic function *len* returns the size of the character object argument.

FILES

/usr/lib/libF77.a

NAME

ioinit – change f77 I/O initialization

SYNOPSIS

logical function ioinit (cctl, bzro, apnd, prefix, vrbose)
logical cctl, bzro, apnd, vrbose
character*(*) prefix

DESCRIPTION

This routine will initialize several global parameters in the f77 I/O system, and attach externally defined files to logical units at run time. The effect of the flag arguments applies to logical units opened after *ioinit* is called. The exception is the preassigned units, 5 and 6, to which *cctl* and *bzro* will apply at any time. *Ioinit* is written in Fortran-77.

By default, carriage control is not recognized on any logical unit. If *cctl* is **.true.** then carriage control will be recognized on formatted output to all logical units except unit 0, the diagnostic channel. Otherwise the default will be restored.

By default, trailing and embedded blanks in input data fields are ignored. If *bzro* is **.true.** then such blanks will be treated as zero's. Otherwise the default will be restored.

By default, all files opened for sequential access are positioned at their beginning. It is sometimes necessary or convenient to open at the END-OF-FILE so that a write will append to the existing data. If *apnd* is **.true.** then files opened subsequently on any logical unit will be positioned at their end upon opening. A value of **.false.** will restore the default behavior.

Many systems provide an automatic association of global names with fortran logical units when a program is run. There is no such automatic association in f77. However, if the argument *prefix* is a non-blank string, then names of the form **prefix**NN will be sought in the program environment. The value associated with each such name found will be used to open logical unit NN for formatted sequential access. For example, if f77 program *myprogram* included the call

```
call ioinit (.true., .false., .false., 'FORT', .false.)
```

then when the following sequence

```
% setenv FORT01 mydata
% setenv FORT12 myresults
% myprogram
```

would result in logical unit 1 opened to file *mydata* and logical unit 12 opened to file *myresults*. Both files would be positioned at their beginning. Any formatted output would have column 1 removed and interpreted as carriage control. Embedded and trailing blanks would be ignored on input.

If the argument *vrbose* is **.true.** then *ioinit* will report on its activity.

The effect of

```
call ioinit (.true., .true., .false., '', .false.)
```

can be achieved without the actual call by including "–lI66" on the *f77* command line. This gives carriage control on all logical units except 0, causes files to be opened at their beginning, and causes blanks to be interpreted as zero's.

The internal flags are stored in a labeled common block with the following definition:

```
integer*2 ieof, ictl, ibzr
common /ioiflg/ ieof, ictl, ibzr
```

FILES

/usr/lib/libI77.a	f77 I/O library
/usr/lib/libI66.a	sets older fortran I/O modes

SEE ALSO

getarg(3F), getenv(3F), "Introduction to the f77 I/O Library"

BUGS

Prefix can be no longer than 30 characters. A pathname associated with an environment name can be no longer than 255 characters.

The "+" carriage control does not work.

NAME

kill – send a signal to a process

SYNOPSIS

function kill (pid, signum)
integer pid, signum

DESCRIPTION

Pid must be the process id of one of the user's processes. *Signum* must be a valid signal number (see sigvec(2)). The returned value will be 0 if successful; an error code otherwise.

FILES

/usr/lib/libU77.a

SEE ALSO

kill(2), sigvec(2), signal(3F), fork(3F), perror(3F)

NAME

link – make a link to an existing file

SYNOPSIS

function link (name1, name2)
character*(*) name1, name2

integer function symlnk (name1, name2)
character*(*) name1, name2

DESCRIPTION

Name1 must be the pathname of an existing file. *Name2* is a pathname to be linked to file *name1*. *Name2* must not already exist. The returned value will be 0 if successful; a system error code otherwise.

Symlnk creates a symbolic link to *name1*.

FILES

/usr/lib/libU77.a

SEE ALSO

link(2), symlink(2), perror(3F), unlink(3F)

BUGS

Pathnames can be no longer than MAXPATHLEN as defined in <*sys/param.h*>.

NAME
loc – return the address of an object

SYNOPSIS
function loc (arg)

DESCRIPTION
The returned value will be the address of *arg*.

FILES
/usr/lib/libU77.a

NAME

long, short – integer object conversion

SYNOPSIS

integer*4 function long (int2)
integer*2 int2

integer*2 function short (int4)
integer*4 int4

DESCRIPTION

These functions provide conversion between short and long integer objects. *Long* is useful when constants are used in calls to library routines and the code is to be compiled with "-i2". *Short* is useful in similar context when an otherwise long object must be passed as a short integer.

FILES

/usr/lib/libF77.a

NAME

perror, gerror, ierrno – get system error messages

SYNOPSIS

subroutine perror (string)
character*(*) string

subroutine gerror (string)
character*(*) string

character*(*) function gerror()

function ierrno()

DESCRIPTION

Perror will write a message to fortran logical unit 0 appropriate to the last detected system error. *String* will be written preceding the standard error message.

Gerror returns the system error message in character variable *string*. *Gerror* may be called either as a subroutine or as a function.

Ierrno will return the error number of the last detected system error. This number is updated only when an error actually occurs. Most routines and I/O statements that might generate such errors return an error code after the call; that value is a more reliable indicator of what caused the error condition.

FILES

/usr/lib/libU77.a

SEE ALSO

intro(2), perror(3)
D. L. Wasley, *Introduction to the f77 I/O Library*

BUGS

String in the call to *perror* can be no longer than 127 characters.

The length of the string returned by *gerror* is determined by the calling program.

NOTES

UNIX system error codes are described in *intro*(2). The f77 I/O error codes and their meanings are:

100	"error in format"
101	"illegal unit number"
102	"formatted io not allowed"
103	"unformatted io not allowed"
104	"direct io not allowed"
105	"sequential io not allowed"
106	"can't backspace file"
107	"off beginning of record"
108	"can't stat file"
109	"no * after repeat count"

110	"off end of record"
111	"truncation failed"
112	"incomprehensible list input"
113	"out of free space"
114	"unit not connected"
115	"read unexpected character"
116	"blank logical input field"
117	"'new' file exists"
118	"can't find 'old' file"
119	"unknown system error"
120	"requires seek ability"
121	"illegal argument"
122	"negative repeat count"
123	"illegal operation for unit"

NAME
putc, fputc – write a character to a fortran logical unit

SYNOPSIS
integer function putc (char)
character char

integer function fputc (lunit, char)
character char

DESCRIPTION
These functions write a character to the file associated with a fortran logical unit bypassing normal fortran I/O. *Putc* writes to logical unit 6, normally connected to the control terminal output.

The value of each function will be zero unless some error occurred; a system error code otherwise. See perror(3F).

FILES
/usr/lib/libU77.a

SEE ALSO
putc(3S), intro(2), perror(3F)

NAME

qsort – quick sort

SYNOPSIS

subroutine qsort (array, len, isize, compar)
external compar
integer*2 compar

DESCRIPTION

One dimensional *array* contains the elements to be sorted. *len* is the number of elements in the array. *isize* is the size of an element, typically -

4 for **integer** and **real**
8 for **double precision** or **complex**
16 for **double complex**
(length of character object) for **character** arrays

Compar is the name of a user supplied integer*2 function that will determine the sorting order. This function will be called with 2 arguments that will be elements of *array.* The function must return -

negative if arg 1 is considered to precede arg 2
zero if arg 1 is equivalent to arg 2
positive if arg 1 is considered to follow arg 2

On return, the elements of *array* will be sorted.

FILES

/usr/lib/libU77.a

SEE ALSO

qsort(3)

NAME

rand, drand, irand – return random values

SYNOPSIS

function irand (iflag)

function rand (iflag)

double precision function drand (iflag)

DESCRIPTION

These functions use *rand*(3C) to generate sequences of random numbers. If *iflag* is '1', the generator is restarted and the first random value is returned. If *iflag* is otherwise non-zero, it is used as a new seed for the random number generator, and the first new random value is returned.

Irand returns positive integers in the range 0 through 2147483647. *Rand* and *drand* return values in the range 0. through 1.0 .

FILES

/usr/lib/libF77.a

SEE ALSO

rand(3C)

NAME

rename – rename a file

SYNOPSIS

integer function rename (from, to)
character*(*) from, to

DESCRIPTION

From must be the pathname of an existing file. *To* will become the new pathname for the file. If *to* exists, then both *from* and *to* must be the same type of file, and must reside on the same filesystem. If *to* exists, it will be removed first.

The returned value will be 0 if successful; a system error code otherwise.

FILES

/usr/lib/libU77.a

SEE ALSO

rename(2), perror(3F)

BUGS

Pathnames can be no longer than MAXPATHLEN as defined in <*sys/param.h*>.

NAME

signal – change the action for a signal

SYNOPSIS

integer function signal(signum, proc, flag)
integer signum, flag
external proc

DESCRIPTION

When a process incurs a signal (see *signal*(3C)) the default action is usually to clean up and abort. The user may choose to write an alternative signal handling routine. A call to *signal* is the way this alternate action is specified to the system.

Signum is the signal number (see *signal*(3C)). If *flag* is negative, then *proc* must be the name of the user signal handling routine. If *flag* is zero or positive, then *proc* is ignored and the value of *flag* is passed to the system as the signal action definition. In particular, this is how previously saved signal actions can be restored. Two possible values for *flag* have specific meanings: 0 means "use the default action" (See NOTES below), 1 means "ignore this signal".

A positive returned value is the previous action definition. A value greater than 1 is the address of a routine that was to have been called on occurrence of the given signal. The returned value can be used in subsequent calls to *signal* in order to restore a previous action definition. A negative returned value is the negation of a system error code. (See *perror*(3F))

FILES

/usr/lib/libU77.a

SEE ALSO

signal(3C), kill(3F), kill(1)

NOTES

f77 arranges to trap certain signals when a process is started. The only way to restore the default **f77** action is to save the returned value from the first call to *signal*.

If the user signal handler is called, it will be passed the signal number as an integer argument.

NAME
sleep – suspend execution for an interval

SYNOPSIS
subroutine sleep (itime)

DESCRIPTION
Sleep causes the calling process to be suspended for *itime* seconds. The actual time can be up to 1 second less than *itime* due to granularity in system timekeeping.

FILES
/usr/lib/libU77.a

SEE ALSO
sleep(3)

NAME

stat, lstat, fstat – get file status

SYNOPSIS

integer function stat (name, statb)
character*(*) name
integer statb(12)

integer function lstat (name, statb)
character*(*) name
integer statb(12)

integer function fstat (lunit, statb)
integer statb(12)

DESCRIPTION

These routines return detailed information about a file. *Stat* and *lstat* return information about file *name*; *fstat* returns information about the file associated with fortran logical unit *lunit*. The order and meaning of the information returned in array *statb* is as described for the structure *stat* under *stat*(2). The "spare" values are not included.

The value of either function will be zero if successful; an error code otherwise.

FILES

/usr/lib/libU77.a

SEE ALSO

stat(2), access(3F), perror(3F), time(3F)

BUGS

Pathnames can be no longer than MAXPATHLEN as defined in <*sys/param.h*>.

NAME

system – execute a UNIX command

SYNOPSIS

integer function system (string)
character*(*) string

DESCRIPTION

System causes *string* to be given to your shell as input as if the string had been typed as a command. If environment variable **SHELL** is found, its value will be used as the command interpreter (shell); otherwise *sh*(1) is used.

The current process waits until the command terminates. The returned value will be the exit status of the shell. See *wait*(2) for an explanation of this value.

FILES

/usr/lib/libU77.a

SEE ALSO

exec(2), wait(2), system(3)

BUGS

String can not be longer than NCARGS–50 characters, as defined in <*sys/param.h*>.

NAME

time, ctime, ltime, gmtime – return system time

SYNOPSIS

integer function time()

character*(*) function ctime (stime)
integer stime

subroutine ltime (stime, tarray)
integer stime, tarray(9)

subroutine gmtime (stime, tarray)
integer stime, tarray(9)

DESCRIPTION

Time returns the time since 00:00:00 GMT, Jan. 1, 1970, measured in seconds. This is the value of the UNIX system clock.

Ctime converts a system time to a 24 character ASCII string. The format is described under *ctime*(3). No 'newline' or NULL will be included.

Ltime and *gmtime* disect a UNIX time into month, day, etc., either for the local time zone or as GMT. The order and meaning of each element returned in *tarray* is described under *ctime*(3).

FILES

/usr/lib/libU77.a

SEE ALSO

ctime(3), itime(3F), idate(3F), fdate(3F)

NAME

topen, tclose, tread, twrite, trewin, tskipf, tstate – f77 tape I/O

SYNOPSIS

integer function topen (tlu, devnam, label)
integer tlu
character*(*) devnam
logical label

integer function tclose (tlu)
integer tlu

integer function tread (tlu, buffer)
integer tlu
character*(*) buffer

integer function twrite (tlu, buffer)
integer tlu
character*(*) buffer

integer function trewin (tlu)
integer tlu

integer function tskipf (tlu, nfiles, nrecs)
integer tlu, nfiles, nrecs

integer function tstate (tlu, fileno, recno, errf, eoff, eotf, tcsr)
integer tlu, fileno, recno, tcsr
logical errf, eoff, eotf

DESCRIPTION

These functions provide a simple interface between f77 and magnetic tape devices. A "tape logical unit", *tlu*, is "topen"ed in much the same way as a normal f77 logical unit is "open"ed. All other operations are performed via the *tlu*. The *tlu* has no relationship at all to any normal f77 logical unit.

Topen associates a device name with a *tlu*. *Tlu* must be in the range 0 to 3. The logical argument *label* should indicate whether the tape includes a tape label. This is used by *trewin* below. *Topen* does not move the tape. The normal returned value is 0. If the value of the function is negative, an error has occured. See *perror*(3F) for details.

Tclose closes the tape device channel and removes its association with *tlu*. The normal returned value is 0. A negative value indicates an error.

Tread reads the next physical record from tape to *buffer*. *Buffer* **must** be of type **character**. The size of *buffer* should be large enough to hold the largest physical record to be read. The actual number of bytes read will be returned as the value of the function. If the value is 0, the end-of-file has been detected. A negative value indicates an error.

Twrite writes a physical record to tape from *buffer*. The physical record length will be the size of *buffer*. *Buffer* **must** be of type **character**. The number of bytes written will be returned. A value of 0 or negative indicates an error.

Trewin rewinds the tape associated with *tlu* to the beginning of the first data file. If the tape is a labelled tape (see *topen* above) then the label is skipped over after rewinding. The normal returned value is 0. A negative value indicates an error.

Tskipf allows the user to skip over files and/or records. First, *nfiles* end-of-file marks are skipped. If the current file is at EOF, this counts as 1 file to skip. (Note: This is the way to reset the EOF status for a *tlu*.) Next, *nrecs* physical records are skipped over. The normal returned value is 0. A negative value indicates an error.

Finally, *tstate* allows the user to determine the logical state of the tape I/O channel and to see the tape drive control status register. The values of *fileno* and *recno* will be returned and indicate the current file and record number. The logical values *errf*, *eoff*, and *eotf* indicate an error has occurred, the current file is at EOF, or the tape has reached logical end-of-tape. End-of-tape (EOT) is indicated by an empty file, often referred to as a double EOF mark. It is not allowed to read past EOT although it is allowed to write. The value of *tcsr* will reflect the tape drive control status register. See *ht*(4) for details.

FILES

/usr/lib/libU77.a

SEE ALSO

ht(4), perror(3F), rewind(1)

NAME

traper – trap arithmetic errors

SYNOPSIS

integer function traper (mask)

DESCRIPTION

NOTE: This routine applies only to the VAX. It is ignored on ICON systems.

NAME

trapov – trap and repair floating point overflow

SYNOPSIS

subroutine trapov (numesg, rtnval)
double precision rtnval

DESCRIPTION

NOTE: This routine applies only to the VAX. It is ignored on ICON systems.

NAME

trpfpe, fpecnt – trap and repair floating point faults

SYNOPSIS

subroutine trpfpe (numesg, rtnval)
double precision rtnval

integer function fpecnt ()

common /fpeflt/ fperr
logical fperr

DESCRIPTION

NOTE: This routine applies only to the VAX. It is ignored on ICON systems.

NAME

ttynam, isatty – find name of a terminal port

SYNOPSIS

character*(*) function ttynam (lunit)

logical function isatty (lunit)

DESCRIPTION

Ttynam returns a blank padded path name of the terminal device associated with logical unit *lunit*.

Isatty returns **.true.** if *lunit* is associated with a terminal device, **.false.** otherwise.

FILES

/dev/*
/usr/lib/libU77.a

DIAGNOSTICS

Ttynam returns an empty string (all blanks) if *lunit* is not associated with a terminal device in directory '/dev'.

NAME

unlink – remove a directory entry

SYNOPSIS

integer function unlink (name)
character*(*) name

DESCRIPTION

Unlink causes the directory entry specified by pathname *name* to be removed. If this was the last link to the file, the contents of the file are lost. The returned value will be zero if successful; a system error code otherwise.

FILES

/usr/lib/libU77.a

SEE ALSO

unlink(2), link(3F), filsys(5), perror(3F)

BUGS

Pathnames can be no longer than MAXPATHLEN as defined in <*sys/param.h*>.

NAME

wait – wait for a process to terminate

SYNOPSIS

integer function wait (status)
integer status

DESCRIPTION

Wait causes its caller to be suspended until a signal is received or one of its child processes terminates. If any child has terminated since the last *wait,* return is immediate; if there are no children, return is immediate with an error code.

If the returned value is positive, it is the process ID of the child and *status* is its termination status (see *wait*(2)). If the returned value is negative, it is the negation of a system error code.

FILES

/usr/lib/libU77.a

SEE ALSO

wait(2), signal(3F), kill(3F), perror(3F)

ICON/UXB OPERATING SYSTEM MATH LIBRARY

ICON®

NAME

intro – introduction to mathematical library functions

DESCRIPTION

These functions constitute the math library, *libm.* They are automatically loaded as needed by the Fortran compiler *f77*(1). The link editor searches this library under the "–lm" option. Declarations for these functions may be obtained from the include file <*math.h*>.

LIST OF FUNCTIONS

Name	*Appears on Page*	*Description*
acos	sin.3m	trigonometric functions
asin	sin.3m	trigonometric functions
atan	sin.3m	trigonometric functions
atan2	sin.3m	trigonometric functions
cabs	hypot.3m	Euclidean distance
ceil	floor.3m	absolute value, floor, ceiling functions
cos	sin.3m	trigonometric functions
cosh	sinh.3m	hyperbolic functions
exp	exp.3m	exponential, logarithm, power, square root
fabs	floor.3m	absolute value, floor, ceiling functions
floor	floor.3m	absolute value, floor, ceiling functions
gamma	gamma.3m	log gamma function
hypot	hypot.3m	Euclidean distance
j0	j0.3m	bessel functions
j1	j0.3m	bessel functions
jn	j0.3m	bessel functions
log	exp.3m	exponential, logarithm, power, square root
log10	exp.3m	exponential, logarithm, power, square root
pow	exp.3m	exponential, logarithm, power, square root
sin	sin.3m	trigonometric functions
sinh	sinh.3m	hyperbolic functions
sqrt	exp.3m	exponential, logarithm, power, square root
tan	sin.3m	trigonometric functions
tanh	sinh.3m	hyperbolic functions
y0	j0.3m	bessel functions
y1	j0.3m	bessel functions
yn	j0.3m	bessel functions

NAME

exp, log, log10, pow, sqrt – exponential, logarithm, power, square root

SYNOPSIS

```
#include <math.h>

double exp(x)
double x;

double log(x)
double x;

double log10(x)
double x;

double pow(x, y)
double x, y;

double sqrt(x)
double x;
```

DESCRIPTION

Exp returns the exponential function of *x*.

Log returns the natural logarithm of *x*; *log10* returns the base 10 logarithm.

Pow returns x^y.

Sqrt returns the square root of *x*.

SEE ALSO

hypot(3M), sinh(3M), intro(3M)

DIAGNOSTICS

Exp and *pow* return a huge value when the correct value would overflow; *errno* is set to ERANGE. *Pow* returns 0 and sets *errno* to EDOM when the second argument is negative and non-integral and when both arguments are 0.

Log returns 0 when *x* is zero or negative; *errno* is set to EDOM.

Sqrt returns 0 when *x* is negative; *errno* is set to EDOM.

NAME

fabs, floor, ceil – absolute value, floor, ceiling functions

SYNOPSIS

#include <math.h>

double floor(x)
double x;

double ceil(x)
double x;

double fabs(x)
double x;

DESCRIPTION

Fabs returns the absolute value $|x|$.

Floor returns the largest integer not greater than x.

Ceil returns the smallest integer not less than x.

SEE ALSO

abs(3)

NAME

gamma – log gamma function

SYNOPSIS

```
#include <math.h>

double gamma(x)
double x;
```

DESCRIPTION

Gamma returns ln $|\Gamma(|x|)|$. The sign of $\Gamma(|x|)$ is returned in the external integer *signgam*. The following C program might be used to calculate Γ:

```
y = gamma(x);
if (y > 88.0)
        error();
y = exp(y);
if(signgam)
        y = -y;
```

DIAGNOSTICS

A huge value is returned for negative integer arguments.

BUGS

There should be a positive indication of error.

NAME

hypot, cabs – Euclidean distance

SYNOPSIS

#include <math.h>

double hypot(x, y)
double x, y;

double cabs(z)
struct { double x, y;} z;

DESCRIPTION

Hypot and *cabs* return

sqrt(x*x + y*y),

taking precautions against unwarranted overflows.

SEE ALSO

exp(3M) for *sqrt*

NAME

j0, j1, jn, y0, y1, yn – bessel functions

SYNOPSIS

```
#include <math.h>

double j0(x)
double x;

double j1(x)
double x;

double jn(n, x)
double x;

double y0(x)
double x;

double y1(x)
double x;

double yn(n, x)
double x;
```

DESCRIPTION

These functions calculate Bessel functions of the first and second kinds for real arguments and integer orders.

DIAGNOSTICS

Negative arguments cause *y0, y1,* and *yn* to return a huge negative value and set *errno* to EDOM.

NAME

sin, cos, tan, asin, acos, atan, atan2 – trigonometric functions

SYNOPSIS

#include <math.h>

double sin(x)
double x;

double cos(x)
double x;

double asin(x)
double x;

double acos(x)
double x;

double atan(x)
double x;

double atan2(x, y)
double x, y;

DESCRIPTION

Sin, cos and *tan* return trigonometric functions of radian arguments. The magnitude of the argument should be checked by the caller to make sure the result is meaningful.

Asin returns the arc sin in the range $-\pi/2$ to $\pi/2$.

Acos returns the arc cosine in the range 0 to π.

Atan returns the arc tangent of *x* in the range $-\pi/2$ to $\pi/2$.

Atan2 returns the arc tangent of x/y in the range $-\pi$ to π.

DIAGNOSTICS

Arguments of magnitude greater than 1 cause *asin* and *acos* to return value 0; *errno* is set to EDOM. The value of *tan* at its singular points is a huge number, and *errno* is set to ERANGE.

BUGS

The value of *tan* for arguments greater than about 2**31 is garbage.

NAME
sinh, cosh, tanh – hyperbolic functions

SYNOPSIS
```
#include <math.h>

double sinh(x)

double cosh(x)
double x;

double tanh(x)
double x;
```

DESCRIPTION
These functions compute the designated hyperbolic functions for real arguments.

DIAGNOSTICS
Sinh and *cosh* return a huge value of appropriate sign when the correct value would overflow.

ICON/UXB OPERATING SYSTEM INTERNET NETWORK LIBRARY

ICON®

NAME

intro – introduction to network library functions

DESCRIPTION

This section describes functions that are applicable to the DARPA Internet network.

LIST OF FUNCTIONS

Name	*Appears on Page*	*Description*
endhostent	gethostent.3n	get network host entry
endnetent	getnetent.3n	get network entry
endprotoent	getprotoent.3n	get protocol entry
endservent	getservent.3n	get service entry
gethostbyaddr	gethostent.3n	get network host entry
gethostbyname	gethostent.3n	get network host entry
gethostent	gethostent.3n	get network host entry
getnetbyaddr	getnetent.3n	get network entry
getnetbyname	getnetent.3n	get network entry
getnetent	getnetent.3n	get network entry
getprotobyname	getprotoent.3n	get protocol entry
getprotobynumber	getprotoent.3n	get protocol entry
getprotoent	getprotoent.3n	get protocol entry
getservbyname	getservent.3n	get service entry
getservbyport	getservent.3n	get service entry
getservent	getservent.3n	get service entry
htonl	byteorder.3n	convert values between host and network byte order
htons	byteorder.3n	convert values between host and network byte order
inet_addr	inet.3n	Internet address manipulation routines
inet_lnaof	inet.3n	Internet address manipulation routines
inet_makeaddr	inet.3n	Internet address manipulation routines
inet_netof	inet.3n	Internet address manipulation routines
inet_network	inet.3n	Internet address manipulation routines
ntohl	byteorder.3n	convert values between host and network byte order
ntohs	byteorder.3n	convert values between host and network byte order
sethostent	gethostent.3n	get network host entry
setnetent	getnetent.3n	get network entry
setprotoent	getprotoent.3n	get protocol entry
setservent	getservent.3n	get service entry

NAME

htonl, htons, ntohl, ntohs – convert values between host and network byte order

SYNOPSIS

#include <sys/types.h>
#include <netinet/in.h>

netlong = htonl(hostlong);
u_long netlong, hostlong;

netshort = htons(hostshort);
u_short netshort, hostshort;

hostlong = ntohl(netlong);
u_long hostlong, netlong;

hostshort = ntohs(netshort);
u_short hostshort, netshort;

DESCRIPTION

These routines convert 16 and 32 bit quantities between network byte order and host byte order. On machines such as the SUN these routines are defined as null macros in the include file <*netinet/in.h*>.

These routines are most often used in conjunction with Internet addresses and ports as returned by *gethostent*(3N) and *getservent*(3N).

SEE ALSO

gethostent(3N), getservent(3N)

BUGS

The VAX handles bytes backwards from most everyone else in the world. This is not expected to be fixed in the near future.

NAME

gethostent, gethostbyaddr, gethostbyname, sethostent, endhostent – get network host entry

SYNOPSIS

#include <netdb.h>

struct hostent *gethostent()

struct hostent *gethostbyname(name)
char *name;

struct hostent *gethostbyaddr(addr, len, type)
char *addr; int len, type;

sethostent(stayopen)
int stayopen

endhostent()

DESCRIPTION

Gethostent, *gethostbyname*, and *gethostbyaddr* each return a pointer to an object with the following structure containing the broken-out fields of a line in the network host data base, */etc/hosts*.

```
struct  hostent {
        char    *h_name;        /* official name of host */
        char    **h_aliases;    /* alias list */
        int     h_addrtype;     /* address type */
        int     h_length;       /* length of address */
        char    *h_addr;        /* address */
};
```

The members of this structure are:

h_name	Official name of the host.
h_aliases	A zero terminated array of alternate names for the host.
h_addrtype	The type of address being returned; currently always AF_INET.
h_length	The length, in bytes, of the address.
h_addr	A pointer to the network address for the host. Host addresses are returned in network byte order.

Gethostent reads the next line of the file, opening the file if necessary.

Sethostent opens and rewinds the file. If the *stayopen* flag is non-zero, the host data base will not be closed after each call to *gethostent* (either directly, or indirectly through one of the other "gethost" calls).

Endhostent closes the file.

Gethostbyname and *gethostbyaddr* sequentially search from the beginning of the file until a matching host name or host address is found, or until EOF is encountered. Host addresses are supplied in network order.

FILES

/etc/hosts

SEE ALSO

hosts(5)

DIAGNOSTICS

Null pointer (0) returned on EOF or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved. Only the Internet address format is currently understood.

NAME

getnetent, getnetbyaddr, getnetbyname, setnetent, endnetent – get network entry

SYNOPSIS

```
#include <netdb.h>

struct netent *getnetent()

struct netent *getnetbyname(name)
char *name;

struct netent *getnetbyaddr(net)
long net;

setnetent(stayopen)
int stayopen

endnetent()
```

DESCRIPTION

Getnetent, *getnetbyname*, and *getnetbyaddr* each return a pointer to an object with the following structure containing the broken-out fields of a line in the network data base, */etc/networks*.

```
struct  netent {
        char    *n_name;        /* official name of net */
        char    **n_aliases;    /* alias list */
        int     n_addrtype;     /* net number type */
        long    n_net;          /* net number */
};
```

The members of this structure are:

n_name	The official name of the network.
n_aliases	A zero terminated list of alternate names for the network.
n_addrtype	The type of the network number returned; currently only AF_INET.
n_net	The network number. Network numbers are returned in machine byte order.

Getnetent reads the next line of the file, opening the file if necessary.

Setnetent opens and rewinds the file. If the *stayopen* flag is non-zero, the net data base will not be closed after each call to *getnetent* (either directly, or indirectly through one of the other "getnet" calls).

Endnetent closes the file.

Getnetbyname and *getnetbyaddr* sequentially search from the beginning of the file until a matching net name or net address is found, or until EOF is encountered. Network numbers are supplied in host order.

FILES

/etc/networks

SEE ALSO

networks(5)

DIAGNOSTICS

Null pointer (0) returned on EOF or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved. Only Internet network numbers are currently understood. Expecting network numbers to fit in no more than 32 bits is probably naive.

NAME

getprotoent, getprotobynumber, getprotobyname, setprotoent, endprotoent – get protocol entry

SYNOPSIS

#include <netdb.h>

struct protoent *getprotoent()

struct protoent *getprotobyname(name)
char *name;

struct protoent *getprotobynumber(proto)
int proto;

setprotoent(stayopen)
int stayopen

endprotoent()

DESCRIPTION

Getprotoent, *getprotobyname*, and *getprotobynumber* each return a pointer to an object with the following structure containing the broken-out fields of a line in the network protocol data base, */etc/protocols.*

```
struct protoent {
  char  *p_name;    /* official name of protocol */
  char  **p_aliases; /* alias list */
  long  p_proto;    /* protocol number */
};
```

The members of this structure are:

p_name	The official name of the protocol.
p_aliases	A zero terminated list of alternate names for the protocol.
p_proto	The protocol number.

Getprotoent reads the next line of the file, opening the file if necessary.

Setprotoent opens and rewinds the file. If the *stayopen* flag is non-zero, the net data base will not be closed after each call to *getprotoent* (either directly, or indirectly through one of the other "getproto" calls).

Endprotoent closes the file.

Getprotobyname and *getprotobynumber* sequentially search from the beginning of the file until a matching protocol name or protocol number is found, or until EOF is encountered.

FILES

/etc/protocols

SEE ALSO

protocols(5)

DIAGNOSTICS

Null pointer (0) returned on EOF or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved. Only the Internet protocols are currently understood.

NAME

getservent, getservbyport, getservbyname, setservent, endservent – get service entry

SYNOPSIS

#include <netdb.h>

struct servent *getservent()

struct servent *getservbyname(name, proto)
char *name, *proto;

struct servent *getservbyport(port, proto)
int port; char *proto;

setservent(stayopen)
int stayopen

endservent()

DESCRIPTION

Getservent, *getservbyname*, and *getservbyport* each return a pointer to an object with the following structure containing the broken-out fields of a line in the network services data base, */etc/services*.

```
struct servent {
  char  *s_name;    /* official name of service */
  char  **s_aliases; /* alias list */
  long  s_port;     /* port service resides at */
  char  *s_proto;   /* protocol to use */
};
```

The members of this structure are:

s_name	The official name of the service.
s_aliases	A zero terminated list of alternate names for the service.
s_port	The port number at which the service resides. Port numbers are returned in network byte order.
s_proto	The name of the protocol to use when contacting the service.

Getservent reads the next line of the file, opening the file if necessary.

Setservent opens and rewinds the file. If the *stayopen* flag is non-zero, the net data base will not be closed after each call to *getservent* (either directly, or indirectly through one of the other "getserv" calls).

Endservent closes the file.

Getservbyname and *getservbyport* sequentially search from the beginning of the file until a matching protocol name or port number is found, or until EOF is encountered. If a protocol name is also supplied (non-NULL), searches must also match the protocol.

FILES

/etc/services

SEE ALSO

getprotoent(3N), services(5)

DIAGNOSTICS

Null pointer (0) returned on EOF or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved. Expecting port numbers to fit in a 32 bit quantity is probably naive.

NAME

inet_addr, inet_network, inet_ntoa, inet_makeaddr, inet_lnaof, inet_netof – Internet address manipulation routines

SYNOPSIS

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

struct in_addr inet_addr(cp)
char *cp;

int inet_network(cp)
char *cp;

char *inet_ntoa(in)
struct inet_addr in;

struct in_addr inet_makeaddr(net, lna)
int net, lna;

int inet_lnaof(in)
struct in_addr in;

int inet_netof(in)
struct in_addr in;
```

DESCRIPTION

The routines *inet_addr* and *inet_network* each interpret character strings representing numbers expressed in the Internet standard "." notation, returning numbers suitable for use as Internet addresses and Internet network numbers, respectively. The routine *inet_ntoa* takes an Internet address and returns an ASCII string representing the address in "." notation. The routine *inet_makeaddr* takes an Internet network number and a local network address and constructs an Internet address from it. The routines *inet_netof* and *inet_lnaof* break apart Internet host addresses, returning the network number and local network address part, respectively.

All Internet address are returned in network order (bytes ordered from left to right). All network numbers and local address parts are returned as machine format integer values.

INTERNET ADDRESSES

Values specified using the "." notation take one of the following forms:

```
a.b.c.d
a.b.c
a.b
a
```

When four parts are specified, each is interpreted as a byte of data and assigned, from left to right, to the four bytes of an Internet address.

When a three part address is specified, the last part is interpreted as a 16-bit quantity and placed in the right most two bytes of the network address. This makes the three part address format convenient for specifying Class B network addresses as "128.net.host".

When a two part address is supplied, the last part is interpreted as a 24-bit quantity and placed in the right most three bytes of the network address. This makes the two part address format convenient for specifying Class A network addresses as "net.host".

When only one part is given, the value is stored directly in the network address without any byte rearrangement.

All numbers supplied as "parts" in a "." notation may be decimal, octal, or hexadecimal, as specified in the C language (i.e. a leading 0x or 0X implies hexadecimal; otherwise, a leading 0 implies octal; otherwise, the number is interpreted as decimal).

SEE ALSO

gethostent(3N), getnetent(3N), hosts(5), networks(5),

DIAGNOSTICS

The value –1 is returned by *inet_addr* and *inet_network* for malformed requests.

BUGS

The problem of host byte ordering versus network byte ordering is confusing. A simple way to specify Class C network addresses in a manner similar to that for Class B and Class A is needed. The string returned by *inet_ntoa* resides in a static memory area.

ICON/UXB OPERATING SYSTEM C STANDARD I/O LIBRARY SUBROUTINES

ICON®

NAME

stdio – standard buffered input/output package

SYNOPSIS

#include <stdio.h>

FILE *stdin;
FILE *stdout;
FILE *stderr;

DESCRIPTION

The functions described in section 3S constitute a user-level buffering scheme. The in-line macros *getc* and *putc*(3S) handle characters quickly. The higher level routines *gets*, *fgets*, *scanf*, *fscanf*, *fread*, *puts*, *fputs*, *printf*, *fprintf*, *fwrite* all use *getc* and *putc*; they can be freely intermixed.

A file with associated buffering is called a *stream*, and is declared to be a pointer to a defined type FILE. *Fopen*(3S) creates certain descriptive data for a stream and returns a pointer to designate the stream in all further transactions. There are three normally open streams with constant pointers declared in the include file and associated with the standard open files:

stdin	standard input file
stdout	standard output file
stderr	standard error file

A constant 'pointer' NULL (0) designates no stream at all.

An integer constant EOF (–1) is returned upon end of file or error by integer functions that deal with streams.

Any routine that uses the standard input/output package must include the header file *<stdio.h>* of pertinent macro definitions. The functions and constants mentioned in sections labeled 3S are declared in the include file and need no further declaration. The constants, and the following 'functions' are implemented as macros; redeclaration of these names is perilous: *getc*, *getchar*, *putc*, *putchar*, *feof*, *ferror*, *fileno*.

SEE ALSO

open(2), close(2), read(2), write(2), fread(3S), fseek(3S), f*(3S)

DIAGNOSTICS

The value EOF is returned uniformly to indicate that a FILE pointer has not been initialized with *fopen*, input (output) has been attempted on an output (input) stream, or a FILE pointer designates corrupt or otherwise unintelligible FILE data.

For purposes of efficiency, this implementation of the standard library has been changed to line buffer output to a terminal by default and attempts to do this transparently by flushing the output whenever a *read*(2) from the standard input is necessary. This is almost always transparent, but may cause confusion or malfunctioning of programs which use standard i/o routines but use *read*(2) themselves to read from the standard input.

In cases where a large amount of computation is done after printing part of a line on an output terminal, it is necessary to *fflush*(3S) the standard output before going off and computing so that the output will appear.

BUGS

The standard buffered functions do not interact well with certain other library and system functions, especially *vfork* and *abort*.

LIST OF FUNCTIONS

Name	*Appears on Page*	*Description*
clearerr	ferror.3s	stream status inquiries
fclose	fclose.3s	close or flush a stream
fdopen	fopen.3s	open a stream
feof	ferror.3s	stream status inquiries
ferror	ferror.3s	stream status inquiries
fflush	fclose.3s	close or flush a stream
fgetc	getc.3s	get character or word from stream
fgets	gets.3s	get a string from a stream
fileno	ferror.3s	stream status inquiries
fopen	fopen.3s	open a stream
fprintf	printf.3s	formatted output conversion
fputc	putc.3s	put character or word on a stream
fputs	puts.3s	put a string on a stream
fread	fread.3s	buffered binary input/output
freopen	fopen.3s	open a stream
fscanf	scanf.3s	formatted input conversion
fseek	fseek.3s	reposition a stream
ftell	fseek.3s	reposition a stream
fwrite	fread.3s	buffered binary input/output
getc	getc.3s	get character or word from stream
getchar	getc.3s	get character or word from stream
gets	gets.3s	get a string from a stream
getw	getc.3s	get character or word from stream
printf	printf.3s	formatted output conversion
putc	putc.3s	put character or word on a stream
putchar	putc.3s	put character or word on a stream
puts	puts.3s	put a string on a stream
putw	putc.3s	put character or word on a stream
rewind	fseek.3s	reposition a stream
scanf	scanf.3s	formatted input conversion
setbuf	setbuf.3s	assign buffering to a stream
setbuffer	setbuf.3s	assign buffering to a stream
setlinebuf	setbuf.3s	assign buffering to a stream
sprintf	printf.3s	formatted output conversion
sscanf	scanf.3s	formatted input conversion
ungetc	ungetc.3s	push character back into input stream

NAME

fclose, fflush – close or flush a stream

SYNOPSIS

#include <stdio.h>

fclose(stream)
FILE *stream;

fflush(stream)
FILE *stream;

DESCRIPTION

Fclose causes any buffers for the named *stream* to be emptied, and the file to be closed. Buffers allocated by the standard input/output system are freed.

Fclose is performed automatically upon calling *exit*(3).

Fflush causes any buffered data for the named output *stream* to be written to that file. The stream remains open.

SEE ALSO

close(2), fopen(3S), setbuf(3S)

DIAGNOSTICS

These routines return EOF if *stream* is not associated with an output file, or if buffered data cannot be transferred to that file.

NAME

ferror, feof, clearerr, fileno – stream status inquiries

SYNOPSIS

#include <stdio.h>

feof(stream)
FILE *stream;

ferror(stream)
FILE *stream

clearerr(stream)
FILE *stream

fileno(stream)
FILE *stream;

DESCRIPTION

Feof returns non-zero when end of file is read on the named input *stream*, otherwise zero.

Ferror returns non-zero when an error has occurred reading or writing the named *stream*, otherwise zero. Unless cleared by *clearerr*, the error indication lasts until the stream is closed.

Clrerr resets the error indication on the named *stream*.

Fileno returns the integer file descriptor associated with the *stream*, see *open*(2).

These functions are implemented as macros; they cannot be redeclared.

SEE ALSO

fopen(3S), open(2)

NAME

fopen, freopen, fdopen – open a stream

SYNOPSIS

#include <stdio.h>

FILE *fopen(filename, type)
char *filename, *type;

FILE *freopen(filename, type, stream)
char *filename, *type;
FILE *stream;

FILE *fdopen(fildes, type)
char *type;

DESCRIPTION

Fopen opens the file named by *filename* and associates a stream with it. *Fopen* returns a pointer to be used to identify the stream in subsequent operations.

Type is a character string having one of the following values:

"r"	open for reading
"w"	create for writing
"a"	append: open for writing at end of file, or create for writing

In addition, each *type* may be followed by a '+' to have the file opened for reading and writing. "r+" positions the stream at the beginning of the file, "w+" creates or truncates it, and "a+" positions it at the end. Both reads and writes may be used on read/write streams, with the limitation that an *fseek, rewind,* or reading an end-of-file must be used between a read and a write or vice-versa.

Freopen substitutes the named file in place of the open *stream*. It returns the original value of *stream*. The original stream is closed.

Freopen is typically used to attach the preopened constant names, **stdin, stdout, stderr,** to specified files.

Fdopen associates a stream with a file descriptor obtained from *open, dup, creat,* or *pipe*(2). The *type* of the stream must agree with the mode of the open file.

SEE ALSO

open(2), fclose(3)

DIAGNOSTICS

Fopen and *freopen* return the pointer **NULL** if *filename* cannot be accessed.

BUGS

Fdopen is not portable to systems other than UNIX.
The read/write *types* do not exist on all systems. Those systems without read/write modes will probably treat the *type* as if the '+' was not present. These are unreliable in any event.

NAME

fread, fwrite – buffered binary input/output

SYNOPSIS

#include <stdio.h>

fread(ptr, sizeof(*ptr), nitems, stream)
FILE *stream;

fwrite(ptr, sizeof(*ptr), nitems, stream)
FILE *stream;

DESCRIPTION

Fread reads, into a block beginning at *ptr*, *nitems* of data of the type of **ptr* from the named input *stream*. It returns the number of items actually read.

If *stream* is **stdin** and the standard output is line buffered, then any partial output line will be flushed before any call to *read*(2) to satisfy the *fread*.

Fwrite appends at most *nitems* of data of the type of **ptr* beginning at *ptr* to the named output *stream*. It returns the number of items actually written.

SEE ALSO

read(2), write(2), fopen(3S), getc(3S), putc(3S), gets(3S), puts(3S), printf(3S), scanf(3S)

DIAGNOSTICS

Fread and *fwrite* return 0 upon end of file or error.

NAME

fseek, ftell, rewind – reposition a stream

SYNOPSIS

#include <stdio.h>

fseek(stream, offset, ptrname)
FILE *stream;
long offset;

long ftell(stream)
FILE *stream;

rewind(stream)

DESCRIPTION

Fseek sets the position of the next input or output operation on the *stream*. The new position is at the signed distance *offset* bytes from the beginning, the current position, or the end of the file, according as *ptrname* has the value 0, 1, or 2.

Fseek undoes any effects of *ungetc*(3S).

Ftell returns the current value of the offset relative to the beginning of the file associated with the named *stream*. It is measured in bytes on UNIX; on some other systems it is a magic cookie, and the only foolproof way to obtain an *offset* for *fseek*.

Rewind(*stream*) is equivalent to *fseek*(*stream*, 0L, 0).

SEE ALSO

lseek(2), fopen(3S)

DIAGNOSTICS

Fseek returns –1 for improper seeks.

NAME

getc, getchar, fgetc, getw – get character or word from stream

SYNOPSIS

#include <stdio.h>

int getc(stream)
FILE *stream;

int getchar()

int fgetc(stream)
FILE *stream;

int getw(stream)
FILE *stream;

DESCRIPTION

Getc returns the next character from the named input *stream*.

Getchar() is identical to *getc*(*stdin*).

Fgetc behaves like *getc*, but is a genuine function, not a macro; it may be used to save object text.

Getw returns the next word (in a 32-bit integer) from the named input *stream*. It returns the constant EOF upon end of file or error, but since that is a good integer value, *feof* and *ferror*(3S) should be used to check the success of *getw*. *Getw* assumes no special alignment in the file.

SEE ALSO

fopen(3S), putc(3S), gets(3S), scanf(3S), fread(3S), ungetc(3S)

DIAGNOSTICS

These functions return the integer constant EOF at end of file or upon read error.

A stop with message, 'Reading bad file', means an attempt has been made to read from a stream that has not been opened for reading by *fopen*.

BUGS

The end-of-file return from *getchar* is incompatible with that in UNIX editions 1-6.

Because it is implemented as a macro, *getc* treats a *stream* argument with side effects incorrectly. In particular, 'getc(*f++);' doesn't work sensibly.

NAME

gets, fgets – get a string from a stream

SYNOPSIS

#include <stdio.h>

char *gets(s)
char *s;

char *fgets(s, n, stream)
char *s;
FILE *stream;

DESCRIPTION

Gets reads a string into *s* from the standard input stream **stdin**. The string is terminated by a newline character, which is replaced in *s* by a null character. *Gets* returns its argument.

Fgets reads *n*–1 characters, or up to a newline character, whichever comes first, from the *stream* into the string *s*. The last character read into *s* is followed by a null character. *Fgets* returns its first argument.

SEE ALSO

puts(3S), getc(3S), scanf(3S), fread(3S), ferror(3S)

DIAGNOSTICS

Gets and *fgets* return the constant pointer NULL upon end of file or error.

BUGS

Gets deletes a newline, *fgets* keeps it, all in the name of backward compatibility.

NAME

printf, fprintf, sprintf – formatted output conversion

SYNOPSIS

#include <stdio.h>

printf(format [, arg] ...)
char *format;

fprintf(stream, format [, arg] ...)
FILE *stream;
char *format;

sprintf(s, format [, arg] ...)
char *s, format;

#include <varargs.h>
_doprnt(format, args, stream)
char *format;
va_list *args;
FILE *stream;

DESCRIPTION

Printf places output on the standard output stream **stdout**. *Fprintf* places output on the named output *stream*. *Sprintf* places 'output' in the string *s*, followed by the character '\0'. All of these routines work by calling the internal routine **_doprnt,** using the variable-length argument facilities of *varargs*(3).

Each of these functions converts, formats, and prints its arguments after the first under control of the first argument. The first argument is a character string which contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which causes conversion and printing of the next successive *arg printf.*

Each conversion specification is introduced by the character %. Following the %, there may be

- an optional minus sign '–' which specifies *left adjustment* of the converted value in the indicated field;
- an optional digit string specifying a *field width;* if the converted value has fewer characters than the field width it will be blank-padded on the left (or right, if the left-adjustment indicator has been given) to make up the field width; if the field width begins with a zero, zero-padding will be done instead of blank-padding;
- an optional period '**.**' which serves to separate the field width from the next digit string;
- an optional digit string specifying a *precision* which specifies the number of digits to appear after the decimal point, for e- and f-conversion, or the maximum number of characters to be printed from a string;
- an optional '#' character specifying that the value should be converted to an "alternate form". For **c**, **d**, **s**, and **u**, conversions, this option has no effect. For **o** conversions, the precision of the number is increased to force the first character of the output string to a zero. For **x**(**X**) conversion, a non-zero result has the string **0x**(**0X**) prepended to it. For **e**, **E**, **f**, **g**, and **G**, conversions, the result will always contain a decimal point, even if no digits follow the point (normally, a decimal point only appears in the results of those conversions if a digit follows the decimal point).

For **g** and **G** conversions, trailing zeros are not removed from the result as they would otherwise be.

- the character **l** specifying that a following **d**, **o**, **x**, or **u** corresponds to a long integer *arg*.
- a character which indicates the type of conversion to be applied.

A field width or precision may be '*' instead of a digit string. In this case an integer *arg* supplies the field width or precision.

The conversion characters and their meanings are

dox	The integer *arg* is converted to decimal, octal, or hexadecimal notation respectively.
f	The float or double *arg* is converted to decimal notation in the style '[−]ddd.ddd' where the number of d's after the decimal point is equal to the precision specification for the argument. If the precision is missing, 6 digits are given; if the precision is explicitly 0, no digits and no decimal point are printed.
e	The float or double *arg* is converted in the style '[−]d.ddd**e**±dd' where there is one digit before the decimal point and the number after is equal to the precision specification for the argument; when the precision is missing, 6 digits are produced.
g	The float or double *arg* is printed in style **d**, in style **f**, or in style **e**, whichever gives full precision in minimum space.
c	The character *arg* is printed.
s	*Arg* is taken to be a string (character pointer) and characters from the string are printed until a null character or until the number of characters indicated by the precision specification is reached; however if the precision is 0 or missing all characters up to a null are printed.
u	The unsigned integer *arg* is converted to decimal and printed (the result will be in the range 0 through MAXUINT, where MAXUINT equals 4294967295).
%	Print a '%'; no argument is converted.

In no case does a non-existent or small field width cause truncation of a field; padding takes place only if the specified field width exceeds the actual width. Characters generated by *printf* are printed by *putc*(3S).

Examples
To print a date and time in the form 'Sunday, July 3, 10:02', where *weekday* and *month* are pointers to null-terminated strings:

```
printf("%s, %s %d, %02d:%02d", weekday, month, day, hour, min);
```

To print π to 5 decimals:

```
printf("pi = %.5f", 4*atan(1.0));
```

SEE ALSO

putc(3S), scanf(3S), ecvt(3)

BUGS

Very wide fields (>128 characters) fail.

NAME

putc, putchar, fputc, putw – put character or word on a stream

SYNOPSIS

#include <stdio.h>

int putc(c, stream)
char c;
FILE *stream;

putchar(c)

fputc(c, stream)
FILE *stream;

putw(w, stream)
FILE *stream;

DESCRIPTION

Putc appends the character *c* to the named output *stream*. It returns the character written.

Putchar(*c*) is defined as *putc*(*c*, **stdout**).

Fputc behaves like *putc*, but is a genuine function rather than a macro.

Putw appends word (that is, **int**) *w* to the output *stream*. It returns the word written. *Putw* neither assumes nor causes special alignment in the file.

SEE ALSO

fopen(3S), fclose(3S), getc(3S), puts(3S), printf(3S), fread(3S)

DIAGNOSTICS

These functions return the constant EOF upon error. Since this is a good integer, *ferror*(3S) should be used to detect *putw* errors.

BUGS

Because it is implemented as a macro, *putc* treats a *stream* argument with side effects improperly. In particular

putc(c, *f++);

doesn't work sensibly.

Errors can occur long after the call to *putc*.

NAME

puts, fputs – put a string on a stream

SYNOPSIS

#include <stdio.h>

puts(s)
char *s;

fputs(s, stream)
char *s;
FILE *stream;

DESCRIPTION

Puts copies the null-terminated string *s* to the standard output stream **stdout** and appends a newline character.

Fputs copies the null-terminated string *s* to the named output *stream*.

Neither routine copies the terminal null character.

SEE ALSO

fopen(3S), gets(3S), putc(3S), printf(3S), ferror(3S)
fread(3S) for *fwrite*

BUGS

Puts appends a newline, *fputs* does not, all in the name of backward compatibility.

NAME

scanf, fscanf, sscanf – formatted input conversion

SYNOPSIS

#include <stdio.h>

scanf(format [, pointer] . . . **)**
char *format;

fscanf(stream, format [, pointer] . . . **)**
FILE *stream;
char *format;

sscanf(s, format [, pointer] . . . **)**
char *s, *format;

DESCRIPTION

Scanf reads from the standard input stream **stdin**. *Fscanf* reads from the named input *stream*. *Sscanf* reads from the character string *s*. Each function reads characters, interprets them according to a format, and stores the results in its arguments. Each expects as arguments a control string *format*, described below, and a set of *pointer* arguments indicating where the converted input should be stored.

The control string usually contains conversion specifications, which are used to direct interpretation of input sequences. The control string may contain:

1. Blanks, tabs or newlines, which match optional white space in the input.
2. An ordinary character (not %) which must match the next character of the input stream.
3. Conversion specifications, consisting of the character %, an optional assignment suppressing character *, an optional numerical maximum field width, and a conversion character.

A conversion specification directs the conversion of the next input field; the result is placed in the variable pointed to by the corresponding argument, unless assignment suppression was indicated by *. An input field is defined as a string of non-space characters; it extends to the next inappropriate character or until the field width, if specified, is exhausted.

The conversion character indicates the interpretation of the input field; the corresponding pointer argument must usually be of a restricted type. The following conversion characters are legal:

% a single '%' is expected in the input at this point; no assignment is done.

d a decimal integer is expected; the corresponding argument should be an integer pointer.

o an octal integer is expected; the corresponding argument should be a integer pointer.

x a hexadecimal integer is expected; the corresponding argument should be an integer pointer.

s a character string is expected; the corresponding argument should be a character pointer pointing to an array of characters large enough to accept the string and a terminating '\0', which will be added. The input field is terminated by a space character or a newline.

c a character is expected; the corresponding argument should be a character pointer. The normal skip over space characters is suppressed in this case; to read the next non-space character, try '%1s'. If a field width is given, the corresponding argument should refer to a character array, and the indicated number of characters is read.

e
f　a floating point number is expected; the next field is converted accordingly and stored through the corresponding argument, which should be a pointer to a *float*. The input format for floating point numbers is an optionally signed string of digits possibly containing a decimal point, followed by an optional exponent field consisting of an E or e followed by an optionally signed integer.

[　indicates a string not to be delimited by space characters. The left bracket is followed by a set of characters and a right bracket; the characters between the brackets define a set of characters making up the string. If the first character is not circumflex (^), the input field is all characters until the first character not in the set between the brackets; if the first character after the left bracket is ^, the input field is all characters until the first character which is in the remaining set of characters between the brackets. The corresponding argument must point to a character array.

The conversion characters **d**, **o** and **x** may be capitalized or preceded by **l** to indicate that a pointer to **long** rather than to **int** is in the argument list. Similarly, the conversion characters **e** or **f** may be capitalized or preceded by **l** to indicate a pointer to **double** rather than to **float**. The conversion characters **d**, **o** and **x** may be preceded by **h** to indicate a pointer to **short** rather than to **int**.

The *scanf* functions return the number of successfully matched and assigned input items. This can be used to decide how many input items were found. The constant EOF is returned upon end of input; note that this is different from 0, which means that no conversion was done; if conversion was intended, it was frustrated by an inappropriate character in the input.

For example, the call

```
int i; float x; char name[50];
scanf("%d%f%s", &i, &x, name);
```

with the input line

```
25   54.32E−1   thompson
```

will assign to *i* the value 25, *x* the value 5.432, and *name* will contain *'thompson\0'*. Or,

```
int i; float x; char name[50];
scanf("%2d%f%*d%[1234567890]", &i, &x, name);
```

with input

```
56789 0123 56a72
```

will assign 56 to *i*, 789.0 to *x*, skip '0123', and place the string '56\0' in *name*. The next call to *getchar* will return 'a'.

SEE ALSO

atof(3), getc(3S), printf(3S)

DIAGNOSTICS

The *scanf* functions return EOF on end of input, and a short count for missing or illegal data items.

BUGS

The success of literal matches and suppressed assignments is not directly determinable.

NAME

setbuf, setbuffer, setlinebuf – assign buffering to a stream

SYNOPSIS

#include <stdio.h>

setbuf(stream, buf)
FILE *stream;
char *buf;

setbuffer(stream, buf, size)
FILE *stream;
char *buf;
int size;

setlinebuf(stream)
FILE *stream;

DESCRIPTION

The three types of buffering available are unbuffered, block buffered, and line buffered. When an output stream is unbuffered, information appears on the destination file or terminal as soon as written; when it is block buffered many characters are saved up and written as a block; when it is line buffered characters are saved up until a newline is encountered or input is read from stdin. *Fflush* (see *fclose*(3S)) may be used to force the block out early. Normally all files are block buffered. A buffer is obtained from *malloc*(3) upon the first *getc* or *putc*(3S) on the file. If the standard stream **stdout** refers to a terminal it is line buffered. The standard stream **stderr** is always unbuffered.

Setbuf is used after a stream has been opened but before it is read or written. The character array *buf* is used instead of an automatically allocated buffer. If *buf* is the constant pointer NULL, input/output will be completely unbuffered. A manifest constant BUFSIZ tells how big an array is needed:

char buf[BUFSIZ];

Setbuffer, an alternate form of *setbuf*, is used after a stream has been opened but before it is read or written. The character array *buf* whose size is determined by the *size* argument is used instead of an automatically allocated buffer. If *buf* is the constant pointer NULL, input/output will be completely unbuffered.

Setlinebuf is used to change *stdout* or *stderr* from block buffered or unbuffered to line buffered. Unlike *setbuf* and *setbuffer* it can be used at any time that the file descriptor is active.

A file can be changed from unbuffered or line buffered to block buffered by using *freopen* (see *fopen*(3S)). A file can be changed from block buffered or line buffered to unbuffered by using *freopen* followed by *setbuf* with a buffer argument of NULL.

SEE ALSO

fopen(3S), getc(3S), putc(3S), malloc(3), fclose(3S), puts(3S), printf(3S), fread(3S)

BUGS

The standard error stream should be line buffered by default.

The *setbuffer* and *setlinebuf* functions are not portable to non 4.2 BSD versions of UNIX.

NAME

ungetc – push character back into input stream

SYNOPSIS

#include <stdio.h>

ungetc(c, stream)
FILE *stream;

DESCRIPTION

Ungetc pushes the character *c* back on an input stream. That character will be returned by the next *getc* call on that stream. *Ungetc* returns *c*.

One character of pushback is guaranteed provided something has been read from the stream and the stream is actually buffered. Attempts to push EOF are rejected.

Fseek(3S) erases all memory of pushed back characters.

SEE ALSO

getc(3S), setbuf(3S), fseek(3S)

DIAGNOSTICS

Ungetc returns EOF if it can't push a character back.

SECTION 3X

ICON/UXB OPERATING SYSTEM OTHER LIBRARIES

ICON®

NAME

intro – introduction to miscellaneous library functions

DESCRIPTION

These functions constitute minor libraries and other miscellaneous run-time facilities. Most are available only when programming in C. The list below includes libraries which provide device independent plotting functions, terminal independent screen management routines for two dimensional non-bitmap display terminals, functions for managing data bases with inverted indexes, and sundry routines used in executing commands on remote machines. The routines *getdiskbyname*, *rcmd*, *rresvport*, *ruserok*, and *rexec* reside in the standard C run-time library "–lc". All other functions are located in separate libraries indicated in each manual entry.

FILES

/lib/libc.a
/usr/lib/libdbm.a
/usr/lib/libtermcap.a
/usr/lib/libcurses.a
/usr/lib/lib2648.a
/usr/lib/libplot.a

LIST OF FUNCTIONS

Name	*Appears on Page*	*Description*
arc	plot.3x	graphics interface
assert	assert.3x	program verification
circle	plot.3x	graphics interface
closepl	plot.3x	graphics interface
cont	plot.3x	graphics interface
curses	curses.3x	screen functions with "optimal" cursor motion
dbminit	dbm.3x	data base subroutines
delete	dbm.3x	data base subroutines
endfsent	getfsent.3x	get file system descriptor file entry
erase	plot.3x	graphics interface
fetch	dbm.3x	data base subroutines
firstkey	dbm.3x	data base subroutines
getdiskbyname	getdisk.3x	get disk description by its name
getfsent	getfsent.3x	get file system descriptor file entry
getfsfile	getfsent.3x	get file system descriptor file entry
getfsspec	getfsent.3x	get file system descriptor file entry
getfstype	getfsent.3x	get file system descriptor file entry
initgroups	initgroups.3x	initialize group access list
label	plot.3x	graphics interface
lib2648	lib2648.3x	subroutines for the HP 2648 graphics terminal
line	plot.3x	graphics interface
linemod	plot.3x	graphics interface
move	plot.3x	graphics interface
nextkey	dbm.3x	data base subroutines
plot: openpl	plot.3x	graphics interface
point	plot.3x	graphics interface

rcmd	rcmd.3x	routines for returning a stream to a remote command
rexec	rexec.3x	return stream to a remote command
rresvport	rcmd.3x	routines for returning a stream to a remote command
ruserok	rcmd.3x	routines for returning a stream to a remote command
setfsent	getfsent.3x	get file system descriptor file entry
space	plot.3x	graphics interface
store	dbm.3x	data base subroutines
tgetent	termcap.3x	terminal independent operation routines
tgetflag	termcap.3x	terminal independent operation routines
tgetnum	termcap.3x	terminal independent operation routines
tgetstr	termcap.3x	terminal independent operation routines
tgoto	termcap.3x	terminal independent operation routines
tputs	termcap.3x	terminal independent operation routines

NAME

assert – program verification

SYNOPSIS

#include <assert.h>

assert(expression)

DESCRIPTION

Assert is a macro that indicates *expression* is expected to be true at this point in the program. It causes an *exit*(2) with a diagnostic comment on the standard output when *expression* is false (0). Compiling with the *cc*(1) option **–DNDEBUG** effectively deletes *assert* from the program.

DIAGNOSTICS

'Assertion failed: file *f* line *n.' F* is the source file and *n* the source line number of the *assert* statement.

NAME

curses – screen functions with "optimal" cursor motion

SYNOPSIS

cc [flags] files **–lcurses –ltermcap** [libraries]

DESCRIPTION

These routines give the user a method of updating screens with reasonable optimization. They keep an image of the current screen, and the user sets up an image of a new one. Then the *refresh()* tells the routines to make the current screen look like the new one. In order to initialize the routines, the routine *initscr()* must be called before any of the other routines that deal with windows and screens are used. The routine *endwin()* should be called before exiting.

SEE ALSO

Screen Updating and Cursor Movement Optimization: A Library Package, Ken Arnold, ioctl(2), getenv(3), tty(4), termcap(5)

AUTHOR

Ken Arnold

FUNCTIONS

addch(ch)	add a character to *stdscr*
addstr(str)	add a string to *stdscr*
box(win,vert,hor)	draw a box around a window
crmode()	set cbreak mode
clear()	clear *stdscr*
clearok(scr,boolf)	set clear flag for *scr*
clrtobot()	clear to bottom on *stdscr*
clrtoeol()	clear to end of line on *stdscr*
delch()	delete a character
deleteln()	delete a line
delwin(win)	delete *win*
echo()	set echo mode
endwin()	end window modes
erase()	erase *stdscr*
getch()	get a char through *stdscr*
getcap(name)	get terminal capability *name*
getstr(str)	get a string through *stdscr*
gettmode()	get tty modes
getyx(win,y,x)	get (y,x) co-ordinates
inch()	get char at current (y,x) co-ordinates
initscr()	initialize screens
insch(c)	insert a char
insertln()	insert a line
leaveok(win,boolf)	set leave flag for *win*
longname(termbuf,name)	get long name from *termbuf*
move(y,x)	move to (y,x) on *stdscr*
mvcur(lasty,lastx,newy,newx)	actually move cursor
newwin(lines,cols,begin_y,begin_x)	create a new window
nl()	set newline mapping
nocrmode()	unset cbreak mode

noecho()	unset echo mode
nonl()	unset newline mapping
noraw()	unset raw mode
overlay(win1,win2)	overlay win1 on win2
overwrite(win1,win2)	overwrite win1 on top of win2
printw(fmt,arg1,arg2,...)	printf on *stdscr*
raw()	set raw mode
refresh()	make current screen look like *stdscr*
resetty()	reset tty flags to stored value
savetty()	stored current tty flags
scanw(fmt,arg1,arg2,...)	scanf through *stdscr*
scroll(win)	scroll *win* one line
scrollok(win,boolf)	set scroll flag
setterm(name)	set term variables for name
standend()	end standout mode
standout()	start standout mode
subwin(win,lines,cols,begin_y,begin_x)	create a subwindow
touchwin(win)	"change" all of *win*
unctrl(ch)	printable version of *ch*
waddch(win,ch)	add char to *win*
waddstr(win,str)	add string to *win*
wclear(win)	clear *win*
wclrtobot(win)	clear to bottom of *win*
wclrtoeol(win)	clear to end of line on *win*
wdelch(win,c)	delete char from *win*
wdeleteln(win)	delete line from *win*
werase(win)	erase *win*
wgetch(win)	get a char through *win*
wgetstr(win,str)	get a string through *win*
winch(win)	get char at current (y,x) in *win*
winsch(win,c)	insert char into *win*
winsertln(win)	insert line into *win*
wmove(win,y,x)	set current (y,x) co-ordinates on *win*
wprintw(win,fmt,arg1,arg2,...)	printf on *win*
wrefresh(win)	make screen look like *win*
wscanw(win,fmt,arg1,arg2,...)	scanf through *win*
wstandend(win)	end standout mode on *win*
wstandout(win)	start standout mode on *win*

NAME

dbminit, fetch, store, delete, firstkey, nextkey – data base subroutines

SYNOPSIS

```
typedef struct {
        char *dptr;
        int dsize;
} datum;

dbminit(file)
char *file;

datum fetch(key)
datum key;

store(key, content)
datum key, content;

delete(key)
datum key;

datum firstkey()

datum nextkey(key)
datum key;
```

DESCRIPTION

These functions maintain key/content pairs in a data base. The functions will handle very large (a billion blocks) databases and will access a keyed item in one or two file system accesses. The functions are obtained with the loader option **–ldbm**.

*Key*s and *content*s are described by the *datum* typedef. A *datum* specifies a string of *dsize* bytes pointed to by *dptr*. Arbitrary binary data, as well as normal ASCII strings, are allowed. The data base is stored in two files. One file is a directory containing a bit map and has '.dir' as its suffix. The second file contains all data and has '.pag' as its suffix.

Before a database can be accessed, it must be opened by *dbminit*. At the time of this call, the files *file*.**dir** and *file*.**pag** must exist. (An empty database is created by creating zero-length '.dir' and '.pag' files.)

Once open, the data stored under a key is accessed by *fetch* and data is placed under a key by *store*. A key (and its associated contents) is deleted by *delete*. A linear pass through all keys in a database may be made, in an (apparently) random order, by use of *firstkey* and *nextkey*. *Firstkey* will return the first key in the database. With any key *nextkey* will return the next key in the database. This code will traverse the data base:

```
        for (key = firstkey(); key.dptr != NULL; key = nextkey(key))
```

DIAGNOSTICS

All functions that return an *int* indicate errors with negative values. A zero return indicates ok. Routines that return a *datum* indicate errors with a null (0) *dptr*.

BUGS

The '.pag' file will contain holes so that its apparent size is about four times its actual content. Older UNIX systems may create real file blocks for these holes when touched. These files cannot be copied by normal means (cp, cat, tp, tar, ar) without filling in the holes.

Dptr pointers returned by these subroutines point into static storage that is changed by subsequent calls.

The sum of the sizes of a key/content pair must not exceed the internal block size (currently 1024 bytes). Moreover all key/content pairs that hash together must fit on a single block. *Store* will return an error in the event that a disk block fills with inseparable data.

Delete does not physically reclaim file space, although it does make it available for reuse.

The order of keys presented by *firstkey* and *nextkey* depends on a hashing function, not on anything interesting.

NAME

directory: opendir, readdir, telldir, seekdir, rewinddir, closedir – directory operations

SYNOPSIS

```
#include <sys/types.h>
#include <dirent.h>

DIR *opendir(filename)
char *filename;

struct dirent *readdir (dirp)
DIR *dirp;

long telldir (dirp)
DIR *dirp;

void seekdir (dirp, loc)
DIR *dirp;
long loc;

void rewinddir (dirp)
DIR *dirp;

void closedir (dirp)
DIR *dirp;
```

DESCRIPTION

Opendir opens the directory named by *filename* and associates a *directory stream* with it. *Opendir* returns a pointer to be used to identify the *directory stream* in subsequent operations. The pointer NULL is returned if *filename* cannot be accessed, or is not a directory, or if it cannot *malloc*(3X) enough memory to hold a DIR structure or a buffer for the directory entries.

Readdir returns a pointer to the next directory entry. No inactive entries are returned. It returns NULL upon reaching the end of the directory or upon detecting an invalid location in the directory.

Telldir returns the current location associated with the named *directory stream*.

Seekdir sets the position of the next *readdir* operation on the *directory stream*. The new position reverts to the one associated with the *directory stream* when the *telldir* operation from which *loc* was obtained was performed. Values returned by *telldir* are good only if the directory has not changed due to compaction or expansion.

Rewinddir resets the position of the named *directory stream* to the beginning of the directory.

Closedir closes the named *directory stream* and frees the DIR structure.

The following errors can occur as a result of these operations.

opendir:

[ENOTDIR]	A component of *filename* is not a directory.
[EACCES]	A component of *filename* denies search permission.
[EMFILE]	*Filename* points outside the allocated address space.

readdir:

[ENOENT]	The current file pointer for the directory is not located at a valid entry.
[EBADF]	The file descriptor determined by the DIR stream is no longer valid. This results if the DIR stream has been closed.

telldir, seekdir, and *closedir:*

[EBADF]	The file descriptor determined by the DIR stream is no longer valid. This results if the DIR stream has been closed.

EXAMPLE

Sample code which searchs a directory for entry *name:*

```
dirp = opendir(".");
while ( (dp = readdir( dirp
        if ( strcmp(dp->d_name, name ) ==
                closedir( dirp );
                return FOUND;
        }
closedir( dirp );
return NOT_FOUND;
```

WARNINGS

Rewinddir is implementd as a macro, so its function address cannot be taken.

NAME

getdiskbyname – get disk description by its name

SYNOPSIS

#include <disktab.h>

struct disktab *
getdiskbyname(name)
char *name;

DESCRIPTION

Getdiskbyname takes a disk name (e.g. rm03) and returns a structure describing its geometry information and the standard disk partition tables. All information obtained from the *disktab*(5) file.

<*disktab.h*> has the following form:

```
/*      disktab.h       4.3     83/08/11        */

/*
 * Disk description table, see disktab(5)
 */
#define         DISKTAB                 "/etc/disktab"

struct  disktab {
        char    *d_name;        /* drive name */
        char    *d_type;        /* drive type */
        int     d_secsize;      /* sector size in bytes */
        int     d_ntracks;      /* # tracks/cylinder */
        int     d_nsectors;     /* # sectors/track */
        int     d_ncylinders;   /* # cylinders */
        int     d_rpm;          /* revolutions/minute */
        struct  partition {
                int     p_size; /* #sectors in partition */
                short   p_bsize;/* block size in bytes */
                short   p_fsize;/* frag size in bytes */
        } d_partitions[8];
};

struct  disktab *getdiskbyname();
```

SEE ALSO

disktab(5)

BUGS

This information should be obtained from the system for locally available disks (in particular, the disk partition tables).

NAME

getfsent, getfsspec, getfsfile, getfstype, setfsent, endfsent – get file system descriptor file entry

SYNOPSIS

```
#include <fstab.h>

struct fstab *getfsent()

struct fstab *getfsspec(spec)
char *spec;

struct fstab *getfsfile(file)
char *file;

struct fstab *getfstype(type)
char *type;

int setfsent()

int endfsent()
```

DESCRIPTION

Getfsent, *getfsspec*, *getfstype*, and *getfsfile* each return a pointer to an object with the following structure containing the broken-out fields of a line in the file system description file, <fstab.h>.

```
struct fstab{
        char    *fs_spec;
        char    *fs_file;
        char    *fs_type;
        int     fs_freq;
        int     fs_passno;
};
```

The fields have meanings described in *fstab*(5).

Getfsent reads the next line of the file, opening the file if necessary.

Setfsent opens and rewinds the file.

Endfsent closes the file.

Getfsspec and *getfsfile* sequentially search from the beginning of the file until a matching special file name or file system file name is found, or until EOF is encountered. *Getfstype* does likewise, matching on the file system type field.

FILES

/etc/fstab

SEE ALSO

fstab(5)

DIAGNOSTICS

Null pointer (0) returned on EOF or error.

BUGS

All information is contained in a static area so it must be copied if it is to be saved.

NAME

initgroups – initialize group access list

SYNOPSIS

initgroups(name, basegid)
char *name;
int basegid;

DESCRIPTION

Initgroups reads through the group file and sets up, using the *setgroups*(2) call, the group access list for the user specified in *name*. The *basegid* is automatically included in the groups list. Typically this value is given as the group number from the password file.

FILES

/etc/group

SEE ALSO

setgroups(2)

DIAGNOSTICS

Initgroups returns -1 if it was not invoked by the super-user.

BUGS

Initgroups uses the routines based on *getgrent*(3). If the invoking program uses any of these routines, the group structure will be overwritten in the call to *initgroups*.

Noone seems to keep /etc/group up to date.

NAME

lib2648 – subroutines for the HP 2648 graphics terminal

SYNOPSIS

#include <stdio.h>

typedef char *bitmat;
FILE *trace;

cc file.c **–l2648**

DESCRIPTION

Lib2648 is a general purpose library of subroutines useful for interactive graphics on the Hewlett-Packard 2648 graphics terminal. To use it you must call the routine *ttyinit*() at the beginning of execution, and *done*() at the end of execution. All terminal input and output must go through the routines *rawchar*, *readline*, *outchar*, and *outstr*.

Lib2648 does the necessary ^E/^F handshaking if *getenv("TERM")* returns "hp2648", as it will if set by *tset*(1). Any other value, including for example "2648", will disable handshaking.

Bit matrix routines are provided to model the graphics memory of the 2648. These routines are generally useful, but are specifically useful for the *update* function which efficiently changes what is on the screen to what is supposed to be on the screen. The primative bit matrix routines are *newmat*, *mat*, and *setmat*.

The file *trace*, if non-null, is expected to be a file descriptor as returned by *fopen*. If so, *lib2648* will trace the progress of the output by writing onto this file. It is provided to make debugging output feasible for graphics programs without messing up the screen or the escape sequences being sent. Typical use of trace will include:

```
switch (argv[1][1]) {
case 'T':
        trace = fopen("trace", "w");
        break;
...
if (trace)
        fprintf(trace, "x is %d, y is %s\n", x, y);
...
dumpmat("before update", xmat);
```

ROUTINES

agoto(x, y)
: Move the alphanumeric cursor to position (x, y), measured from the upper left corner of the screen.

aoff() Turn the alphanumeric display off.

aon() Turn the alphanumeric display on.

areaclear(rmin, cmin, rmax, cmax)
: Clear the area on the graphics screen bordered by the four arguments. In normal mode the area is set to all black, in inverse video mode it is set to all white.

beep()
: Ring the bell on the terminal.

bitcopy(dest, src, rows, cols) bitmat dest,
Copy a *rows* by *cols* bit matrix from *src* to (user provided) *dest.*

cleara()
Clear the alphanumeric display.

clearg()
Clear the graphics display. Note that the 2648 will only clear the part of the screen that is visible if zoomed in.

curoff()
Turn the graphics cursor off.

curon()
Turn the graphics cursor on.

dispmsg(str, x, y, maxlen) char *str;
Display the message *str* in graphics text at position *(x, y)*. The maximum message length is given by *maxlen*, and is needed to for dispmsg to know how big an area to clear before drawing the message. The lower left corner of the first character is at *(x, y)*.

done()
Should be called before the program exits. Restores the tty to normal, turns off graphics screen, turns on alphanumeric screen, flushes the standard output, etc.

draw(x, y)
Draw a line from the pen location to *(x, y)*. As with all graphics coordinates, *(x, y)* is measured from the bottom left corner of the screen. *(x, y)* coordinates represent the first quadrant of the usual Cartesian system.

drawbox(r, c, color, rows, cols)
Draw a rectangular box on the graphics screen. The lower left corner is at location *(r, c)*. The box is *rows* rows high and *cols* columns wide. The box is drawn if *color* is 1, erased if *color* is 0. *(r, c)* absolute coordinates represent row and column on the screen, with the origin at the lower left. They are equivalent to *(x, y)* except for being reversed in order.

dumpmat(msg, m, rows, cols) char *msg; bitmat m;
If *trace* is non-null, write a readable ASCII representation of the matrix *m* on *trace*. *Msg* is a label to identify the output.

emptyrow(m, rows, cols, r) bitmat m;
Returns 1 if row *r* of matrix *m* is all zero, else returns 0. This routine is provided because it can be implemented more efficiently with a knowledge of the internal representation than a series of calls to *mat*.

error(msg) char *msg;
Default error handler. Calls *message(msg)* and returns. This is called by certain routines in *lib2648*. It is also suitable for calling by the user program. It is probably a good idea for a fancy graphics program to supply its own error procedure which uses *setjmp*(3) to restart the program.

gdefault()
Set the terminal to the default graphics modes.

goff() Turn the graphics display off.

gon() Turn the graphics display on.

koff() Turn the keypad off.

kon() Turn the keypad on. This means that most special keys on the terminal (such as the alphanumeric arrow keys) will transmit an escape sequence instead of doing their function locally.

line(x1, y1, x2, y2)
Draw a line in the current mode from *(x1, y1)* to *(x2, y2)*. This is equivalent to *move(x1, y1); draw(x2, y2);* except that a bug in the terminal involving repeated lines from the same point is compensated for.

lowleft()
Move the alphanumeric cursor to the lower left (home down) position.

mat(m, rows, cols, r, c) bitmat m;
Used to retrieve an element from a bit matrix. Returns 1 or 0 as the value of the *[r, c]* element of the *rows* by *cols* matrix *m*. Bit matrices are numbered *(r, c)* from the upper left corner of the matrix, beginning at (0, 0). *R* represents the row, and *c* represents the column.

message(str) char *str;
Display the text message *str* at the bottom of the graphics screen.

minmax(g, rows, cols, rmin, cmin, rmax, cmax) bitmat g;
int *rmin, *cmin, *rmax, *cmax;
Find the smallest rectangle that contains all the 1 (on) elements in the bit matrix g. The coordinates are returned in the variables pointed to by rmin, cmin, rmax, cmax.

move(x, y)
Move the pen to location *(x, y)*. Such motion is internal and will not cause output until a subsequent *sync()*.

movecurs(x, y)
Move the graphics cursor to location *(x, y)*.

bitmat newmat(rows, cols)
Create (with *malloc*(3)) a new bit matrix of size *rows* by *cols*. The value created (e.g. a pointer to the first location) is returned. A bit matrix can be freed directly with *free*.

outchar(c) char c;
Print the character *c* on the standard output. All output to the terminal should go through this routine or *outstr*.

outstr(str) char *str;
Print the string str on the standard output by repeated calls to *outchar*.

printg()
Print the graphics display on the printer. The printer must be configured as device 6 (the default) on the HPIB.

char rawchar()
Read one character from the terminal and return it. This routine or *readline* should be used to get all input, rather than *getchar*(3).

rboff()
Turn the rubber band line off.

rbon()
Turn the rubber band line on.

char *rdchar(c) char c;
Return a readable representation of the character *c*. If *c* is a printing character it returns itself, if a control character it is shown in the ^X notation, if negative an apostrophe is prepended. Space returns ^`, rubout returns ^?.

NOTE: A pointer to a static place is returned. For this reason, it will not work to pass rdchar twice to the same *fprintf*/*sprintf* call. You must instead save one of the values in your own buffer with strcpy.

readline(prompt, msg, maxlen) char *prompt, *msg;
Display *prompt* on the bottom line of the graphics display and read one line of text from the user, terminated by a newline. The line is placed in the buffer *msg*, which has size *maxlen* characters. Backspace processing is supported.

setclear()
Set the display to draw lines in erase mode. (This is reversed by inverse video mode.)

setmat(m, rows, cols, r, c, val) bitmat m;
The basic operation to store a value in an element of a bit matrix. The *[r, c]* element of *m* is set to *val,* which should be either 0 or 1.

setset()
Set the display to draw lines in normal (solid) mode. (This is reversed by inverse video mode.)

setxor()
Set the display to draw lines in exclusive or mode.

sync() Force all accumulated output to be displayed on the screen. This should be followed by fflush(stdout). The cursor is not affected by this function. Note that it is normally never necessary to call *sync*, since *rawchar* and *readline* call *sync()* and *fflush(stdout)* automatically.

togvid()
Toggle the state of video. If in normal mode, go into inverse video mode, and vice versa. The screen is reversed as well as the internal state of the library.

ttyinit()
Set up the terminal for processing. This routine should be called at the beginning of execution. It places the terminal in CBREAK mode, turns off echo, sets the proper modes in the terminal, and initializes the library.

update(mold, mnew, rows, cols, baser, basec)
bitmat mold, mnew; Make whatever changes are needed to make a window on the screen look like *mnew*. *Mold* is what the window on the screen currently looks like. The window has size *rows* by *cols*, and the lower left corner on the screen of the window is *[baser, basec]*. Note: *update* was not intended to be used for the entire screen. It would work but be very slow and take 64K bytes of memory just for mold and mnew. It was intended for 100 by 100 windows with objects in the center of them, and is quite fast for such windows.

vidinv()
Set inverse video mode.

vidnorm()
Set normal video mode.

zermat(m, rows, cols) bitmat m;
Set the bit matrix *m* to all zeros.

zoomn(size)
Set the hardware zoom to value *size,* which can range from 1 to 15.

zoomoff()
Turn zoom off. This forces the screen to zoom level 1 without affecting the current internal zoom number.

zoomon()
Turn zoom on. This restores the screen to the previously specified zoom size.

DIAGNOSTICS

The routine *error* is called when an error is detected. The only error currently detected is overflow of the buffer provided to *readline*.

Subscripts out of bounds to *setmat* return without setting anything.

FILES

/usr/lib/lib2648.a

SEE ALSO

fed(1)

AUTHOR

Mark Horton

BUGS

This library is not supported. It makes no attempt to use all of the features of the terminal, only those needed by fed. Contributions from users will be accepted for addition to the library.

The HP 2648 terminal is somewhat unreliable at speeds over 2400 baud, even with the ^E/^F handshaking. In an effort to improve reliability, handshaking is done every 32 characters. (The manual claims it is only necessary every 80 characters.) Nonetheless, I/O errors sometimes still occur.

There is no way to control the amount of debugging output generated on *trace* without modifying the source to the library.

NAME

plot: openpl, erase, label, line, circle, arc, move, cont, point, linemod, space, closepl – graphics interface

SYNOPSIS

openpl()

erase()

label(s)
char s[];

line(x1, y1, x2, y2)

circle(x, y, r)

arc(x, y, x0, y0, x1, y1)

move(x, y)

cont(x, y)

point(x, y)

linemod(s)
char s[];

space(x0, y0, x1, y1)

closepl()

DESCRIPTION

These subroutines generate graphic output in a relatively device-independent manner. See *plot*(5) for a description of their effect. *Openpl* must be used before any of the others to open the device for writing. *Closepl* flushes the output.

String arguments to *label* and *linemod* are null-terminated, and do not contain newlines.

Various flavors of these functions exist for different output devices. They are obtained by the following *ld*(1) options:

–lplot	device-independent graphics stream on standard output for *plot*(1) filters
–l300	GSI 300 terminal
–l300s	GSI 300S terminal
–l450	DASI 450 terminal
–l4014	Tektronix 4014 terminal

SEE ALSO

plot(5), plot(1G), graph(1G)

NAME

rcmd, rresvport, ruserok – routines for returning a stream to a remote command

SYNOPSIS

rem = rcmd(ahost, inport, locuser, remuser, cmd, fd2p);
char **ahost;
u_short inport;
char *locuser, *remuser, *cmd;
int *fd2p;

s = rresvport(port);
int *port;

ruserok(rhost, superuser, ruser, luser);
char *rhost;
int superuser;
char *ruser, *luser;

DESCRIPTION

Rcmd is a routine used by the super-user to execute a command on a remote machine using an authentication scheme based on reserved port numbers. *Rresvport* is a routine which returns a descriptor to a socket with an address in the privileged port space. *Ruserok* is a routine used by servers to authenticate clients requesting service with *rcmd*. All three functions are present in the same file and are used by the *rshd*(8C) server (among others).

Rcmd looks up the host **ahost* using *gethostbyname*(3N), returning –1 if the host does not exist. Otherwise **ahost* is set to the standard name of the host and a connection is established to a server residing at the well-known Internet port *inport*.

If the call succeeds, a socket of type SOCK_STREAM is returned to the caller, and given to the remote command as **stdin** and **stdout**. If *fd2p* is non-zero, then an auxiliary channel to a control process will be set up, and a descriptor for it will be placed in **fd2p*. The control process will return diagnostic output from the command (unit 2) on this channel, and will also accept bytes on this channel as being UNIX signal numbers, to be forwarded to the process group of the command. If *fd2p* is 0, then the **stderr** (unit 2 of the remote command) will be made the same as the **stdout** and no provision is made for sending arbitrary signals to the remote process, although you may be able to get its attention by using out-of-band data.

The protocol is described in detail in *rshd*(8C).

The *rresvport* routine is used to obtain a socket with a privileged address bound to it. This socket is suitable for use by *rcmd* and sevral other routines. Privileged addresses consist of a port in the range 0 to 1023. Only the super-user is allowed to bind an address of this sort to a socket.

Ruserok takes a remote host's name, as returned by a *gethostent*(3N) routine, two user names and a flag indicating if the local user's name is the super-user. It then checks the files */etc/hosts.equiv* and, possibly, *.rhosts* in the current working directory (normally the local user's home directory) to see if the request for service is allowed. A 1 is returned if the machine name is listed in the "hosts.equiv" file, or the host and remote user name are found in the ".rhosts" file; otherwise *ruserok* returns 0. If the *superuser* flag is 1, the checking of the "host.equiv" file is bypassed.

SEE ALSO

rlogin(1C), rsh(1C), rexec(3X), rexecd(8C), rlogind(8C), rshd(8C)

BUGS

There is no way to specify options to the *socket* call which *rcmd* makes.

NAME

rexec – return stream to a remote command

SYNOPSIS

rem = rexec(ahost, inport, user, passwd, cmd, fd2p);
char **ahost;
u_short inport;
char *user, *passwd, *cmd;
int *fd2p;

DESCRIPTION

Rexec looks up the host **ahost* using *gethostbyname*(3N), returning –1 if the host does not exist. Otherwise **ahost* is set to the standard name of the host. If a username and password are both specified, then these are used to authenticate to the foreign host; otherwise the environment and then the user's *.netrc* file in his home directory are searched for appropriate information. If all this fails, the user is prompted for the information.

The port *inport* specifies which well-known DARPA Internet port to use for the connection: it will normally be the value returned from the call "getservbyname("exec", "tcp")" (see *getservent*(3N)). The protocol for connection is described in detail in *rexecd*(8C).

If the call succeeds, a socket of type SOCK_STREAM is returned to the caller, and given to the remote command as **stdin** and **stdout**. If *fd2p* is non-zero, then a auxiliary channel to a control process will be setup, and a descriptor for it will be placed in **fd2p*. The control process will return diagnostic output from the command (unit 2) on this channel, and will also accept bytes on this channel as being UNIX signal numbers, to be forwarded to the process group of the command. If *fd2p* is 0, then the **stderr** (unit 2 of the remote command) will be made the same as the **stdout** and no provision is made for sending arbitrary signals to the remote process, although you may be able to get its attention by using out-of-band data.

SEE ALSO

rcmd(3X), rexecd(8C)

BUGS

There is no way to specify options to the *socket* call which *rexec* makes.

NAME

tgetent, tgetnum, tgetflag, tgetstr, tgoto, tputs – terminal independent operation routines

SYNOPSIS

```
char PC;
char *BC;
char *UP;
short ospeed;

tgetent(bp, name)
char *bp, *name;

tgetnum(id)
char *id;

tgetflag(id)
char *id;

char *
tgetstr(id, area)
char *id, **area;

char *
tgoto(cm, destcol, destline)
char *cm;

tputs(cp, affcnt, outc)
register char *cp;
int affcnt;
int (*outc)();
```

DESCRIPTION

These functions extract and use capabilities from the terminal capability data base *termcap*(5). These are low level routines; see *curses*(3X) for a higher level package.

Tgetent extracts the entry for terminal *name* into the buffer at *bp. Bp* should be a character buffer of size 1024 and must be retained through all subsequent calls to *tgetnum, tgetflag,* and *tgetstr. Tgetent* returns -1 if it cannot open the *termcap* file, 0 if the terminal name given does not have an entry, and 1 if all goes well. It will look in the environment for a TERMCAP variable. If found, and the value does not begin with a slash, and the terminal type **name** is the same as the environment string TERM, the TERMCAP string is used instead of reading the termcap file. If it does begin with a slash, the string is used as a path name rather than */etc/termcap*. This can speed up entry into programs that call *tgetent,* as well as to help debug new terminal descriptions or to make one for your terminal if you can't write the file */etc/termcap.*

Tgetnum gets the numeric value of capability *id,* returning -1 if is not given for the terminal. *Tgetflag* returns 1 if the specified capability is present in the terminal's entry, 0 if it is not. *Tgetstr* gets the string value of capability *id,* placing it in the buffer at *area,* advancing the *area* pointer. It decodes the abbreviations for this field described in *termcap*(5), except for cursor addressing and padding information.

Tgoto returns a cursor addressing string decoded from *cm* to go to column *destcol* in line *destline.* It uses the external variables **UP** (from the **up** capability) and **BC** (if **bc** is given rather than **bs**) if necessary to avoid placing **\n**, **^D** or **^@** in the returned string. (Programs which call tgoto should be sure to turn off the XTABS bit(s), since *tgoto* may now output a tab. Note that programs using termcap should in general turn off XTABS anyway since some terminals use control I for other functions, such as nondestructive space.) If a % sequence is given which is not understood, then *tgoto* returns "OOPS".

Tputs decodes the leading padding information of the string *cp*; *affcnt* gives the number of lines affected by the operation, or 1 if this is not applicable, *outc* is a routine which is called with each character in turn. The external variable *ospeed* should contain the output speed of the terminal as encoded by *stty*(3). The external variable **PC** should contain a pad character to be used (from the **pc** capability) if a null (^@) is inappropriate.

FILES

/usr/lib/libtermcap.a	–ltermcap library
/etc/termcap	data base

SEE ALSO

ex(1), curses(3X), termcap(5)

AUTHOR

William Joy

SECTION 4

ICON/UXB OPERATING SYSTEM SPECIAL FILES

ICON®

NAME

dmem, kmem – main memory

DESCRIPTION

Dmem is a special file that is an image of the disk cache processors physical memory. *Kmem* is a special file that is an image of the main memory of the computer. It may be used, for example, to examine (and even to patch) the system.

Byte addresses in *mem* are interpreted as kernel virtual memory addresses. References to non-existent locations cause errors to be returned.

Examining and patching device registers is likely to lead to unexpected results when read-only or write-only bits are present.

Kernel virtual addresses range from 0x40000000 to 0xffffffff. Disk cache addresses are in the range of zero to the amount of disk cache memory on the system.

FILES

/dev/dmem
/dev/kmem

BUGS

Memory files are accessed one byte at a time, an inappropriate method for some device registers.

NAME

null – data sink

DESCRIPTION

Data written on a null special file is discarded.

Reads from a null special file always return 0 bytes.

FILES

/dev/null

NAME
tty – general terminal interface

SYNOPSIS
#include <sgtty.h>

DESCRIPTION
This section describes both a particular special file **/dev/tty** and the terminal drivers used for conversational computing.

Line disciplines.

The system provides different *line disciplines* for controlling communications lines. In this version of the system there are three disciplines available:

old	The old (standard) terminal driver. This is used when using the standard shell *sh*(1) and for compatibility with other standard version 7 UNIX systems.
new	A newer terminal driver, with features for job control; this must be used when using *csh*(1).
net	A line discipline used for networking and loading data into the system over communications lines. It allows high speed input at very low overhead, and is described in *bk*(4).

Line discipline switching is accomplished with the TIOCSETD *ioctl:*

int ldisc = LDISC; ioctl(filedes, TIOCSETD, &ldisc);

where LDISC is OTTYDISC for the standard tty driver, NTTYDISC for the new driver and NETLDISC for the networking discipline. The standard (currently old) tty driver is discipline 0 by convention. The current line discipline can be obtained with the TIOCGETD ioctl. Pending input is discarded when the line discipline is changed.

All of the low-speed asynchronous communications ports can use any of the available line disciplines, no matter what hardware is involved. The remainder of this section discusses the "old" and "new" disciplines.

The control terminal.

When a terminal file is opened, it causes the process to wait until a connection is established. In practice, user programs seldom open these files; they are opened by *init*(8) and become a user's standard input and output file.

If a process which has no control terminal opens a terminal file, then that terminal file becomes the control terminal for that process. The control terminal is thereafter inherited by a child process during a *fork*(2), even if the control terminal is closed.

The file **/dev/tty** is, in each process, a synonym for a *control terminal* associated with that process. It is useful for programs that wish to be sure of writing messages on the terminal no matter how output has been redirected. It can also be used for programs that demand a file name for output, when typed output is desired and it is tiresome to find out which terminal is currently in use.

Process groups.

Command processors such as *csh*(1) can arbitrate the terminal between different *jobs* by placing related jobs in a single process group and associating this process group with the terminal. A terminals associated process group may be set using the TIOCSPGRP *ioctl*(2):

ioctl(fildes, TIOCSPGRP, &pgrp)

or examined using TIOCGPGRP rather than TIOCSPGRP, returning the current process group in *pgrp*. The new terminal driver aids in this arbitration by restricting access to the terminal by processes which are not in the current process group; see **Job access control** below.

Modes.

The terminal drivers have three major modes, characterized by the amount of processing on the input and output characters:

cooked The normal mode. In this mode lines of input are collected and input editing is done. The edited line is made available when it is completed by a newline or when an EOT (control-D, hereafter ^D) is entered. A carriage return is usually made synonymous with newline in this mode, and replaced with a newline whenever it is typed. All driver functions (input editing, interrupt generation, output processing such as delay generation and tab expansion, etc.) are available in this mode.

CBREAK This mode eliminates the character, word, and line editing input facilities, making the input character available to the user program as it is typed. Flow control, literal-next and interrupt processing are still done in this mode. Output processing is done.

RAW This mode eliminates all input processing and makes all input characters available as they are typed; no output processing is done either.

The style of input processing can also be very different when the terminal is put in non-blocking i/o mode; see *fcntl*(2). In this case a *read*(2) from the control terminal will never block, but rather return an error indication (EWOULDBLOCK) if there is no input available.

A process may also request a SIGIO signal be sent it whenever input is present. To enable this mode the FASYNC flag should be set using *fcntl*(2).

Input editing.

A UNIX terminal ordinarily operates in full-duplex mode. Characters may be typed at any time, even while output is occurring, and are only lost when the system's character input buffers become completely choked, which is rare, or when the user has accumulated the maximum allowed number of input characters that have not yet been read by some program. Currently this limit is 256 characters. In the old terminal driver all the saved characters are thrown away when the limit is reached, without notice; the new driver simply refuses to accept any further input, and rings the terminal bell.

Input characters are normally accepted in either even or odd parity with the parity bit being stripped off before the character is given to the program. By clearing either the EVEN or ODD bit in the flags word it is possible to have input characters with that parity discarded (see the **Summary** below.)

In all of the line disciplines, it is possible to simulate terminal input using the TIOCSTI ioctl, which takes, as its third argument, the address of a character. The system pretends that this character was typed on the argument terminal, which must be the control terminal except for the super-user (this call is not in standard version 7 UNIX).

Input characters are normally echoed by putting them in an output queue as they arrive. This may be disabled by clearing the ECHO bit in the flags word using the *stty*(3) call or the TIOCSETN or TIOCSETP ioctls (see the **Summary** below).

In cooked mode, terminal input is processed in units of lines. A program attempting to read will normally be suspended until an entire line has been received (but see the description of SIGTTIN in **Modes** above and FIONREAD in **Summary** below.) No matter how many characters are requested in the read call, at most one line will be returned. It is not, however, necessary to read a whole line at once; any number of characters may be requested in a read, even one, without losing information.

During input, line editing is normally done, with the character '#' logically erasing the last character typed and the character '@' logically erasing the entire current input line. These are often reset on crt's, with ^H replacing #, and ^U replacing @. These characters never erase beyond the beginning of the current input line or an ^D. These characters may be entered literally by preceding them with '\'; in the old teletype driver both the '\' and the character entered literally will appear on the screen; in the new driver the '\' will normally disappear.

The drivers normally treat either a carriage return or a newline character as terminating an input line, replacing the return with a newline and echoing a return and a line feed. If the CRMOD bit is cleared in the local mode word then the processing for carriage return is disabled, and it is simply echoed as a return, and does not terminate cooked mode input.

In the new driver there is a literal-next character ^V which can be typed in both cooked and CBREAK mode preceding **any** character to prevent its special meaning. This is to be preferred to the use of '\' escaping erase and kill characters, but '\' is (at least temporarily) retained with its old function in the new driver for historical reasons.

The new terminal driver also provides two other editing characters in normal mode. The word-erase character, normally ^W, erases the preceding word, but not any spaces before it. For the purposes of ^W, a word is defined as a sequence of non-blank characters, with tabs counted as blanks. Finally, the reprint character, normally ^R, retypes the pending input beginning on a new line. Retyping occurs automatically in cooked mode if characters which would normally be erased from the screen are fouled by program output.

Input echoing and redisplay

In the old terminal driver, nothing special occurs when an erase character is typed; the erase character is simply echoed. When a kill character is typed it is echoed followed by a new-line (even if the character is not killing the line, because it was preceded by a '\'!.)

The new terminal driver has several modes for handling the echoing of terminal input, controlled by bits in a local mode word.

Hardcopy terminals. When a hardcopy terminal is in use, the LPRTERA bit is normally set in the local mode word. Characters which are logically erased are then printed out backwards preceded by '\' and followed by '/' in this mode.

Crt terminals. When a crt terminal is in use, the LCRTBS bit is normally set in the local mode word. The terminal driver then echoes the proper number of erase characters when input is erased; in the normal case where the erase character is a ^H this causes the cursor of the terminal to back up to where it was before the logically erased character was typed. If the input has become fouled due to interspersed asynchronous output, the input is automatically retyped.

Erasing characters from a crt. When a crt terminal is in use, the LCRTERA bit may be set to cause input to be erased from the screen with a "backspace-space-backspace" sequence when character or word deleting sequences are used. A LCRTKIL bit may be set as well, causing the input to be erased in this manner on line kill sequences as well.

Echoing of control characters. If the LCTLECH bit is set in the local state word, then non-printing (control) characters are normally echoed as ^X (for some X) rather than being echoed unmodified; delete is echoed as ^?.

The normal modes for using the new terminal driver on crt terminals are speed dependent. At speeds less than 1200 baud, the LCRTERA and LCRTKILL processing is painfully slow, so *stty*(1) normally just sets LCRTBS and LCTLECH; at speeds of 1200 baud or greater all of these bits are normally set. *Stty*(1) summarizes these option settings and the use of the new terminal driver as "newcrt."

Output processing.

When one or more characters are written, they are actually transmitted to the terminal as soon as previously-written characters have finished typing. (As noted above, input characters are normally echoed by putting them in the output queue as they arrive.) When a process produces characters more rapidly than they can be typed, it will be suspended when its output queue exceeds some limit. When the queue has drained down to some threshold the program is resumed. Even parity is normally generated on output. The EOT character is not transmitted in cooked mode to prevent terminals that respond to it from hanging up; programs using raw or cbreak mode should be careful.

The terminal drivers provide necessary processing for cooked and CBREAK mode output including delay generation for certain special characters and parity generation. Delays are available after backspaces ^H, form feeds ^L, carriage returns ^M, tabs ^I and newlines ^J. The driver will also optionally expand tabs into spaces, where the tab stops are assumed to be set every eight columns. These functions are controlled by bits in the tty flags word; see **Summary** below.

The terminal drivers provide for mapping between upper and lower case on terminals lacking lower case, and for other special processing on deficient terminals.

Finally, in the new terminal driver, there is a output flush character, normally ^O, which sets the LFLUSHO bit in the local mode word, causing subsequent output to be flushed until it is cleared by a program or more input is typed. This character has effect in both cooked and CBREAK modes and causes pending input to be retyped if there is any pending input. An ioctl to flush the characters in the input and output queues TIOCFLUSH, is also available.

Upper case terminals and Hazeltines

If the LCASE bit is set in the tty flags, then all upper-case letters are mapped into the corresponding lower-case letter. The upper-case letter may be generated by preceding it by '\'. If the new terminal driver is being used, then upper case letters are preceded by a '\' when output. In addition, the following escape sequences can be generated on output and accepted on input:

for	\`	|	~	{	}
use	\\'	\\!	\\^	\\(	\\)

To deal with Hazeltine terminals, which do not understand that ~ has been made into an ASCII character, the LTILDE bit may be set in the local mode word when using the new terminal driver; in this case the character ~ will be replaced with the character \` on output.

Flow control.

There are two characters (the stop character, normally ^S, and the start character, normally ^Q) which cause output to be suspended and resumed respectively. Extra stop characters typed when output is already stopped have no effect, unless the start and stop characters are made the same, in which case output resumes.

A bit in the flags word may be set to put the terminal into TANDEM mode. In this mode the system produces a stop character (default ^S) when the input queue is in danger of overflowing, and a start character (default ^Q) when the input has drained sufficiently. This mode is useful when the terminal is actually another machine that obeys the conventions.

Line control and breaks.

There are several *ioctl* calls available to control the state of the terminal line. The TIOCSBRK ioctl will set the break bit in the hardware interface causing a break condition to exist; this can be cleared (usually after a delay with *sleep*(3)) by TIOCCBRK. Break conditions in the input are reflected as a null character in RAW mode or as the interrupt character in cooked or CBREAK mode. The TIOCCDTR ioctl will clear the data terminal ready condition; it can be set again by TIOCSDTR.

When the carrier signal from the dataset drops (usually because the user has hung up his terminal) a SIGHUP hangup signal is sent to the processes in the distinguished process group of the terminal; this usually causes them to terminate (the SIGHUP can be suppressed by setting the LNOHANG bit in the local state word of the driver.) Access to the terminal by other processes is then normally revoked, so any further reads will fail, and programs that read a terminal and test for end-of-file on their input will terminate appropriately.

When using an ACU it is possible to ask that the phone line be hung up on the last close with the TIOCHPCL ioctl; this is normally done on the outgoing line.

Interrupt characters.

There are several characters that generate interrupts in cooked and CBREAK mode; all are sent the processes in the control group of the terminal, as if a TIOCGPGRP ioctl were done to get the process group and then a *killpg*(2) system call were done, except that these characters also flush pending input and output when typed at a terminal (*à'la* TIOCFLUSH). The characters shown here are the defaults; the field names in the structures (given below) are also shown. The characters may be changed, although this is not often done.

^?	**t_intrc** (Delete) generates a SIGINT signal. This is the normal way to stop a process which is no longer interesting, or to regain control in an interactive program.
^\	**t_quitc** (FS) generates a SIGQUIT signal. This is used to cause a program to terminate and produce a core image, if possible, in the file **core** in the current directory.
^Z	**t_suspc** (EM) generates a SIGTSTP signal, which is used to suspend the current process group.
^Y	**t_dsuspc** (SUB) generates a SIGTSTP signal as ^Z does, but the signal is sent when a program attempts to read the ^Y, rather than when it is typed.

Job access control.

When using the new terminal driver, if a process which is not in the distinguished process group of its control terminal attempts to read from that terminal its process group is sent a SIGTTIN signal. This signal normally causes the members of that process group to stop. If, however, the process is ignoring SIGTTIN, has SIGTTIN blocked, is an *orphan process*, or is in the middle of process creation using *vfork*(2)), it is instead returned an end-of-file. (An *orphan process* is a process whose parent has exited and has been inherited by the *init*(8) process.) Under older UNIX systems these processes would typically have had their input files reset to **/dev/null,** so this is a compatible change.

When using the new terminal driver with the LTOSTOP bit set in the local modes, a process is prohibited from writing on its control terminal if it is not in the distinguished process group for that terminal. Processes which are holding or ignoring SIGTTOU signals, which are orphans, or which are in the middle of a *vfork*(2) are excepted and allowed to produce output.

Summary of modes.

Unfortunately, due to the evolution of the terminal driver, there are 4 different structures which contain various portions of the driver data. The first of these (**sgttyb**) contains that part of the information largely common between version 6 and version 7 UNIX systems. The second contains additional control characters added in version 7. The third is a word of local state peculiar to the new terminal driver, and the fourth is another structure of special characters added for the new driver. In the future a single structure may be made available to programs which need to access all this information; most programs need not concern themselves with all this state.

Basic modes: sgtty.

The basic *ioctl*s use the structure defined in <*sgtty.h*>:

```
struct sgttyb {
        char  sg_ispeed;
        char  sg_ospeed;
        char  sg_erase;
        char  sg_kill;
        short sg_flags;
};
```

The *sg_ispeed* and *sg_ospeed* fields describe the input and output speeds of the device according to the following table, which corresponds to the DEC DH-11 interface. If other hardware is used, impossible speed changes are ignored. Symbolic values in the table are as defined in <*sgtty.h*>.

B0	0	(hang up dataphone)
B50	1	50 baud
B75	2	75 baud
B110	3	110 baud
B134	4	134.5 baud
B150	5	150 baud
B200	6	200 baud
B300	7	300 baud
B600	8	600 baud
B1200	9	1200 baud
B1800	10	1800 baud
B2400	11	2400 baud
B4800	12	4800 baud
B9600	13	9600 baud
EXTA	14	External A
EXTB	15	External B

In the current configuration, only 110, 150, 300 and 1200 baud are really supported on dial-up lines. Code conversion and line control required for IBM 2741's (134.5 baud) must be implemented by the user's program. The half-duplex line discipline required for the 202 dataset (1200 baud) is not supplied; full-duplex 212 datasets work fine.

The *sg_erase* and *sg_kill* fields of the argument structure specify the erase and kill characters respectively. (Defaults are # and @.)

The *sg_flags* field of the argument structure contains several bits that determine the system's treatment of the terminal:

ALLDELAY	0177400	Delay algorithm selection
BSDELAY	0100000	Select backspace delays (not implemented):
BS0	0	
BS1	0100000	
VTDELAY	0040000	Select form-feed and vertical-tab delays:
FF0	0	
FF1	0100000	
CRDELAY	0030000	Select carriage-return delays:
CR0	0	
CR1	0010000	
CR2	0020000	

CR3	0030000	
TBDELAY	0006000	Select tab delays:
TAB0	0	
TAB1	0001000	
TAB2	0004000	
XTABS	0006000	
NLDELAY	0001400	Select new-line delays:
NL0	0	
NL1	0000400	
NL2	0001000	
NL3	0001400	
EVENP	0000200	Even parity allowed on input (most terminals)
ODDP	0000100	Odd parity allowed on input
RAW	0000040	Raw mode: wake up on all characters, 8-bit interface
CRMOD	0000020	Map CR into LF; echo LF or CR as CR-LF
ECHO	0000010	Echo (full duplex)
LCASE	0000004	Map upper case to lower on input
CBREAK	0000002	Return each character as soon as typed
TANDEM	0000001	Automatic flow control

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases a value of 0 indicates no delay.

Backspace delays are currently ignored but might be used for Terminet 300's.

If a form-feed/vertical tab delay is specified, it lasts for about 2 seconds.

Carriage-return delay type 1 lasts about .08 seconds and is suitable for the Terminet 300. Delay type 2 lasts about .16 seconds and is suitable for the VT05 and the TI 700. Delay type 3 is suitable for the concept-100 and pads lines to be at least 9 characters at 9600 baud.

New-line delay type 1 is dependent on the current column and is tuned for Teletype model 37's. Type 2 is useful for the VT05 and is about .10 seconds. Type 3 is unimplemented and is 0.

Tab delay type 1 is dependent on the amount of movement and is tuned to the Teletype model 37. Type 3, called XTABS, is not a delay at all but causes tabs to be replaced by the appropriate number of spaces on output.

Input characters with the wrong parity, as determined by bits 200 and 100, are ignored in cooked and CBREAK mode.

RAW disables all processing save output flushing with LFLUSHO; full 8 bits of input are given as soon as it is available; all 8 bits are passed on output. A break condition in the input is reported as a null character. If the input queue overflows in raw mode it is discarded; this applies to both new and old drivers.

CRMOD causes input carriage returns to be turned into new-lines; input of either CR or LF causes LF-CR both to be echoed (for terminals with a new-line function).

CBREAK is a sort of half-cooked (rare?) mode. Programs can read each character as soon as typed, instead of waiting for a full line; all processing is done except the input editing: character and word erase and line kill, input reprint, and the special treatment of \ or EOT are disabled.

TANDEM mode causes the system to produce a stop character (default ^S) whenever the input queue is in danger of overflowing, and a start character (default ^Q) when the input queue has drained sufficiently. It is useful for flow control when the 'terminal' is really another computer which understands the conventions.

Basic ioctls

In addition to the TIOCSETD and TIOCGETD disciplines discussed in **Line disciplines** above, a large number of other *ioctl*(2) calls apply to terminals, and have the general form:

#include <sgtty.h>

ioctl(fildes, code, arg)
struct sgttyb *arg;

The applicable codes are:

TIOCGETP	Fetch the basic parameters associated with the terminal, and store in the pointed-to *sgttyb* structure.
TIOCSETP	Set the parameters according to the pointed-to *sgttyb* structure. The interface delays until output is quiescent, then throws away any unread characters, before changing the modes.
TIOCSETN	Set the parameters like TIOCSETP but do not delay or flush input. Input is not preserved, however, when changing to or from RAW.

With the following codes the *arg* is ignored.

TIOCEXCL	Set "exclusive-use" mode: no further opens are permitted until the file has been closed.
TIOCNXCL	Turn off "exclusive-use" mode.
TIOCHPCL	When the file is closed for the last time, hang up the terminal. This is useful when the line is associated with an ACU used to place outgoing calls.
TIOCFLUSH	All characters waiting in input or output queues are flushed.

The remaining calls are not available in vanilla version 7 UNIX. In cases where arguments are required, they are described; *arg* should otherwise be given as 0.

TIOCSTI	the argument is the address of a character which the system pretends was typed on the terminal.
TIOCSBRK	the break bit is set in the terminal.
TIOCCBRK	the break bit is cleared.
TIOCSDTR	data terminal ready is set.
TIOCCDTR	data terminal ready is cleared.
TIOCGPGRP	arg is the address of a word into which is placed the process group number of the control terminal.
TIOCSPGRP	arg is a word (typically a process id) which becomes the process group for the control terminal.
FIONREAD	returns in the long integer whose address is arg the number of immediately readable characters from the argument unit. This works for files, pipes, and terminals, but not (yet) for multiplexed channels.

Tchars

The second structure associated with each terminal specifies characters that are special in both the old and new terminal interfaces: The following structure is defined in <*sys/ioctl.h*>, which is automatically included in <*sgtty.h*>:

```
struct tchars {
        char   t_intrc;        /* interrupt */
        char   t_quitc;        /* quit */
        char   t_startc;       /* start output */
        char   t_stopc;        /* stop output */
        char   t_eofc;         /* end-of-file */
        char   t_brkc;         /* input delimiter (like nl) */
};
```

The default values for these characters are ^?, ^\, ^Q, ^S, ^D, and -1. A character value of -1 eliminates the effect of that character. The *t_brkc* character, by default -1, acts like a new-line in that it terminates a 'line,' is echoed, and is passed to the program. The 'stop' and 'start' characters may be the same, to produce a toggle effect. It is probably counterproductive to make other special characters (including erase and kill) identical. The applicable ioctl calls are:

TIOCGETC Get the special characters and put them in the specified structure.

TIOCSETC Set the special characters to those given in the structure.

Local mode

The third structure associated with each terminal is a local mode word; except for the LNOHANG bit, this word is interpreted only when the new driver is in use. The bits of the local mode word are:

LCRTBS	000001 Backspace on erase rather than echoing erase
LPRTERA	000002 Printing terminal erase mode
LCRTERA	000004 Erase character echoes as backspace-space-backspace
LTILDE	000010 Convert ~ to ` on output (for Hazeltine terminals)
LMDMBUF	000020 Stop/start output when carrier drops
LLITOUT	000040 Suppress output translations
LTOSTOP	000100 Send SIGTTOU for background output
LFLUSHO	000200 Output is being flushed
LNOHANG	000400 Don't send hangup when carrier drops
LETXACK	001000 Diablo style buffer hacking (unimplemented)
LCRTKIL	002000 BS-space-BS erase entire line on line kill
LINTRUP	004000 Generate interrupt SIGTINT when input ready to read
LCTLECH	010000 Echo input control chars as ^X, delete as ^?
LPENDIN	020000 Retype pending input at next read or input character
LDECCTQ	040000 Only ^Q restarts output after ^S, like DEC systems

The applicable *ioctl* functions are:

TIOCLBIS arg is the address of a mask which is the bits to be set in the local mode

	word.
TIOCLBIC	arg is the address of a mask of bits to be cleared in the local mode word.
TIOCLSET	arg is the address of a mask to be placed in the local mode word.
TIOCLGET	arg is the address of a word into which the current mask is placed.

Local special chars

The final structure associated with each terminal is the *ltchars* structure which defines interrupt characters for the new terminal driver. Its structure is:

```
struct ltchars {
        char  t_suspc;      /* stop process signal */
        char  t_dsuspc;     /* delayed stop process
                                  signal */
        char  t_rprntc;     /* reprint line */
        char  t_flushc;     /* flush output (toggles) */
        char  t_werasc;     /* word erase */
        char  t_lnextc;     /* literal next character */
};
```

The default values for these characters are ^Z, ^Y, ^R, ^O, ^W, and ^V. A value of -1 disables the character.

The applicable *ioctl* functions are:

TIOCSLTC	args is the address of a *ltchars* structure which defines the new local special characters.
TIOCGLTC	args is the address of a *ltchars* structure into which is placed the current set of local special characters.

FILES

/dev/tty
/dev/tty*
/dev/console

SEE ALSO

csh(1), stty(1), ioctl(2), sigvec(2), stty(3C), getty(8), init(8)

BUGS

Half-duplex terminals are not supported.

SECTION 5

ICON/UXB OPERATING SYSTEM FILE FORMATS

ICON®

NAME
a.out – assembler and link editor output

SYNOPSIS
#include <a.out.h>

DESCRIPTION
A.out is the output file of the link editor *ld*(1). Ld makes *a.out* executable if there were no errors and no unresolved external references. Layout information as given in the include file for the ICON is:

```
/*
 * Header prepended to each a.out file.
 */
struct exec {
        long     a_magic;  /* magic number */
        unsigned a_text;   /* size of text segment */
        unsigned a_data;   /* size of initialized data */
        unsigned a_bss;    /* size of uninitialized data */
        unsigned a_syms;   /* size of symbol table */
        unsigned a_entry;  /* entry point */
        unsigned a_trsize; /* size of text relocation */
        unsigned a_drsize; /* size of data relocation */
};

#define  OMAGIC 0407       /* old impure format */
#define  NMAGIC 0410       /* read-only text */
#define  ZMAGIC 0413       /* demand load format */

/*
 * Macros which take exec structures as arguments and tell
 * whether the file has a reasonable magic number or offsets
 * to text |symbols |strings.
 */
#define  N_BADMAG(x) \
   (((x).a_magic)!=OMAGIC && ((x).a_magic)!=NMAGIC
   && ((x).a_magic)!=ZMAGIC)

#define  N_TXTOFF(x) \
   ((x).a_magic==ZMAGIC ? 1024 : sizeof (struct exec))
#define N_SYMOFF(x) \
   (N_TXTOFF(x) + (x).a_text+(x).a_data + (x).a_trsize
   +(x).a_drsize)
#define  N_STROFF(x) \
   (N_SYMOFF(x) + (x).a_syms)
```

The file has five sections: a header, the program text and data, relocation information, a symbol table and a string table (in that order). The last three may be omitted if the program was loaded with the '–s' option of *ld* or if the symbols and relocation have been removed by *strip*(1).

In the header the sizes of each section are given in bytes. The size of the header is not included in any of the other sizes.

When an *a.out* file is executed, three logical segments are set up: the text segment, the data segment (with uninitialized data, which starts off as all 0, following initialized), and a stack. The text segment begins at 0 in the core image; the header is not loaded. If the magic number in the header is OMAGIC (0407), it indicates that the text segment is not to be write-protected and shared, so the data segment is immediately contiguous with the text segment. This is the oldest kind of executable program and is rarely used. If the magic number is NMAGIC (0410) or ZMAGIC (0413), the data segment begins at the first 0 mod 1024 byte boundary following the text segment, and the text segment is not writable by the program; if other processes are executing the same file, they will share the text segment. For ZMAGIC format, the text segment begins at a 0 mod 1024 byte boundary in the *a.out* file, the remaining bytes after the header in the first block are reserved and should be zero. In this case the text and data sizes must both be multiples of 1024 bytes, and the pages of the file will be brought into the running image as needed, and not pre-loaded as with the other formats. This is especially suitable for very large programs and is the default format produced by *ld*(1).

The stack will occupy the highest possible locations in the core image: growing downwards from 0x3fffd000. The stack is automatically extended as required. The data segment is only extended as requested by *brk*(2).

After the header in the file follow the text, data, text relocation data relocation, symbol table and string table in that order. The text begins at the byte 1024 in the file for ZMAGIC format or just after the header for the other formats. The N_TXTOFF macro returns this absolute file position when given the name of an exec structure as argument. The data segment is contiguous with the text and immediately followed by the text relocation and then the data relocation information. The symbol table follows all this; its position is computed by the N_SYMOFF macro. Finally, the string table immediately follows the symbol table at a position which can be gotten easily using N_STROFF. The first 4 bytes of the string table are not used for string storage, but rather contain the size of the string table; this size INCLUDES the 4 bytes, the minimum string table size is thus 4.

The layout of a symbol table entry and the principal flag values that distinguish symbol types are given in the include file as follows:

```
/*
 * Format of a symbol table entry.
 */
struct nlist {
        union {
            char        *n_name; /* for use when in-core */
            long        n_strx;  /* index into file string table */
        } n_un;
        unsigned char n_type;    /* type flag, i.e. N_TEXT etc; see below */
        char          n_other;
        short         n_desc;    /* see <stab.h> */
        unsigned      n_value;   /* value of this symbol (or offset) */
};
#define n_hash        n_desc     /* used internally by ld */

/*
 * Simple values for n_type.
 */
#define N_UNDF        0x0        /* undefined */
```

```
#define N_ABS     0x2     /* absolute */
#define N_TEXT    0x4     /* text */
#define N_DATA    0x6     /* data */
#define N_BSS     0x8     /* bss */
#define N_COMM    0x12    /* common (internal to ld) */
#define N_FN      0x1f    /* file name symbol */

#define N_EXT     01      /* external bit, or'ed in */
#define N_TYPE    0x1e    /* mask for all the type bits */

/*
 * Other permanent symbol table entries have some of the N_STAB bits set.
 * These are given in <stab.h>
 */
#define N_STAB    0xe0    /* if any of these bits set, don't discard */

/*
 * Format for namelist values.
 */
#define N_FORMAT "%08x"
```

In the *a.out* file a symbol's n_un.n_strx field gives an index into the string table. A n_strx value of 0 indicates that no name is associated with a particular symbol table entry. The field n_un.n_name can be used to refer to the symbol name only if the program sets this up using n_strx and appropriate data from the string table.

If a symbol's type is undefined external, and the value field is non-zero, the symbol is interpreted by the loader *ld* as the name of a common region whose size is indicated by the value of the symbol.

The value of a byte in the text or data which is not a portion of a reference to an undefined external symbol is exactly that value which will appear in memory when the file is executed. If a byte in the text or data involves a reference to an undefined external symbol, as indicated by the relocation information, then the value stored in the file is an offset from the associated external symbol. When the file is processed by the link editor and the external symbol becomes defined, the value of the symbol will be added to the bytes in the file.

If relocation information is present, it amounts to eight bytes per relocatable datum as in the following structure:

```
/*
 * Format of a relocation datum.
 */
struct relocation_info {
  int       r_address;          /* address which is relocated */
  unsigned r_symbolnum:24, /* local symbol ordinal */
            r_pcrel:1,          /* was relocated pc relative already */
            r_length:2,         /* 0=byte, 1=word, 2=long */
            r_extern:1,         /* does not include value of sym referenced */
            :4;                 /* nothing, yet */
};
```

There is no relocation information if a_trsize+ a_drsize==0. If r_extern is 0, then r_symbolnum is actually a n_type for the relocation (i.e. N_TEXT meaning relative to segment text origin.)

SEE ALSO

adb(1), as(1), ld(1), nm(1), dbx(1), stab(5), strip(1)

BUGS

Not having the size of the string table in the header is a loss, but expanding the header size would have meant stripped executable file incompatibility, and we couldn't hack this just now.

NAME

acct – execution accounting file

SYNOPSIS

#include <sys/acct.h>

DESCRIPTION

The *acct*(2) system call makes entries in an accounting file for each process that terminates. The accounting file is a sequence of entries whose layout, as defined by the include file is:

```
/*      acct.h        6.1              83/07/29*/

/*
 * Accounting structures;
 * these use a comp_t type which is a 3 bits base 8
 * exponent, 13 bit fraction "floating point" number.
 */
typedef u_short comp_t;

struct  acct
{
        char          ac_comm[10];
                      /* Accounting command name */
        comp_t        ac_utime;
                      /* Accounting user time */
        comp_t        ac_stime;
                      /* Accounting system time */
        comp_t        ac_etime;
                      /* Accounting elapsed time */
        time_t        ac_btime;
                      /* Beginning time */
        short         ac_uid;
                      /* Accounting user ID */
        short         ac_gid;
                      /* Accounting group ID */
        short         ac_mem;
                      /* average memory usage */
        comp_t        ac_io;
                      /* number of disk IO blocks */
        dev_t         ac_tty;
                      /* control typewriter */
        char          ac_flag;
                      /* Accounting flag */
};

#define AFORK         0001
                      /* has executed fork, but no exec */
#define ASU           0002
                      /* used super-user privileges */
#define ACOMPAT       0004
                      /* used compatibility mode */
#define ACORE         0010
```

```
                        /* dumped core */
#define AXSIG           0020
                        /* killed by a signal */

#ifdef KERNEL
struct  acct            acctbuf;
struct  inode           *acctp;
#endif
```

If the process does an *execve*(2), the first 10 characters of the filename appear in *ac_comm*. The accounting flag contains bits indicating whether *execve*(2) was ever accomplished, and whether the process ever had super-user privileges.

SEE ALSO

acct(2), execve(2), sa(8)

NAME

aliases – aliases file for sendmail

SYNOPSIS

/usr/lib/aliases

DESCRIPTION

This file describes user id aliases used by */usr/lib/sendmail.* It is formatted as a series of lines of the form

name: name_1, name2, name_3, . . .

The *name* is the name to alias, and the *name_n* are the aliases for that name. Lines beginning with white space are continuation lines. Lines beginning with '#' are comments.

Aliasing occurs only on local names. Loops can not occur, since no message will be sent to any person more than once.

After aliasing has been done, local and valid recipients who have a ".forward" file in their home directory have messages forwarded to the list of users defined in that file.

This is only the raw data file; the actual aliasing information is placed into a binary format in the files */usr/lib/aliases.dir* and */usr/lib/aliases.pag* using the program *newaliases*(1). A *newaliases* command should be executed each time the aliases file is changed for the change to take effect.

SEE ALSO

newaliases(1), dbm(3X), sendmail(8)
SENDMAIL Installation and Operation Guide.
SENDMAIL An Internetwork Mail Router.

BUGS

Because of restrictions in *dbm*(3X) a single alias cannot contain more than about 1000 bytes of information. You can get longer aliases by "chaining"; that is, make the last name in the alias be a dummy name which is a continuation alias.

NAME

ar – archive (library) file format

SYNOPSIS

#include <ar.h>

DESCRIPTION

The archive command *ar* combines several files into one. Archives are used mainly as libraries to be searched by the link-editor *ld.*

A file produced by *ar* has a magic string at the start, followed by the constituent files, each preceded by a file header. The magic number and header layout as described in the include file are:

```
/*      ar.h       4.183/05/03*/

#define ARMAG  "!<arch>\n"
#define SARMAG 8

#define ARFMAG "`\n"

struct ar_hdr {
        char      ar_name[16];
        char      ar_date[12];
        char      ar_uid[6];
        char      ar_gid[6];
        char      ar_mode[8];
        char      ar_size[10];
        char      ar_fmag[2];
};
```

The name is a blank-padded string. The *ar_fmag* field contains ARFMAG to help verify the presence of a header. The other fields are left-adjusted, blank-padded numbers. They are decimal except for *ar_mode*, which is octal. The date is the modification date of the file at the time of its insertion into the archive.

Each file begins on a even (0 mod 2) boundary; a new-line is inserted between files if necessary. Nevertheless the size given reflects the actual size of the file exclusive of padding.

There is no provision for empty areas in an archive file.

The encoding of the header is portable across machines. If an archive contains printable files, the archive itself is printable.

SEE ALSO

ar(1), ld(1), nm(1)

BUGS

File names lose trailing blanks. Most software dealing with archives takes even an included blank as a name terminator.

NAME

core – format of memory image file

SYNOPSIS

#include <machine/param.h>
#include <sys/user.h>
#include <sys/proc.h>

DESCRIPTION

The UNIX System writes out a memory image of a terminated process when any of various errors occur. See *sigvec*(2) for the list of reasons; the most common are memory violations, illegal instructions, bus errors, and user-generated quit signals. The memory image is called 'core' and is written in the process's working directory (provided it can be; normal access controls apply).

The maximum size of a *core* file is limited by *setrlimit*(2). Files which would be larger than the limit are not created.

The core file consists of the *u*. area, whose size (in bytes) is defined by the UBYTES manifest in the *<machine/param.h>* file. The *u*. area starts with a *user* structure as given in *<sys/user.h>*. The remainder of the core file consists of the supervisor stack area, whose size is given (in bytes) by the SUPERSTACKSIZE manifest in the *<machine/param.h>* file, the proc structure, whose size is given (in bytes) by the PROCSIZE manifest in the *<machine/param.h>* file, the data pages and then the stack pages of the process image. The amount of data space image in the core file is given (in bytes) by the variables *p_segmap[DATA_SEG].segsize + p_segmap[BSS_SEG].segsize* in the *proc* area. If the program that produced the core was an OMAGIC program, the data size will include the text size, *p_segmap[TEXT_SEG].segsize,* also from the *proc* area (this segment will precede the data segments). The amount of stack image in the core file is given (in bytes) by the variable *p_segmap[STACK_SEG].segsize* in the *proc* area.

In general the debugger *adb*(1) is sufficient to deal with core images.

SEE ALSO

adb(1), dbx(1), sigvec(2), setrlimit(2)

NAME

dir – format of directories

SYNOPSIS

#include <sys/types.h>
#include <sys/dir.h>

DESCRIPTION

A directory behaves exactly like an ordinary file, save that no user may write into a directory. The fact that a file is a directory is indicated by a bit in the flag word of its i-node entry; see *fs*(5). The structure of a directory entry as given in the include file is:

```
/*
 * A directory consists of some number of blocks of DIRBLKSIZ
 * bytes, where DIRBLKSIZ is chosen such that it can be transferred
 * to disk in a single atomic operation (e.g. 512 bytes on most machines).
 *
 * Each DIRBLKSIZ byte block contains some number of directory entry
 * structures, which are of variable length. Each directory entry has
 * a struct direct at the front of it, containing its inode number,
 * the length of the entry, and the length of the name contained in
 * the entry. These are followed by the name padded to a 4 byte boundary
 * with null bytes. All names are guaranteed null terminated.
 * The maximum length of a name in a directory is MAXNAMLEN.
 *
 * The macro DIRSIZ(dp) gives the amount of space required to represent
 * a directory entry. Free space in a directory is represented by
 * entries which have dp->d_reclen > DIRSIZ(dp). All DIRBLKSIZ bytes
 * in a directory block are claimed by the directory entries. This
 * usually results in the last entry in a directory having a large
 * dp->d_reclen. When entries are deleted from a directory, the
 * space is returned to the previous entry in the same directory
 * block by increasing its dp->d_reclen. If the first entry of
 * a directory block is free, then its dp->d_ino is set to 0.
 * Entries other than the first in a directory do not normally have
 * dp->d_ino set to 0.
 */
#ifdef KERNEL
#define DIRBLKSIZ DEV_BSIZE
#else
#define  DIRBLKSIZ 512
#endif

#define MAXNAMLEN 255

/*
 * The DIRSIZ macro gives the minimum record
 * length which will hold the directory entry.
 * This requires the amount of space in struct
 * direct without the d_name field, plus enough
 * space for the name with a terminating null
 * byte (dp->d_namlen+1), rounded up to a 4
 * byte boundary.
 */
#undef DIRSIZ
#define DIRSIZ(dp) \
   ((sizeof (struct direct) - (MAXNAMLEN+1)) +
   (((dp)->d_namlen+1 + 3) &~ 3))

struct   direct {
         u_long      d_ino;
         short       d_reclen;
         short       d_namlen;
         char        d_name[MAXNAMLEN + 1];
         /* typically shorter */
};

struct _dirdesc {
         int         dd_fd;
         long        dd_loc;
         long        dd_size;
         char        dd_buf[DIRBLKSIZ];
};
```

By convention, the first two entries in each directory are for '.' and '..'. The first is an entry for the directory itself. The second is for the parent directory. The meaning of '..' is modified for the root directory of the master file system ("/"), where '..' has the same meaning as '.'.

SEE ALSO

fs(5)

NAME

dosdisks – list of MPS/DOS virtual disks

DESCRIPTION

The file */etc/dosdisks* contains a list of the pathnames for all files to be used as vdisks for MPS/DOS. The files are created by *dosdisk(8)* and each new file pathname is appended to */etc/dosdisks* by *dosdisk*. The vdisks are accessed in the order in which they appear in */etc/dosdisks;* the order the filenames appear may be changed to cause the vdisks to have different MPS/DOS assignments. To delete a vdisk, remove the MPS/UX file, then edit */etc/dosdisks* and remove the line specifying the deleted vdisk. The space for the deleted vdisk will be be reclaimed when MPS/UX is rebooted. Removing a 'd' partition vdisk is somewhat more involved; contact Icon for further assistance.

The first disk to appear should always be bootable, or MPS/DOS will be unable to initialize. See "Technical Note on Dosc and Proc/286 Support" for full details of vdisk support.

FILES

/etc/dosprinters

See Also

dosdisk(8), Technical Note on Dosc

NAME

dosprinters – destinations for spooled output from SLPT printers

DESCRIPTION

The file */etc/dosprinters* is read by the *dosprint* program and specifies destination and options for the SLPT printers used under MPS/DOS. It contains zero or more lines in the following format:

```
n pr [opt]
```

where "n" is the SLPT printer number (0-7), "pr" is the printer name *lpr(1)* is to use for printing, and "[opt]" is an optional string which is passed to lpr which can be used to set various modes. For example,

```
1 lp
3 lp -p
7 laser3
```

specifies that the output from SLPT1 should be spooled to "lp" (this is actually the default); the output from SLPT3 is spooled to "lp" with the -p flag (which causes *lpr* to pass the file through the "pr" filter); and the output from SLPT7 is spooled to a printer known in /etc/printcap as "laser3". Notice that it is not necessary to specify an entry for all 8 printers; all SLPT devices default to "lp" with no options.

FILES

/etc/dosprinters

See Also

lpr(1), printcap(5), Technical Note on Dosc

NAME

dstrules – Daylight savings time and time zone name rule file.

DESCRIPTION

The *dstrules* file contains a set of rules for daylight savings time, and time zone names. This allows for modification of daylight savings time rules or time zone names without recompilation. Upon its initial invocation in any process, the *ctime(3)* library routine reads the *dstrules* configuration file for a set of rules. If none are found, it uses a default table of rules which are current as of April 1, 1987. The same holds true for *timezone(3)*.

The general format of the file is:

```
%R     offset   hemisphere    # Rule 0
yeareffective1  startday      endday
yeareffective2  startday      endday
yeareffective3  startday      endday
       .
       .
       .
yeareffectiven  startday      endday
9999            0             0

%R     offset   hemisphere    # Rule n
       .
       .
       .

%Z
minuteswest1    standardname  dstname # Time zone 1
minuteswest2    standardname  dstname # Time zone 2
minuteswest3    standardname  dstname # Time zone 3
       .
       .
       .
minuteswestn    standardname  dstname # Time zone n
```

Comments begin with a "#" and are ended with the end of the line. Fields must be separated by tabs.

Each rule begins with **%R** and must be ended with a lambda which is an impossible date in the future, for example 9999. In a rule, *offset* is the number of hours time is to be shifted during daylight savings time. *Hemisphere* is one of **N** or **S** denoting the northern or southern hemispheres, respectively. The parameter *yeareffective* is the year that begins the period during which daylight savings time is in effect between *startday* and *endday*. Let us consider the following example of a rule definition:

```
%R    1     N
1970  119   303
1974  5     333
1975  58    303
1976  119   303
1987  96    303
```

```
9999  0     0
```

In the example shown above, from 1790 to 1973 daylight savings time begins on the Sunday closest to the 119TH day and ends on the Sunday closes to the 303RD day. During 1974, daylight savings time begins on or about the 5TH day and ends on or about the 333RD day, and so forth.

The time zone name definition section begins with **%Z.** If you use **%Z** more than once in your *dstrules* file, the table may not be parsed correctly, and the default tables compiled into *timezone(3)* will be used. In a time zone name, *minuteswest* is the number of minutes west of GMT for that zone. *Standardname* is the name for the zone when no daylight savings time is in effect, and *dstname* is the name for the zone when daylight savings time is in effect. The entry "*" for a zone name is interpreted as a null string. If you use a null string for *dstname* when daylight savings time is in effect, *timezone(3)* may become confused, and create its own string.

BUGS

The daylight savings time rules for parts of Europe are not confirmed.

Daylight savings time must begin on a Sunday.

It is not possible to give more than one timezone name to a particular offset from GMT without the rule file parser becoming extreemly confused.

FILES

/etc/dstrules

SEE ALSO

ctime(3), timezone(3), date(1)

NAME

dump, dumpdates – incremental dump format

SYNOPSIS

#include <sys/types.h>
#include <sys/inode.h>
#include <dumprestor.h>

DESCRIPTION

Tapes used by *dump* and *restore*(8) contain:

a header record
two groups of bit map records
a group of records describing directories
a group of records describing files

The format of the header record and of the first record of each description as given in the include file <*dumprestor.h*> is:

```
#define NTREC       10
#define MLEN        16
#define MSIZ        4096

#define TS_TAPE     1
#define TS_INODE    2
#define TS_BITS     3
#define TS_ADDR     4
#define TS_END      5
#define TS_CLRI     6
#define MAGIC       (int) 60011
#define CHECKSUM (int) 84446

struct  spcl {
        int         c_type;
        time_t      c_date;
        time_t      c_ddate;
        int         c_volume;
        daddr_t             c_tapea;
        ino_t       c_inumber;
        int         c_magic;
        int         c_checksum;
        struct      dinode          c_dinode;
        int         c_count;
        char        c_addr[BSIZE];
} spcl;

struct  idates {
        char        id_name[16];
        char        id_incno;
        time_t      id_ddate;
};

#define         DUMPOUTFMT          "%-16s %c %s"
        /* for printf */  /* name, incno, ctime(date) */
```

```
#define         DUMPINFMT "%16s %c %[^\n]\n"
        /* inverse for scanf */
```

NTREC is the number of 1024 byte records in a physical tape block. MLEN is the number of bits in a bit map word. MSIZ is the number of bit map words.

The TS_ entries are used in the *c_type* field to indicate what sort of header this is. The types and their meanings are as follows:

TS_TAPE	Tape volume label
TS_INODE	A file or directory follows. The *c_dinode* field is a copy of the disk inode and contains bits telling what sort of file this is.
TS_BITS	A bit map follows. This bit map has a one bit for each inode that was dumped.
TS_ADDR	A subrecord of a file description. See *c_addr* below.
TS_END	End of tape record.
TS_CLRI	A bit map follows. This bit map contains a zero bit for all inodes that were empty on the file system when dumped.
MAGIC	All header records have this number in *c_magic*.
CHECKSUM	Header records checksum to this value.

The fields of the header structure are as follows:

c_type	The type of the header.
c_date	The date the dump was taken.
c_ddate	The date the file system was dumped from.
c_volume	The current volume number of the dump.
c_tapea	The current number of this (1024-byte) record.
c_inumber	The number of the inode being dumped if this is of type TS_INODE.
c_magic	This contains the value MAGIC above, truncated as needed.
c_checksum	This contains whatever value is needed to make the record sum to CHECKSUM.
c_dinode	This is a copy of the inode as it appears on the file system; see *fs*(5).
c_count	The count of characters in *c_addr*.
c_addr	An array of characters describing the blocks of the dumped file. A character is zero if the block associated with that character was not present on the file system, otherwise the character is non-zero. If the block was not present on the file system, no block was dumped; the block will be restored as a hole in the file. If there is not sufficient space in this record to describe all of the blocks in a file, TS_ADDR records will be scattered through the file, each one picking up where the last left off.

Each volume except the last ends with a tapemark (read as an end of file). The last volume ends with a TS_END record and then the tapemark.

The structure *idates* describes an entry in the file */etc/dumpdates* where dump history is kept. The fields of the structure are:

id_name	The dumped filesystem is '/dev/*id_nam*'.
id_incno	The level number of the dump tape; see *dump*(8).
id_ddate	The date of the incremental dump in system format see *types*(5).

FILES

/etc/dumpdates

SEE ALSO

dump(8), restore(8), fs(5), types(5)

NAME

fcntl – file control options

SYNOPSIS

#include <fcntl.h>

DESCRIPTION

The *fcntl*(2) function provides for control over open files. The *include* file describes *requests* and *arguments* to *fcntl* and *open*(2).

```
/*
 * Flag values accessible to open(2) and fcntl(2)
 * (The first three can only be set by open)
 */
#define     O_RDONLY  0
#define     O_WRONLY  1
#define     O_RDWR    2
#define     O_NDELAY  FNDELAY    /* Non-blocking I/O */
#define     O_APPEND  FAPPEND    /* append (writes guaran-
                                    teed at the end) */

#ifndefF_DUPFD
/* fcntl(2) requests */
#define     F_DUPFD   0        /* Duplicate files */
#define     F_GETFD   1        /* Get file flags */
#define     F_SETFD   2        /* Set file flags */
#define     F_GETFL   3        /* Get file flags */
#define     F_SETFL   4        /* Set file flags */
#define     F_GETOWN  5        /* Get owner */
#define     F_SETOWN  6        /* Set owner */
#define     F_GETLK   7        /* Get file lock */
#define     F_SETLK   8        /* Set file lock */
#define     F_SETLKW  9        /* Set file lock and wait */

/* flags for F_GETFL, F_SETFL-- copied from <sys/file.h> */
#define     FNDELAY    00004 /* non-blocking reads */
#define     FAPPEND    00010 /* append on each write */
#define     FASYNC     00100 /* signal pgrp when data ready */

/*
 * file segment locking set data type
 * information passed to system by user
 */
struct flock {
        short   l_type;
        short   l_whence;
        long    l_start;
        long    l_len;  /* len = 0 means until end of file */
        int     l_pid;
};
```

```
/* file segment locking types */
#define        F_RDLCK    01      /* Read lock */
#define        F_WRLCK    02      /* Write lock */
#define        F_UNLCK    03      /* Remove lock(s) */
#endif
```

SEE ALSO

fcntl(2), open(2).

NAME

fs, inode – format of file system volume

SYNOPSIS

#include <sys/types.h>
#include <sys/fs.h>
#include <sys/inode.h>

DESCRIPTION

Every file system storage volume (disk, nine-track tape, for instance) has a common format for certain vital information. Every such volume is divided into a certain number of blocks. The block size is a parameter of the file system. Sectors 0 to 15 on a file system are used to contain primary and secondary bootstrapping programs.

The actual file system begins at sector 16 with the *super block*. The layout of the super block as defined by the include file *<sys/fs.h>* is:

```
#define     FS_MAGIC    0x011954
struct      fs {
    struct      fs *fs_link;    /* linked list of file systems */
    struct      fs *fs_rlink;   /*    used for incore super blocks */
    daddr_t     fs_sblkno;      /* addr of super-block in filesys */
    daddr_t     fs_cblkno;      /* offset of cyl-block in filesys */
    daddr_t     fs_iblkno;      /* offset of inode-blocks in filesys */
    daddr_t     fs_dblkno;      /* offset of first data after cg */
    long        fs_cgoffset;    /* cylinder group offset in cylinder */
    long        fs_cgmask;      /* used to calc mod fs_ntrak */
    time_t      fs_time;        /* last time written */
    long        fs_size;        /* number of blocks in fs */
    long        fs_dsize;       /* number of data blocks in fs */
    long        fs_ncg;         /* number of cylinder groups */
    long        fs_bsize;       /* size of basic blocks in fs */
    long        fs_fsize;       /* size of frag blocks in fs */
    long        fs_frag;        /* number of frags in a block in fs */
/* these are configuration parameters */
    long        fs_minfree;     /* minimum percentage of free blocks */
    long        fs_rotdelay;    /* num of ms for optimal next block */
    long        fs_rps;         /* disk revolutions per second */
/* these fields can be computed from the others */
    long        fs_bmask;       /* "blkoff" calc of blk offsets */
    long        fs_fmask;       /* "fragoff" calc of frag offsets */
    long        fs_bshift;      /* "lblkno" calc of logical blkno */
    long        fs_fshift;      /* "numfrags" calc number of frags */
/* these are configuration parameters */
    long        fs_maxcontig;   /* max number of contiguous blks */
    long        fs_maxbpg;      /* max number of blks per cyl group */
/* these fields can be computed from the others */
    long        fs_fragshift;   /* block to frag shift */
    long        fs_fsbtodb;     /* fsbtodb and dbtofsb shift constant */
    long        fs_sbsize;      /* actual size of super block */
    long        fs_csmask;      /* csum block offset */
    long        fs_csshift;     /* csum block number */
    long        fs_nindir;      /* value of NINDIR */
    long      fs_inopb;      /* value of INOPB */
    long      fs_nspf;       /* value of NSPF */
    long      fs_sparecon[6]; /* reserved for future constants */
/* sizes determined by number of cylinder groups and their sizes */
    daddr_t fs_csaddr;       /* blk addr of cyl grp summary area */
    long      fs_cssize;     /* size of cyl grp summary area */
    long      fs_cgsize;     /* cylinder group size */
/* these fields should be derived from the hardware */
    long      fs_ntrak;      /* tracks per cylinder */
    long      fs_nsect;      /* sectors per track */
    long      fs_spc;        /* sectors per cylinder */
/* this comes from the disk driver partitioning */
    long      fs_ncyl;       /* cylinders in file system */
/* these fields can be computed from the others */
    long      fs_cpg;        /* cylinders per group */
    long      fs_ipg;        /* inodes per group */
    long      fs_fpg;        /* blocks per group * fs_frag */
/* this data must be re-computed after crashes */
    struct    csum fs_cstotal;/* cylinder summary information */
/* these fields are cleared at mount time */
    char      fs_fmod;       /* super block modified flag */
    char      fs_clean;      /* file system is clean flag */
    char      fs_ronly;      /* mounted read-only flag */
    char      fs_flags;      /* currently unused flag */
    char      fs_fsmnt;      /* name mounted on */
              [MAXMNTLEN]
/* these fields retain the current block allocation info */
    long      fs_cgrotor;    /* last cg searched */
    struct    csum *fs_csp;  /* list of fs_cs info buffers */
              [MAXCSBUFS]
    long      fs_cpc;        /* cyl per cycle in postbl */
    short     fs_postbl;     /* head of blocks for each rotation */
              [MAXCPG][NRPOS]
    long      fs_magic;      /* magic number */
    u_char    fs_rotbl[1];   /* list of blocks for each rotation */
/* actually longer */
};
```

Each disk drive contains some number of file systems. A file system consists of a number of cylinder groups. Each cylinder group has inodes and data.

A file system is described by its super-block, which in turn describes the cylinder groups. The super-block is critical data and is replicated in each cylinder group to protect against catastrophic loss. This is done at file system creation time and the critical super-block data does not change, so the copies need not be referenced further unless disaster strikes.

Addresses stored in inodes are capable of addressing fragments of 'blocks'. File system blocks of at most size MAXBSIZE can be optionally broken into 2, 4, or 8 pieces, each of which is addressable; these pieces may be DEV_BSIZE, or some multiple of a DEV_BSIZE unit.

Large files consist of exclusively large data blocks. To avoid undue wasted disk space, the last data block of a small file is allocated as only as many fragments of a large block as are necessary. The file system format retains only a single pointer to such a fragment, which is a piece of a single large block that has been divided. The size of such a fragment is determinable from information in the inode, using the "blksize(fs, ip, lbn)" macro.

The file system records space availability at the fragment level; to determine block availability, aligned fragments are examined.

The root inode is the root of the file system. Inode 0 can't be used for normal purposes and historically bad blocks were linked to inode 1, thus the root inode is 2 (inode 1 is no longer used for this purpose, however numerous dump tapes make this assumption, so we are stuck with it). The *lost+found* directory is given the next available inode when it is initially created by *mkfs*.

fs_minfree gives the minimum acceptable percentage of file system blocks which may be free. If the freelist drops below this level only the super-user may continue to allocate blocks. This may be set to 0 if no reserve of free blocks is deemed necessary, however severe performance degradations will be observed if the file system is run at greater than 90% full; thus the default value of *fs_minfree* is 10%.

Empirically the best trade-off between block fragmentation and overall disk utilization at a loading of 90% comes with a fragmentation of 4, thus the default fragment size is a fourth of the block size.

Cylinder group related limits: Each cylinder keeps track of the availability of blocks at different rotational positions, so that sequential blocks can be laid out with minimum rotational latency. NRPOS is the number of rotational positions which are distinguished. With NRPOS 8 the resolution of the summary information is 2ms for a typical 3600 rpm drive.

fs_rotdelay gives the minimum number of milliseconds to initiate another disk transfer on the same cylinder. It is used in determining the rotationally optimal layout for disk blocks within a file; the default value for *fs_rotdelay* is 2ms.

Each file system has a statically allocated number of inodes. An inode is allocated for each NBPI bytes of disk space. The inode allocation strategy is extremely conservative.

MAXIPG bounds the number of inodes per cylinder group, and is needed only to keep the structure simpler by having the only a single variable size element (the free bit map).

N.B.: MAXIPG must be a multiple of INOPB(fs).

MINBSIZE is the smallest allowable block size. With a MINBSIZE of 4096 it is possible to create files of size 2^32 with only two levels of indirection. MINBSIZE must be big enough to hold a cylinder group block, thus changes to (struct cg) must keep its size within MINBSIZE. MAXCPG is limited only to dimension an array in (struct cg); it can be made larger as long as that structure's size remains within the bounds dictated by MINBSIZE. Note that super blocks are never more than size SBSIZE.

The path name on which the file system is mounted is maintained in *fs_fsmnt*. MAXMNTLEN defines the amount of space allocated in the super block for this name. The limit on the amount of summary information per file system is defined by MAXCSBUFS. It is currently parameterized for a maximum of two million cylinders.

Per cylinder group information is summarized in blocks allocated from the first cylinder group's data blocks. These blocks are read in from *fs_csaddr* (size *fs_cssize*) in addition to the super block.

N.B.: sizeof (struct csum) must be a power of two in order for the "fs_cs" macro to work.

Super block for a file system: MAXBPC bounds the size of the rotational layout tables and is limited by the fact that the super block is of size SBSIZE. The size of these tables is **inversely** proportional to the block size of the file system. The size of the tables is increased when sector sizes are not powers of two, as this increases the number of cylinders included before the rotational pattern repeats (*fs_cpc*). The size of the rotational layout tables is derived from the number of bytes remaining in (struct fs).

MAXBPG bounds the number of blocks of data per cylinder group, and is limited by the fact that cylinder groups are at most one block. The size of the free block table is derived from the size of blocks and the number of remaining bytes in the cylinder group structure (struct cg).

Inode: The inode is the focus of all file activity in the UNIX file system. There is a unique inode allocated for each active file, each current directory, each mounted-on file, text file, and the root. An inode is 'named' by its device/i-number pair. For further information, see the include file <*sys/inode.h*>.

NAME

fstab – static information about the filesystems

SYNOPSIS

#include <fstab.h>

DESCRIPTION

The file */etc/fstab* contains descriptive information about the various file systems. */etc/fstab* is only *read* by programs, and not written; it is the duty of the system administrator to properly create and maintain this file. The order of records in */etc/fstab* is important because *fsck, mount,* and *umount* sequentially iterate through */etc/fstab* doing their thing.

The special file name is the **block** special file name, and not the character special file name. If a program needs the character special file name, the program must create it by appending a "r" after the last "/" in the special file name.

If *fs_type* is "rw" or "ro" then the file system whose name is given in the *fs_file* field is normally mounted read-write or read-only on the specified special file. If *fs_type* is "rq", then the file system is normally mounted read-write with disk quotas enabled. The *fs_freq* field is used for these file systems by the *dump*(8) command to determine which file systems need to be dumped. The *fs_passno* field is used by the *fsck*(8) program to determine the order in which file system checks are done at reboot time. The root file system should be specified with a *fs_passno* of 1, and other file systems should have larger numbers. File systems within a drive should have distinct numbers, but file systems on different drives can be checked on the same pass to utilize parallelism available in the hardware.

If *fs_type* is "sw" then the special file is made available as a piece of swap space by the *swapon*(8) command at the end of the system reboot procedure. The fields other than *fs_spec* and *fs_type* are not used in this case.

If *fs_type* is "rq" then at boot time the file system is automatically processed by the *quotacheck*(8) command and disk quotas are then enabled with *quotaon*(8). File system quotas are maintained in a file "quotas", which is located at the root of the associated file system.

If *fs_type* is specified as "xx" the entry is ignored. This is useful to show disk partitions which are currently not used.

```
#define     FSTAB_RW  "rw"   /* read-write device */
#define     FSTAB_RO  "ro"   /* read-only device */
#define     FSTAB_RQ  "rq"   /* read-write with quotas */
#define     FSTAB_SW  "sw"   /* swap device */
#define     FSTAB_XX  "xx"   /* ignore totally */

struct fstab {
   char    *fs_spec;/* block special device name */
   char    *fs_file;/* file system path prefix */
   char    *fs_type;/* rw,ro,sw or xx */
   int     fs_freq;/* dump frequency, in days */
   int     fs_passno;/* pass number on parallel dump */
};
```

The proper way to read records from */etc/fstab* is to use the routines getfsent(), getfsspec(), getfstype(), and getfsfile().

FILES

/etc/fstab

SEE ALSO

getfsent(3X)

NAME
gettytab – terminal configuration data base

SYNOPSIS
/etc/gettytab

DESCRIPTION
Gettytab is a simplified version of the *termcap*(5) data base used to describe terminal lines. The initial terminal login process *getty*(8) accesses the *gettytab* file each time it starts, allowing simpler reconfiguration of terminal characteristics. Each entry in the data base is used to describe one class of terminals.

There is a default terminal class, *default*, that is used to set global defaults for all other classes. (That is, the *default* entry is read, then the entry for the class required is used to override particular settings.)

CAPABILITIES
Refer to *termcap*(5) for a description of the file layout. The *default* column below lists defaults obtained if there is no entry in the table obtained, nor one in the special *default* table.

Name	Type	Default	Description
ap	bool	false	terminal uses any parity
bd	num	0	backspace delay
bk	str	0377	alternate end of line character (input break)
cb	bool	false	use crt backspace mode
cd	num	0	carriage-return delay
ce	bool	false	use crt erase algorithm
ck	bool	false	use crt kill algorithm
cl	str	NULL	screen clear sequence
co	bool	false	console - add \n after login prompt
ds	str	^Y	delayed suspend character
ec	bool	false	leave echo OFF
ep	bool	false	terminal uses even parity
er	str	^?	erase character
et	str	^D	end of text (EOF) character
ev	str	NULL	initial enviroment
f0	num	unused	tty mode flags to write messages
f1	num	unused	tty mode flags to read login name
f2	num	unused	tty mode flags to leave terminal as
fd	num	0	form-feed (vertical motion) delay
fl	str	^O	output flush character
hc	bool	false	do NOT hangup line on last close
he	str	NULL	hostname editing string
hn	str	hostname	hostname
ht	bool	false	terminal has real tabs
ig	bool	false	ignore garbage characters in login name
im	str	NULL	initial (banner) message
in	str	^C	interrupt character
is	num	unused	input speed
kl	str	^U	kill character
lc	bool	false	terminal has lower case
lm	str	login:	login prompt

ln	str	^V	"literal next" character
lo	str	/bin/login	program to exec when name obtained
nd	num	0	newline (line-feed) delay
nl	bool	false	terminal has (or might have) a newline character
nx	str	default	next table (for auto speed selection)
op	bool	false	terminal uses odd parity
os	num	unused	output speed
pc	str	\0	pad character
pe	bool	false	use printer (hard copy) erase algorithm
pf	num	0	delay between first prompt and following flush (seconds)
ps	bool	false	line connected to a MICOM port selector
qu	str	^\	quit character
rp	str	^R	line retype character
rw	bool	false	do NOT use raw for input, use cbreak
sp	num	unused	line speed (input and output)
su	str	^Z	suspend character
tc	str	none	table continuation
to	num	0	timeout (seconds)
tt	str	NULL	terminal type (for enviroment)
ub	bool	false	do unbuffered output (of prompts etc)
uc	bool	false	terminal is known upper case only
we	str	^W	word erase character
xc	bool	false	do NOT echo control chars as ^X
xf	str	^S	XOFF (stop output) character
xn	str	^Q	XON (start output) character

If no line speed is specified, speed will not be altered from that which prevails when getty is entered. Specifying an input or output speed will override line speed for stated direction only.

Terminal modes to be used for the output of the message, for input of the login name, and to leave the terminal set as upon completion, are derived from the boolean flags specified. If the derivation should prove inadequate, any (or all) of these three may be overriden with one of the **f0**, **f1**, or **f2** numeric specifications, which can be used to specify (usually in octal, with a leading '0') the exact values of the flags. Local (new tty) flags are set in the top 16 bits of this (32 bit) value.

Should *getty* receive a null character (presumed to indicate a line break) it will restart using the table indicated by the **nx** entry. If there is none, it will re-use its original table.

Delays are specified in milliseconds, the nearest possible delay available in the tty driver will be used. Should greater certainty be desired, delays with values 0, 1, 2, and 3 are interpreted as choosing that particular delay algorithm from the driver.

The **cl** screen clear string may be preceded by a (decimal) number of milliseconds of delay required (a la termcap). This delay is simulated by repeated use of the pad character **pc**.

The initial message, and login message, **im** and **lm** may include the character sequence **%h** to obtain the hostname. (**%%** obtains a single '%' character.) The hostname is normally obtained from the system, but may be set by the **hn** table entry. In either case it may be edited with **he**. The **he** string is a sequence of characters, each character that is neither '@' nor '#' is copied into the final hostname. A '@' in the **he** string, causes one character from the real hostname to be copied to the final hostname. A '#' in the **he** string, causes the next character of the real hostname to be skipped. Surplus '@' and '#' characters are ignored.

When getty execs the login process, given in the **lo** string (usually "/bin/login"), it will have set the enviroment to include the terminal type, as indicated by the **tt** string (if it exists). The **ev** string, can be used to enter additional data into the environment. It is a list of comma separated strings, each of which will presumably be of the form *name=value*.

If a non-zero timeout is specified, with **to**, then getty will exit within the indicated number of seconds, either having received a login name and passed control to *login*, or having received an alarm signal, and exited. This may be useful to hangup dial in lines.

Output from *getty* is even parity unless **op** is specified. **Op** may be specified with **ap** to allow any parity on input, but generate odd parity output. Note: this only applies while getty is being run, terminal driver limitations prevent a more complete implementation. *Getty* does not check parity of input characters in *RAW* mode.

SEE ALSO

termcap(5), getty(8).

BUGS

Some ignorant peasants insist on changing the default special characters, so it is wise to always specify (at least) the erase, kill, and interrupt characters in the **default** table. In **all** cases, '#' or '^H' typed in a login name will be treated as an erase character, and '@' will be treated as a kill character.

The delay stuff is a real crock. Apart form its general lack of flexibility, some of the delay algorithms are not implemented. The terminal driver should support sane delay settings.

Currently *login*(1) stomps on the environment, so there is no point setting it in *gettytab*.

The **he** capability is stupid.

Termcap format is horrid, something more rational should have been chosen.

NAME
group – group file

DESCRIPTION
Group contains for each group the following information:

group name
encrypted password
numerical group ID
a comma separated list of all users allowed in the group

This is an ASCII file. The fields are separated by colons; Each group is separated from the next by a new-line. If the password field is null, no password is demanded.

This file resides in directory /etc. Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical group ID's to names.

FILES
/etc/group

SEE ALSO
setgroups(2), initgroups(3X), crypt(3), passwd(1), passwd(5)

BUGS
The *passwd*(1) command won't change the passwords.

NAME

hosts – host name data base

DESCRIPTION

The *hosts* file contains information regarding the known hosts on the DARPA Internet. For each host a single line should be present with the following information:

official host name
Internet address
aliases

Items are separated by any number of blanks and/or tab characters. A "#" indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file. This file is normally created from the official host data base maintained at the Network Information Control Center (NIC), though local changes may be required to bring it up to date regarding unofficial aliases and/or unknown hosts.

Network addresses are specified in the conventional "." notation using the *inet_addr*() routine from the Internet address manipulation library, *inet*(3N). Host names may contain any printable character other than a field delimiter, newline, or comment character.

FILES

/etc/hosts

SEE ALSO

gethostent(3N)

BUGS

A name server should be used instead of a static file. A binary indexed file format should be available for fast access.

NAME

mtab – mounted file system table

SYNOPSIS

#include <fstab.h>
#include <mtab.h>

DESCRIPTION

Mtab resides in directory */etc* and contains a table of devices mounted by the *mount* command. *Umount* removes entries.

The table is a series of *mtab* structures, as defined in <mtab.h>. Each entry contains the null-padded name of the place where the special file is mounted, the null-padded name of the special file, and a type field, one of those defined in *<fstab.h>*. The special file has all its directories stripped away; that is, everything through the last '/' is thrown away. The type field indicates if the file system is mounted read-only, read-write, or read-write with disk quotas enabled.

This table is present only so people can look at it. It does not matter to *mount* if there are duplicated entries nor to *umount* if a name cannot be found.

FILES

/etc/mtab

SEE ALSO

mount(8)

NAME

mttys – Multi-Link partition information

DESCRIPTION

The file */etc/mttys* is read by the *dosc* program and specifies the maximum number of Multi-Link partitions that can be active. There is currently only one line in the file, which contains the decimal number of partitions. Currently the number may range from 1 to 8.

FILES

/etc/mttys

SEE ALSO

dosc(1)

NAME

networks – network name data base

DESCRIPTION

The *networks* file contains information regarding the known networks which comprise the DARPA Internet. For each network a single line should be present with the following information:

official network name
network number
aliases

Items are separated by any number of blanks and/or tab characters. A "#" indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file. This file is normally created from the official network data base maintained at the Network Information Control Center (NIC), though local changes may be required to bring it up to date regarding unofficial aliases and/or unknown networks.

Network number may be specified in the conventional "." notation using the *inet_network*() routine from the Internet address manipulation library, *inet*(3N). Network names may contain any printable character other than a field delimiter, newline, or comment character.

FILES

/etc/networks

SEE ALSO

getnetent(3N)

BUGS

A name server should be used instead of a static file. A binary indexed file format should be available for fast access.

NAME

passwd – password file

DESCRIPTION

Passwd contains for each user the following information:

name (login name, contains no upper case)
encrypted password
numerical user ID
numerical group ID
user's real name, office, extension, home phone.
initial working directory
program to use as Shell

The name may contain '&', meaning insert the login name. This information is set by the *chfn*(1) command and used by the *finger*(1) command.

This is an ASCII file. Each field within each user's entry is separated from the next by a colon. Each user is separated from the next by a new-line. If the password field is null, no password is demanded; if the Shell field is null, then */bin/sh* is used.

This file resides in directory /etc. Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical user ID's to names.

Appropriate precautions must be taken to lock the file against changes if it is to be edited with a text editor; *vipw*(8) does the necessary locking.

FILES

/etc/passwd

SEE ALSO

getpwent(3), login(1), crypt(3), passwd(1), group(5), chfn(1), finger(1), vipw(8), adduser(8)

BUGS

A binary indexed file format should be available for fast access.

User information (name, office, etc.) should be stored elsewhere.

NAME

phones – remote host phone number data base

DESCRIPTION

The file /etc/phones contains the system-wide private phone numbers for the *tip*(1C) program. This file is normally unreadable, and so may contain privileged information. The format of the file is a series of lines of the form: <system-name>[\t]*<phone-number>. The system name is one of those defined in the *remote*(5) file and the phone number is constructed from [0123456789-=*%]. The "=" and "*" characters are indicators to the auto call units to pause and wait for a second dial tone (when going through an exchange). The "=" is required by the DF02-AC and the "*" is required by the BIZCOMP 1030.

Only one phone number per line is permitted. However, if more than one line in the file contains the same system name *tip*(1C) will attempt to dial each one in turn, until it establishes a connection.

FILES

/etc/phones

SEE ALSO

tip(1C), remote(5)

NAME

plot – graphics interface

DESCRIPTION

Files of this format are produced by routines described in *plot*(3X), and are interpreted for various devices by commands described in *plot*(1G). A graphics file is a stream of plotting instructions. Each instruction consists of an ASCII letter usually followed by bytes of binary information. The instructions are executed in order. A point is designated by four bytes representing the x and y values; each value is a signed integer. The last designated point in an **l**, **m**, **n**, or **p** instruction becomes the 'current point' for the next instruction.

Each of the following descriptions begins with the name of the corresponding routine in *plot*(3X).

m move: The next four bytes give a new current point.

n cont: Draw a line from the current point to the point given by the next four bytes. See *plot*(1G).

p point: Plot the point given by the next four bytes.

l line: Draw a line from the point given by the next four bytes to the point given by the following four bytes.

t label: Place the following ASCII string so that its first character falls on the current point. The string is terminated by a newline.

a arc: The first four bytes give the center, the next four give the starting point, and the last four give the end point of a circular arc. The least significant coordinate of the end point is used only to determine the quadrant. The arc is drawn counter-clockwise.

c circle: The first four bytes give the center of the circle, the next two the radius.

e erase: Start another frame of output.

f linemod: Take the following string, up to a newline, as the style for drawing further lines. The styles are 'dotted,' 'solid,' 'longdashed,' 'shortdashed,' and 'dotdashed.' Effective only in *plot 4014* and *plot ver.*

s space: The next four bytes give the lower left corner of the plotting area; the following four give the upper right corner. The plot will be magnified or reduced to fit the device as closely as possible.

Space settings that exactly fill the plotting area with unity scaling appear below for devices supported by the filters of *plot*(1G). The upper limit is just outside the plotting area. In every case the plotting area is taken to be square; points outside may be displayable on devices whose face isn't square.

4014	space(0, 0, 3120, 3120);
ver	space(0, 0, 2048, 2048);
300, 300s	space(0, 0, 4096, 4096);
450	space(0, 0, 4096, 4096);

SEE ALSO

plot(1G), plot(3X), graph(1G)

NAME

printcap – printer capability data base

SYNOPSIS

/etc/printcap

DESCRIPTION

Printcap is a simplified version of the *termcap*(5) data base used to describe line printers. The spooling system accesses the *printcap* file every time it is used, allowing dynamic addition and deletion of printers. Each entry in the data base is used to describe one printer. This data base may not be substituted for, as is possible for *termcap*, because it may allow accounting to be bypassed.

The default printer is normally *lp*, though the environment variable PRINTER may be used to override this. Each spooling utility supports an option, **–P***printer,* to allow explicit naming of a destination printer.

Refer to the *4.2BSD Line Printer Spooler Manual* for a complete discussion on how setup the database for a given printer.

CAPABILITIES

Refer to *termcap* for a description of the file layout.

Name	**Type**	**Default**	**Description**
af	str	NULL	name of accounting file
br	num	none	if lp is a tty, set the baud rate (ioctl call)
cf	str	NULL	cifplot data filter
df	str	NULL	tex data filter (DVI format)
fc	num	0	if lp is a tty, clear flag bits (sgtty.h)
ff	str	"\f"	string to send for a form feed
fo	bool	false	print a form feed when device is opened
fs	num	0	like 'fc' but set bits
gf	str	NULL	graph data filter (plot (3X) format)
ic	bool	false	driver supports (non standard) ioctl to indent printout
if	str	NULL	name of text filter which does accounting
lf	str	"/dev/console"	error logging file name
lo	str	"lock"	name of lock file
lp	str	"/dev/lp"	device name to open for output
mx	num	1000	maximum file size (in BUFSIZ blocks), zero = unlimited
nd	str	NULL	next directory for list of queues (unimplemented)
nf	str	NULL	ditroff data filter (device independent troff)
of	str	NULL	name of output filtering program
pl	num	66	page length (in lines)
pw	num	132	page width (in characters)
px	num	0	page width in pixels (horizontal)
py	num	0	page length in pixels (vertical)
rf	str	NULL	filter for printing FORTRAN style text files
rm	str	NULL	machine name for remote printer
rp	str	"lp"	remote printer name argument
rs	bool	false	restrict remote users to those with local accounts
rw	bool	false	open the printer device for reading and writing
sb	bool	false	short banner (one line only)
sc	bool	false	suppress multiple copies

sd	str	"/usr/spool/lpd"	spool directory
sf	bool	false	suppress form feeds
sh	bool	false	suppress printing of burst page header
st	str	"status"	status file name
tf	str	NULL	troff data filter (cat phototypesetter)
tr	str	NULL	trailer string to print when queue empties
vf	str	NULL	raster image filter
xc	num	0	if lp is a tty, clear local mode bits (tty (4))
xs	num	0	like 'xc' but set bits

Error messages sent to the console have a carriage return and a line feed appended to them, rather than just a line feed.

If the local line printer driver supports indentation, the daemon must understand how to invoke it.

SEE ALSO

termcap(5), lpc(8), lpd(8), pac(8), lpr(1), lpq(1), lprm(1)
4.2BSD Line Printer Spooler Manual

NAME

protocols – protocol name data base

DESCRIPTION

The *protocols* file contains information regarding the known protocols used in the DARPA Internet. For each protocol a single line should be present with the following information:

official protocol name
protocol number
aliases

Items are separated by any number of blanks and/or tab characters. A "#" indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

Protocol names may contain any printable character other than a field delimiter, newline, or comment character.

FILES

/etc/protocols

SEE ALSO

getprotoent(3N)

BUGS

A name server should be used instead of a static file. A binary indexed file format should be available for fast access.

NAME

rcsfile – format of RCS file

DESCRIPTION

An RCS file is an ASCII file. Its contents is described by the grammar below. The text is free format, i.e., spaces, tabs and new lines have no significance except in strings. Strings are enclosed by '@'. If a string contains a '@', it must be doubled.

The meta syntax uses the following conventions: '|' (bar) separates alternatives; '{' and '}' enclose optinal phrases; '{' and '}*' enclose phrases that may be repeated zero or more times; '{' and '}+' enclose phrases that must appear at least once and may be repeated; '<' and '>' enclose nonterminals.

```
<rcstext>      ::=   <admin> {<delta>}* <desc> {<deltatext>}*

<admin>        ::=   head        {<num>};
                     access      {<id>}*;
                     symbols     {<id> : <num>}*;
                     locks       {<id> : <num>}*;
                     comment     {<string>};

<delta>        ::=   <num>
                     date        <num>;
                     author      <id>;
                     state       {<id>};
                     branches    {<num>}*;
                     next        {<num>};

<desc>         ::=   desc        <string>

<deltatext>    ::=   <num>
                     log         <string>
                     text        <string>


<num>          ::=   {<digit>{.}}+

<digit>        ::=   0 | 1 | ... | 9

<id>           ::=   <letter>{<idchar>}*

<letter>       ::=   A | B | ... | Z | a | b | ... | z

<idchar>       ::=   Any printing ASCII character except space,
                     tab, carriage return, new line, and <special>.

<special>      ::=   ; | : | , | @

<string>       ::=   @{any ASCII character, with '@' doubled}*@
```

Identifiers are case sensitive. Keywords are in lower case only. The sets of keywords and identifiers may overlap.

The <delta> nodes form a tree. All nodes whose numbers consist of a single pair (e.g., 2.3, 2.1, 1.3, etc.) are on the "trunk", and are linked through the "next" field in order of decreasing numbers. The "head" field in the <admin> node points to the head of that sequence (i.e., contains the highest pair).

All <delta> nodes whose numbers consist of 2n fields (n≥2) (e.g., 3.1.1.1, 2.1.2.2, etc.) are linked as follows. All nodes whose first (2n)-1 number fields are identical are linked through the "next" field in order of increasing numbers. For each such sequence, the <delta> node whose number is identical to the first 2(n-1) number fields of the deltas on that sequence is called the branchpoint. The "branches" field of a node contains a list of the numbers of the first nodes of all sequences for which it is a branchpoint. This list is ordered in increasing numbers.

Example:

```
                              Head
                                |
                                |
                                v
                            ---------
       / \          / \     |       |     / \           / \
      /   \        /   \    |  2.1  |    /   \         /   \
     /     \      /     \   |       |   /     \       /     \
    /1.2.1.3\    /1.3.1.1\  |       |  /1.2.2.2\     /1.2.2.1.1.1\
    ---------    ---------  ---------  ---------    -------------
        ^            ^          |          ^              ^
        |            |          |          |              |
        |            |          v          |              |
       / \           |      ---------     / \             |
      /   \          |      \  1.3  /    /   \            |
     /     \         --------\     /    /     \------------
    /1.2.1.1\                 \   /    /1.2.2.1\
    ---------                  \ /     ---------
        ^                       |          ^
        |                       |          |
        |                       v          |
        |                   ---------      |
        |                   \  1.2  /      |
        ---------------------\     /--------
                              \   /
                               \ /
                                |
                                |
                                v
                            ---------
                            \  1.1  /
                             \     /
                              \   /
                               \ /
```

Fig. 1: A revision tree

IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN, 47907.
Revision Number: 3.0 ; Release Date: 82/11/18 .
Copyright © 1982 by Walter F. Tichy.

SEE ALSO

ci (1), co (1), ident (1), rcs (1), rcsdiff (1), rcsintro (1), rcsmerge (1), rlog (1), sccstorcs (8).

NAME

remote – remote host description file

DESCRIPTION

The systems known by *tip*(1C) and their attributes are stored in an ASCII file which is structured somewhat like the *termcap*(5) file. Each line in the file provides a description for a single *system*. Fields are separated by a colon (":"). Lines ending in a \ character with an immediately following newline are continued on the next line.

The first entry is the name(s) of the host system. If there is more than one name for a system, the names are separated by vertical bars. After the name of the system comes the fields of the description. A field name followed by an '=' sign indicates a string value follows. A field name followed by a '#' sign indicates a following numeric value.

Entries named "tip*" and "cu*" are used as default entries by *tip*, and the *cu* interface to *tip*, as follows. When *tip* is invoked with only a phone number, it looks for an entry of the form "tip300", where 300 is the baud rate with which the connection is to be made. When the *cu* interface is used, entries of the form "cu300" are used.

CAPABILITIES

Capabilities are either strings (str), numbers (num), or boolean flags (bool). A string capability is specified by *capability=value*; e.g. "dv=/dev/harris". A numeric capability is specified by *capability#value*; e.g. "xa#99". A boolean capability is specified by simply listing the capability.

at	(str) Auto call unit type.
br	(num) The baud rate used in establishing a connection to the remote host. This is a decimal number. The default baud rate is 300 baud.
cm	(str) An initial connection message to be sent to the remote host. For example, if a host is reached through port selector, this might be set to the appropriate sequence required to switch to the host.
cu	(str) Call unit if making a phone call. Default is the same as the 'dv' field.
di	(str) Disconnect message sent to the host when a disconnect is requested by the user.
du	(bool) This host is on a dial-up line.
dv	(str) UNIX device(s) to open to establish a connection. If this file refers to a terminal line, *tip*(1C) attempts to perform an exclusive open on the device to insure only one user at a time has access to the port.
el	(str) Characters marking an end-of-line. The default is NULL. '~' escapes are only recognized by *tip* after one of the characters in 'el', or after a carriage-return.
fs	(str) Frame size for transfers. The default frame size is equal to BUFSIZ.
hd	(bool) The host uses half-duplex communication, local echo should be performed.
ie	(str) Input end-of-file marks. The default is NULL.
oe	(str) Output end-of-file string. The default is NULL. When *tip* is transferring a file, this string is sent at end-of-file.
pa	(str) The type of parity to use when sending data to the host. This may be one of "even", "odd", "none", "zero" (always set bit 8 to zero), "one" (always set bit 8 to 1). The default is even parity.
pn	(str) Telephone number(s) for this host. If the telephone number field contains an @ sign, *tip* searches the file */etc/phones* file for a list of telephone numbers; c.f.

NAME
services – service name data base

DESCRIPTION
The *services* file contains information regarding the known services available in the DARPA Internet. For each service a single line should be present with the following information:

official service name
port number
protocol name
aliases

Items are separated by any number of blanks and/or tab characters. The port number and protocol name are considered a single *item*; a "/" is used to separate the port and protocol (e.g. "512/tcp"). A "#" indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

Service names may contain any printable character other than a field delimiter, newline, or comment character.

FILES
/etc/services

SEE ALSO
getservent(3N)

BUGS
A name server should be used instead of a static file. A binary indexed file format should be available for fast access.

NAME

stab – symbol table types

SYNOPSIS

#include <stab.h>

DESCRIPTION

Stab.h defines some values of the n_type field of the symbol table of a.out files. These are the types for permanent symbols (i.e. not local labels, etc.) used by the debugger *dbx* and the Berkeley Pascal compiler *pc*(1). Symbol table entries can be produced by the *.stabs* assembler directive. This allows one to specify a double-quote delimited name, a symbol type, one char and one short of information about the symbol, and an unsigned long (usually an address). To avoid having to produce an explicit label for the address field, the *.stabd* directive can be used to implicitly address the current location. If no name is needed, symbol table entries can be generated using the *.stabn* directive. The loader promises to preserve the order of symbol table entries produced by *.stab* directives. As described in *a.out*(5), an element of the symbol table consists of the following structure:

```
/*
 * Format of a symbol table entry.
 */
struct nlist {
        union {
                char  *n_name; /* for use when in-core */
                long  n_strx;  /* index into file string table */
        } n_un;
        unsigned char n_type;  /* type flag */
        char          n_other; /* unused */
        short         n_desc;  /* see struct desc, below */
        unsigned n_value;      /* address or offset or line */
};
```

The low bits of the n_type field are used to place a symbol into at most one segment, according to the following masks, defined in <*a.out.h*>. A symbol can be in none of these segments by having none of these segment bits set.

```
/*
 * Simple values for n_type.
 */
#define N_UNDF   0x0  /* undefined */
#define N_ABS    0x2  /* absolute */
#define N_TEXT   0x4  /* text */
#define N_DATA   0x6  /* data */
#define N_BSS    0x8  /* bss */

#define N_EXT    01   /* external bit, or'ed in */
```

The n_value field of a symbol is relocated by the linker, *ld*(1) as an address within the appropriate segment. N_value fields of symbols not in any segment are unchanged by the linker. In addition, the linker will discard certain symbols, according to rules of its own, unless the n_type field has one of the following bits set:

```
/*
 * Other permanent symbol table entries have some of the
 * N_STAB bits set. These are given in <stab.h>
 */
#define N_STAB    0xe0/* if any of these bits set, don't discard */
```

This allows up to 112 (7 * 16) symbol types, split between the various segments. Some of these have already been claimed. The symbolic debugger, *dbx*, uses the following n_type values:

```
#define N_GSYM   0x20 /* global symbol: name,,0,type,0 */
#define N_FNAME  0x22 /* procedure name (f77 kludge): name,,0 */
#define N_FUN    0x24 /* procedure: name,,0,linenumber,address */
#define N_STSYM  0x26 /* static symbol: name,,0,type,address */
#define N_LCSYM  0x28 /* .lcomm symbol: name,,0,type,address */
#define N_RSYM   0x40 /* register sym: name,,0,type,register */
#define N_SLINE  0x44 /* src line: 0,,0,linenumber,address */
#define N_SSYM   0x60 /* structure elt: name,,0,type,struct_offset */
#define N_SO     0x64 /* source file name: name,,0,0,address */
#define N_LSYM   0x80 /* local sym: name,,0,type,offset */
#define N_SOL    0x84 /* #included file name: name,,0,0,address */
#define N_PSYM   0xa0 /* parameter: name,,0,type,offset */
#define N_ENTRY  0xa4 /* alternate entry: name,linenumber,address */
#define N_LBRAC  0xc0 /* left bracket: 0,,0,nesting level,address */
#define N_RBRAC  0xe0 /* right bracket: 0,,0,nesting level,address */
#define N_BCOMM0xe2 /* begin common: name,, */
#define N_ECOMM0xe4 /* end common: name,, */
#define N_ECOML  0xe8 /* end common (local name): ,,address */
#define N_LENG   0xfe /* second stab entry with length information */
```

where the comments give *dbx* conventional use for *.stabs* and the n_name, n_other, n_desc, and n_value fields of the given n_type. *Dbx* uses the n_desc field to hold a type specifier in the form used by the Portable C Compiler, *cc*(1), in which a base type is qualified in the following structure:

```
struct desc {
        short q6:2,
              q5:2,
              q4:2,
              q3:2,
              q2:2,
              q1:2,
              basic:4;
};
```

There are four qualifications, with q1 the most significant and q6 the least significant:

0	none
1	pointer
2	function
3	array

The sixteen basic types are assigned as follows:

0	undefined
1	function argument
2	character
3	short
4	int
5	long
6	float
7	double
8	structure
9	union
10	enumeration
11	member of enumeration
12	unsigned character
13	unsigned short
14	unsigned int
15	unsigned long

The Berkeley Pascal compiler, *pc*(1), uses the following n_type value:

```
#define  N_PC  0x30  /* global pascal symbol: name,,0,subtype,line */
```

and uses the following subtypes to do type checking across separately compiled files:

1	source file name
2	included file name
3	global label
4	global constant
5	global type
6	global variable
7	global function
8	global procedure
9	external function
10	external procedure
11	library variable
12	library routine

SEE ALSO

as(1), ld(1), dbx(1), a.out(5)

BUGS

Dbx assumes that a symbol of type N_GSYM with name *name* is located at address _*name*.

More basic types are needed.

NAME

tar – tape archive file format

DESCRIPTION

Tar, (the tape archive command) dumps several files into one, in a medium suitable for transportation.

A "tar tape" or file is a series of blocks. Each block is of size TBLOCK. A file on the tape is represented by a header block which describes the file, followed by zero or more blocks which give the contents of the file. At the end of the tape are two blocks filled with binary zeros, as an end-of-file indicator.

The blocks are grouped for physical I/O operations. Each group of *n* blocks (where *n* is set by the **b** keyletter on the *tar*(1) command line — default is 20 blocks) is written with a single system call; on nine-track tapes, the result of this write is a single tape record. The last group is always written at the full size, so blocks after the two zero blocks contain random data. On reading, the specified or default group size is used for the first read, but if that read returns less than a full tape block, the reduced block size is used for further reads.

The header block looks like:

```
#define TBLOCK    512
#define NAMSIZ    100

union hblock {
        char dummy[TBLOCK];
        struct header {
                char name[NAMSIZ];
                char mode[8];
                char uid[8];
                char gid[8];
                char size[12];
                char mtime[12];
                char chksum[8];
                char linkflag;
                char linkname[NAMSIZ];
        } dbuf;
};
```

Name is a null-terminated string. The other fields are zero-filled octal numbers in ASCII. Each field (of width w) contains w-2 digits, a space, and a null, except *size* and *mtime*, which do not contain the trailing null. *Name* is the name of the file, as specified on the *tar* command line. Files dumped because they were in a directory which was named in the command line have the directory name as prefix and */filename* as suffix. *Mode* is the file mode, with the top bit masked off. *Uid* and *gid* are the user and group numbers which own the file. *Size* is the size of the file in bytes. Links and symbolic links are dumped with this field specified as zero. *Mtime* is the modification time of the file at the time it was dumped. *Chksum* is a decimal ASCII value which represents the sum of all the bytes in the header block. When calculating the checksum, the *chksum* field is treated as if it were all blanks. *Linkflag* is ASCII '0' if the file is "normal" or a special file, ASCII '1' if it is an hard link, and ASCII '2' if it is a symbolic link. The name linked-to, if any, is in *linkname,* with a trailing null. Unused fields of the header are binary zeros (and are included in the checksum).

The first time a given i-node number is dumped, it is dumped as a regular file. The second and subsequent times, it is dumped as a link instead. Upon retrieval, if a link entry is retrieved, but not the file it was linked to, an error message is printed and the tape must be manually re-scanned to retrieve the linked-to file.

The encoding of the header is designed to be portable across machines.

SEE ALSO

tar(1)

BUGS

Names or linknames longer than NAMSIZ produce error reports and cannot be dumped.

NAME

termcap – terminal capability data base

SYNOPSIS

/etc/termcap

DESCRIPTION

Termcap is a data base describing terminals, used, *e.g.*, by *vi*(1) and *curses*(3X). Terminals are described in *termcap* by giving a set of capabilities which they have, and by describing how operations are performed. Padding requirements and initialization sequences are included in *termcap*.

Entries in *termcap* consist of a number of ':' separated fields. The first entry for each terminal gives the names which are known for the terminal, separated by '|' characters. The first name is always 2 characters long and is used by older version 6 systems which store the terminal type in a 16 bit word in a systemwide data base. The second name given is the most common abbreviation for the terminal, and the last name given should be a long name fully identifying the terminal. The second name should contain no blanks; the last name may well contain blanks for readability.

CAPABILITIES

(P) indicates padding may be specified
(P*) indicates that padding may be based on no. lines affected

Name	**Type**	**Pad?**	**Description**
ae	str	(P)	End alternate character set
al	str	(P*)	Add new blank line
am	bool		Terminal has automatic margins
as	str	(P)	Start alternate character set
bc	str		Backspace if not **^H**
bs	bool		Terminal can backspace with **^H**
bt	str	(P)	Back tab
bw	bool		Backspace wraps from column 0 to last column
CC	str		Command character in prototype if terminal settable
cd	str	(P*)	Clear to end of display
ce	str	(P)	Clear to end of line
ch	str	(P)	Like cm but horizontal motion only, line stays same
cl	str	(P*)	Clear screen
cm	str	(P)	Cursor motion
co	num		Number of columns in a line
cr	str	(P*)	Carriage return, (default **^M**)
cs	str	(P)	Change scrolling region (vt100), like cm
cv	str	(P)	Like ch but vertical only.
da	bool		Display may be retained above
dB	num		Number of millisec of bs delay needed
db	bool		Display may be retained below
dC	num		Number of millisec of cr delay needed
dc	str	(P*)	Delete character
dF	num		Number of millisec of ff delay needed

dl	str	(P*)	Delete line
dm	str		Delete mode (enter)
dN	num		Number of millisec of nl delay needed
do	str		Down one line
dT	num		Number of millisec of tab delay needed
ed	str		End delete mode
ei	str		End insert mode; give ":ei=:" if **ic**
eo	str		Can erase overstrikes with a blank
ff	str	(P*)	Hardcopy terminal page eject (default **^L**)
hc	bool		Hardcopy terminal
hd	str		Half-line down (forward 1/2 linefeed)
ho	str		Home cursor (if no **cm**)
hu	str		Half-line up (reverse 1/2 linefeed)
hz	str		Hazeltine; can't print ~'s
ic	str	(P)	Insert character
if	str		Name of file containing **is**
im	bool		Insert mode (enter); give ":im=:" if **ic**
in	bool		Insert mode distinguishes nulls on display
ip	str	(P*)	Insert pad after character inserted
is	str		Terminal initialization string
k0-k9	str		Sent by "other" function keys 0-9
kb	str		Sent by backspace key
kd	str		Sent by terminal down arrow key
ke	str		Out of "keypad transmit" mode
kh	str		Sent by home key
kl	str		Sent by terminal left arrow key
kn	num		Number of "other" keys
ko	str		Termcap entries for other non-function keys
kr	str		Sent by terminal right arrow key
ks	str		Put terminal in "keypad transmit" mode
ku	str		Sent by terminal up arrow key
l0-l9	str		Labels on "other" function keys
li	num		Number of lines on screen or page
ll	str		Last line, first column (if no **cm**)
ma	str		Arrow key map, used by vi version 2 only
mi	bool		Safe to move while in insert mode
ml	str		Memory lock on above cursor.
ms	bool		Safe to move while in standout and underline mode
mu	str		Memory unlock (turn off memory lock).
nc	bool		No correctly working carriage return (DM2500,H2000)
nd	str		Non-destructive space (cursor right)
nl	str	(P*)	Newline character (default **\n**)
ns	bool		Terminal is a CRT but doesn't scroll.
os	bool		Terminal overstrikes
pc	str		Pad character (rather than null)
pt	bool		Has hardware tabs (may need to be set with **is**)

se	str		End stand out mode
sf	str	(P)	Scroll forwards
sg	num		Number of blank chars left by so or se
so	str		Begin stand out mode
sr	str	(P)	Scroll reverse (backwards)
ta	str	(P)	Tab (other than ^I or with padding)
tc	str		Entry of similar terminal - must be last
te	str		String to end programs that use **cm**
ti	str		String to begin programs that use **cm**
uc	str		Underscore one char and move past it
ue	str		End underscore mode
ug	num		Number of blank chars left by us or ue
ul	bool		Terminal underlines even though it doesn't overstrike
up	str		Upline (cursor up)
us	str		Start underscore mode
vb	str		Visible bell (may not move cursor)
ve	str		Sequence to end open/visual mode
vs	str		Sequence to start open/visual mode
xb	bool		Beehive (f1=escape, f2=ctrl C)
xn	bool		A newline is ignored after a wrap (Concept)
xr	bool		Return acts like **ce** \r \n (Delta Data)
xs	bool		Standout not erased by writing over it (HP 264?)
xt	bool		Tabs are destructive, magic so char (Teleray 1061)

A Sample Entry

The following entry, which describes the Concept–100, is among the more complex entries in the *termcap* file as of this writing. (This particular concept entry is outdated, and is used as an example only.)

```
c1 |c100 |concept100:is=\EU\Ef\E7\E5\E8\El\ENH\EK\E\200\Eo&\200:\
        :al=3*\E^R:am:bs:cd=16*\E^C:ce=16\E^S:cl=2*^L:cm=\Ea%+ %+ :co#80:\
        :dc=16\E^A:dl=3*\E^B:ei=\E\200:eo:im=\E^P:in:ip=16*:li#24:mi:nd=\E=:\
        :se=\Ed\Ee:so=\ED\EE:ta=8\t:ul:up=\E;:vb=\Ek\EK:xn:
```

Entries may continue onto multiple lines by giving a \ as the last character of a line, and that empty fields may be included for readability (here between the last field on a line and the first field on the next). Capabilities in *termcap* are of three types: Boolean capabilities which indicate that the terminal has some particular feature, numeric capabilities giving the size of the terminal or the size of particular delays, and string capabilities, which give a sequence which can be used to perform particular terminal operations.

Types of Capabilities

All capabilities have two letter codes. For instance, the fact that the Concept has "automatic margins" (i.e. an automatic return and linefeed when the end of a line is reached) is indicated by the capability **am**. Hence the description of the Concept includes **am**. Numeric capabilities are followed by the character '#' and then the value. Thus **co** which indicates the number of columns the terminal has gives the value '80' for the Concept.

Finally, string valued capabilities, such as **ce** (clear to end of line sequence) are given by the two character code, an '=', and then a string ending at the next following ':'. A delay in milliseconds may appear after the '=' in such a capability, and padding characters are supplied by the editor after the remainder of the string is sent to provide this delay. The delay can be either a integer, e.g. '20', or an integer followed by an '*', i.e. '3*'. A '*' indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-unit padding required. When a '*' is specified, it is sometimes useful to give a delay of the form '3.5' specify a delay per unit to tenths of milliseconds.

A number of escape sequences are provided in the string valued capabilities for easy encoding of characters there. A **\E** maps to an ESCAPE character, **^x** maps to a control-x for any appropriate x, and the sequences **\n \r \t \b \f** give a newline, return, tab, backspace and formfeed. Finally, characters may be given as three octal digits after a \, and the characters ^ and \ may be given as \^ and \\. If it is necessary to place a : in a capability it must be escaped in octal as **\072**. If it is necessary to place a null character in a string capability it must be encoded as **\200**. The routines which deal with *termcap* use C strings, and strip the high bits of the output very late so that a **\200** comes out as a **\000** would.

Preparing Descriptions

We now outline how to prepare descriptions of terminals. The most effective way to prepare a terminal description is by imitating the description of a similar terminal in *termcap* and to build up a description gradually, using partial descriptions with *ex* to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the *termcap* file to describe it or bugs in *ex*. To easily test a new terminal description you can set the environment variable TERMCAP to a pathname of a file containing the description you are working on and the editor will look there rather than in */etc/termcap.* TERMCAP can also be set to the termcap entry itself to avoid reading the file when starting up the editor. (This only works on version 7 systems.)

Basic capabilities

The number of columns on each line for the terminal is given by the **co** numeric capability. If the terminal is a CRT, then the number of lines on the screen is given by the **li** capability. If the terminal wraps around to the beginning of the next line when it reaches the right margin, then it should have the **am** capability. If the terminal can clear its screen, then this is given by the **cl** string capability. If the terminal can backspace, then it should have the **bs** capability, unless a backspace is accomplished by a character other than **^H** (ugh) in which case you should give this character as the **bc** string capability. If it overstrikes (rather than clearing a position when a character is struck over) then it should have the **os** capability.

A very important point here is that the local cursor motions encoded in *termcap* are undefined at the left and top edges of a CRT terminal. The editor will never attempt to backspace around the left edge, nor will it attempt to go up locally off the top. The editor assumes that feeding off the bottom of the screen will cause the screen to scroll up, and the **am** capability tells whether the cursor sticks at the right edge of the screen. If the terminal has switch selectable automatic margins, the *termcap* file usually assumes that this is on, i.e. **am**.

These capabilities suffice to describe hardcopy and "glass-tty" terminals. Thus the model 33 teletype is described as

```
t3 |33 |tty33:co#72:os
```

while the Lear Siegler ADM-3 is described as

```
cl |adm3|3|lsi adm3:am:bs:cl=^Z:li#24:co#80
```

Cursor addressing

Cursor addressing in the terminal is described by a **cm** string capability, with *printf*(3S) like escapes **%x** in it. These substitute to encodings of the current line or column position, while other characters are passed through unchanged. If the **cm** string is thought of as being a function, then its arguments are the line and then the column to which motion is desired, and the **%** encodings have the following meanings:

%d	as in *printf*, 0 origin
%2	like %2d
%3	like %3d
%.	like %c
%+x	adds *x* to value, then %.
%>xy	if value > x adds y, no output.
%r	reverses order of line and column, no output
%i	increments line/column (for 1 origin)
%%	gives a single %
%n	exclusive or row and column with 0140 (DM2500)
%B	BCD (16*(x/10)) + (x%10), no output.
%D	Reverse coding (x-2*(x%16)), no output. (Delta Data).

Consider the HP2645, which, to get to row 3 and column 12, needs to be sent \E&a12c03Y padded for 6 milliseconds. Note that the order of the rows and columns is inverted here, and that the row and column are printed as two digits. Thus its **cm** capability is "cm=6\E&%r%2c%2Y". The Microterm ACT-IV needs the current row and column sent preceded by a **^T**, with the row and column simply encoded in binary, "cm=^T%.%.". Terminals which use "%." need to be able to backspace the cursor (**bs** or **bc**), and to move the cursor up one line on the screen (**up** introduced below). This is necessary because it is not always safe to transmit **\t**, **\n** **^D** and **\r**, as the system may change or discard them.

A final example is the LSI ADM-3a, which uses row and column offset by a blank character, thus "cm=\E=%+ %+ ".

Cursor motions

If the terminal can move the cursor one position to the right, leaving the character at the current position unchanged, then this sequence should be given as **nd** (non-destructive space). If it can move the cursor up a line on the screen in the same column, this should be given as **up**. If the terminal has no cursor addressing capability, but can home the cursor (to very upper left corner of screen) then this can be given as **ho**; similarly a fast way of getting to the lower left hand corner can be given as **ll**; this may involve going up with **up** from the home position, but the editor will never do this itself (unless **ll** does) because it makes no assumption about the effect of moving up from the home position.

Area clears

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, this should be given as **ce**. If the terminal can clear from the current position to the end of the display, then this should be given as **cd**. The editor only uses **cd** from the first column of a line.

Insert/delete line

If the terminal can open a new blank line before the line where the cursor is, this should be given as **al**; this is done only from the first position of a line. The cursor must then appear on the newly blank line. If the terminal can delete the line which the cursor is on, then this should be given as **dl**; this is done only from the first position on the line to be deleted. If the terminal can scroll the screen backwards, then this can be given as **sb**, but just **al** suffices. If the terminal can retain display memory above then the **da** capability should be given; if display memory can be retained below then **db** should be given. These let the editor understand that deleting a line on the screen may bring non-blank lines up from below or that scrolling back with **sb** may bring down non-blank lines.

Insert/delete character

There are two basic kinds of intelligent terminals with respect to insert/delete character which can be described using *termcap.* The most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line rigidly. Other terminals, such as the Concept 100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen which is either eliminated, or expanded to two untyped blanks. You can find out which kind of terminal you have by clearing the screen and then typing text separated by cursor motions. Type "abc def" using local cursor motions (not spaces) between the "abc" and the "def". Then position the cursor before the "abc" and put the terminal in insert mode. If typing characters causes the rest of the line to shift rigidly and characters to fall off the end, then your terminal does not distinguish between blanks and untyped positions. If the "abc" shifts over to the "def" which then move together around the end of the current line and onto the next as you insert, you have the second type of terminal, and should give the capability **in**, which stands for "insert null". If your terminal does something different and unusual then you may have to modify the editor to get it to use the insert mode your terminal defines. We have seen no terminals which have an insert mode not not falling into one of these two classes.

The editor can handle both terminals which have an insert mode, and terminals which send a simple sequence to open a blank position on the current line. Give as **im** the sequence to get into insert mode, or give it an empty value if your terminal uses a sequence to insert a blank position. Give as **ei** the sequence to leave insert mode (give this, with an empty value also if you gave **im** so). Now give as **ic** any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode will not give **ic**, terminals which send a sequence to open a screen position should give it here. (Insert mode is preferable to the sequence to open a position on the screen if your terminal has both.) If post insert padding is needed, give this as a number of milliseconds in **ip** (a string option). Any other sequence which may need to be sent after an insert of a single character may also be given in **ip**.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (e.g. if there is a tab after the insertion position). If your terminal allows motion while in insert mode you can give the capability **mi** to speed up inserting in this case. Omitting **mi** will affect only speed. Some terminals (notably Datamedia's) must not have **mi** because of the way their insert mode works.

Finally, you can specify delete mode by giving **dm** and **ed** to enter and exit delete mode, and **dc** to delete a single character while in delete mode.

Highlighting, underlining, and visible bells

If your terminal has sequences to enter and exit standout mode these can be given as **so** and **se** respectively. If there are several flavors of standout mode (such as inverse video, blinking, or underlining – half bright is not usually an acceptable "standout" mode unless the terminal is in inverse video mode constantly) the preferred mode is inverse video by itself. If the code to change into or out of standout mode leaves one or even two blank spaces on the screen, as the TVI 912 and Teleray 1061 do, then **ug** should be given to tell how many spaces are left.

Codes to begin underlining and end underlining can be given as **us** and **ue** respectively. If the terminal has a code to underline the current character and move the cursor one space to the right, such as the Microterm Mime, this can be given as **uc**. (If the underline code does not move the cursor to the right, give the code followed by a nondestructive space.)

Many terminals, such as the HP 2621, automatically leave standout mode when they move to a new line or the cursor is addressed. Programs using standout mode should exit standout mode before moving the cursor or sending a newline.

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement) then this can be given as **vb**; it must not move the cursor. If the terminal should be placed in a different mode during open and visual modes of *ex,* this can be given as **vs** and **ve**, sent at the start and end of these modes respectively. These can be used to change, e.g., from a underline to a block cursor and back.

If the terminal needs to be in a special mode when running a program that addresses the cursor, the codes to enter and exit this mode can be given as **ti** and **te**. This arises, for example, from terminals like the Concept with more than one page of memory. If the terminal has only memory relative cursor addressing and not screen relative cursor addressing, a one screen-sized window must be fixed into the terminal for cursor addressing to work properly.

If your terminal correctly generates underlined characters (with no special codes needed) even though it does not overstrike, then you should give the capability **ul**. If overstrikes are erasable with a blank, then this should be indicated by giving **eo**.

Keypad

If the terminal has a keypad that transmits codes when the keys are pressed, this information can be given. Note that it is not possible to handle terminals where the keypad only works in local (this applies, for example, to the unshifted HP 2621 keys). If the keypad can be set to transmit or not transmit, give these codes as **ks** and **ke**. Otherwise the keypad is assumed to always transmit. The codes sent by the left arrow, right arrow, up arrow, down arrow, and home keys can be given as **kl, kr, ku, kd,** and **kh** respectively. If there are function keys such as f0, f1, ..., f9, the codes they send can be given as **k0, k1, ..., k9**. If these keys have labels other than the default f0 through f9, the labels can be given as **l0, l1, ..., l9**. If there are other keys that transmit the same code as the terminal expects for the corresponding function, such as clear screen, the *termcap* 2 letter codes can be given in the **ko** capability, for example, ":ko=cl,ll,sf,sb:", which says that the terminal has clear, home down, scroll down, and scroll up keys that transmit the same thing as the cl, ll, sf, and sb entries.

The **ma** entry is also used to indicate arrow keys on terminals which have single character arrow keys. It is obsolete but still in use in version 2 of vi, which must be run on some minicomputers due to memory limitations. This field is redundant with **kl, kr, ku, kd,** and **kh**. It consists of groups of two characters. In each group, the first character is what an arrow key sends, the second character is the corresponding vi command. These commands are **h** for **kl**, **j** for **kd**, **k** for **ku**, **l** for **kr**, and **H** for **kh**. For example, the mime would be **:ma=^Kj^Zk^Xl:** indicating arrow keys left (^H), down (^K), up (^Z), and right (^X). (There is no home key on the mime.)

Miscellaneous

If the terminal requires other than a null (zero) character as a pad, then this can be given as **pc**.

If tabs on the terminal require padding, or if the terminal uses a character other than **^I** to tab, then this can be given as **ta**.

Hazeltine terminals, which don't allow '~' characters to be printed should indicate **hz**. Datamedia terminals, which echo carriage-return linefeed for carriage return and then ignore a following linefeed should indicate **nc**. Early Concept terminals, which ignore a linefeed immediately after an **am** wrap, should indicate **xn**. If an erase-eol is required to get rid of standout (instead of merely writing on top of it), **xs** should be given. Teleray terminals, where tabs turn all characters moved over to blanks, should indicate **xt**. Other specific terminal problems may be corrected by adding more capabilities of the form **x***x*.

Other capabilities include **is**, an initialization string for the terminal, and **if**, the name of a file containing long initialization strings. These strings are expected to properly clear and then set the tabs on the terminal, if the terminal has settable tabs. If both are given, **is** will be printed before **if**. This is useful where **if** is */usr/lib/tabset/std* but **is** clears the tabs first.

Similar Terminals

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability **tc** can be given with the name of the similar terminal. This capability must be *last* and the combined length of the two entries must not exceed 1024. Since *termlib* routines search the entry from left to right, and since the tc capability is replaced by the corresponding entry, the capabilities given at the left override the ones in the similar terminal. A capability can be canceled with **xx@** where xx is the capability. For example, the entry

```
hn |2621nl:ks@:ke@:tc=2621:
```

defines a 2621nl that does not have the **ks** or **ke** capabilities, and hence does not turn on the function key labels when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

FILES

/etc/termcap file containing terminal descriptions

SEE ALSO

ex(1), curses(3X), termcap(3X), tset(1), vi(1), ul(1), more(1)

AUTHOR

William Joy
Mark Horton added underlining and keypad support

BUGS

Ex allows only 256 characters for string capabilities, and the routines in *termcap*(3X) do not check for overflow of this buffer. The total length of a single entry (excluding only escaped newlines) may not exceed 1024.

The **ma**, **vs**, and **ve** entries are specific to the *vi* program.

Not all programs support all entries. There are entries that are not supported by any program.

NAME

ttys – terminal initialization data

DESCRIPTION

The *ttys* file is read by the *init* program and specifies which terminal special files are to have a process created for them so that people can log in. There is one line in the *ttys* file per special file.

The first character of a line in the *ttys* file is either '0' or '1'. If the first character on the line is a '0', the *init* program ignores that line. If the first character on the line is a '1', the *init* program creates a login process for that line. The second character on each line is used as an argument to *getty*(8), which performs such tasks as baud-rate recognition, reading the login name, and calling *login*. For normal lines, the character is '0'; other characters can be used, for example, with hard-wired terminals where speed recognition is unnecessary or which have special characteristics. (*Getty* will have to be fixed in such cases.) The remainder of the line is the terminal's entry in the device directory, /dev.

FILES

/etc/ttys

SEE ALSO

gettytab(5), init(8), getty(8), login(1)

NAME
ttytype – data base of terminal types by port

SYNOPSIS
/etc/ttytype

DESCRIPTION
Ttytype is a database containing, for each tty port on the system, the kind of terminal that is attached to it. There is one line per port, containing the terminal kind (as a name listed in termcap (5)), a space, and the name of the tty, minus /dev/.

This information is read by *tset*(1) and by *login*(1) to initialize the TERM variable at login time.

SEE ALSO
tset(1), login(1)

BUGS
Some lines are merely known as "dialup" or "plugboard".

NAME

types – primitive system data types

SYNOPSIS

#include <sys/types.h>

DESCRIPTION

The data types defined in the include file are used in UNIX system code; some data of these types are accessible to user code:

```
/*      types.h   6.1     83/07/29*/

#ifndef __TYPES
#define __TYPES

/*
 * Basic system types and major/minor device
 * constructing/busting macros.
 */

/* major part of a device */
#define MAJOR(x)        ((int)(((unsigned)(x)>>8)&0377))

/* minor part of a device */
#define MINOR(x)        ((int)((x)&0377))

/* make a device number */
#define makedev(x,y)    ((dev_t)(((x)<<8) | (y)))

typedef unsigned char   u_char;
typedef unsigned short  u_short;
typedef unsigned int    u_int;
typedef unsigned long   u_long;
typedef unsigned short  ushort;/* sys III compat */

#ifdef vax
typedef struct    _physadr { int r[1]; } *physadr;
typedef struct    label_t {
        int       val[14];
} label_t;
#endif
typedef struct    _quad { long val[2]; } quad;
typedef long      daddr_t;
typedef char *    caddr_t;
typedef u_long    ino_t;
typedef long      swblk_t;
typedef int       size_t;
typedef int       time_t;
typedef short     dev_t;
typedef int       off_t;

typedef struct    fd_set { int fds_bits[1]; } fd_set;
```

```
#endif
```

The form *daddr_t* is used for disk addresses except in an i-node on disk, see *fs*(5). Times are encoded in seconds since 00:00:00 GMT, January 1, 1970. The major and minor parts of a device code specify kind and unit number of a device and are installation-dependent. Offsets are measured in bytes from the beginning of a file. The *label_t* variables are used to save the processor state while another process is running.

SEE ALSO

fs(5), time(3), lseek(2), adb(1)

NAME

utmp, wtmp – login records

SYNOPSIS

#include <utmp.h>

DESCRIPTION

The *utmp* file records information about who is currently using the system. The file is a sequence of entries with the following structure declared in the include file:

```
/* utmp.h          4.283/05/22*/

/*
 * Structure of utmp and wtmp files.
 *
 * Assuming the number 8 is unwise.
 */
struct utmp {
    char  ut_line[8];   /* tty name */
    char  ut_name[8]; /* user id */
    char  ut_host[16]; /* host name, if remote */
    long  ut_time;      /* time on */
};
```

This structure gives the name of the special file associated with the user's terminal, the user's login name, and the time of the login in the form of *time*(3C).

The *wtmp* file records all logins and logouts. A null user name indicates a logout on the associated terminal. Furthermore, the terminal name '~' indicates that the system was rebooted at the indicated time; the adjacent pair of entries with terminal names '|' and '}' indicate the system-maintained time just before and just after a *date* command has changed the system's idea of the time.

Wtmp is maintained by *login*(1) and *init*(8). Neither of these programs creates the file, so if it is removed record-keeping is turned off. It is summarized by *ac*(8).

FILES

/etc/utmp
/usr/adm/wtmp

SEE ALSO

login(1), init(8), who(1), ac(8)

NAME

uuencode – format of an encoded uuencode file

DESCRIPTION

Files output by ***uuencode(1C)*** consist of a header line, followed by a number of body lines, and a trailer line. *Uudecode(1C)* will ignore any lines preceding the header or following the trailer. Lines preceding a header must not, of course, look like a header.

The header line is distinguished by having the first 6 characters "begin ". The word *begin* is followed by a mode (in octal), and a string which names the remote file. A space separates the three items in the header line.

The body consists of a number of lines, each at most 62 characters long (including the trailing newline). These consist of a character count, followed by encoded characters, followed by a newline. The character count is a single printing character, and represents an integer, the number of bytes the rest of the line represents. Such integers are always in the range from 0 to 63 and can be determined by subtracting the character space (octal 40) from the character.

Groups of 3 bytes are stored in 4 characters, 6 bits per character. All are offset by a space to make the characters printing. The last line may be shorter than the normal 45 bytes. If the size is not a multiple of 3, this fact can be determined by the value of the count on the last line. Extra garbage will be included to make the character count a multiple of 4. The body is terminated by a line with a count of zero. This line consists of one ASCII space.

The trailer line consists of "end" on a line by itself.

SEE ALSO

uuencode(1C), uudecode(1C), uusend(1C), uucp(1C), mail(1)

NAME

vgrindefs – vgrind's language definition data base

SYNOPSIS

/usr/lib/vgrindefs

DESCRIPTION

Vgrindefs contains all language definitions for vgrind. The data base is very similar to *termcap*(5).

FIELDS

The following table names and describes each field.

Name	**Type**	**Description**
pb	str	regular expression for start of a procedure
bb	str	regular expression for start of a lexical block
be	str	regular expression for the end of a lexical block
cb	str	regular expression for the start of a comment
ce	str	regular expression for the end of a comment
sb	str	regular expression for the start of a string
se	str	regular expression for the end of a string
lb	str	regular expression for the start of a character constant
le	str	regular expression for the end of a character constant
tl	bool	present means procedures are only defined at the top lexical level
oc	bool	present means upper and lower case are equivalent
kw	str	a list of keywords separated by spaces

Example

The following entry, which describes the C language, is typical of a language entry.

```
C|c:    :pb=^\d?*?\d?\p\d??):bb={:be=}:cb=/*:ce=*/:
        :sb=":se=\e":lb=':le=\e':tl:\
        :kw=asm auto break case char continue default\
        do double else enum extern float for fortran goto \
        if int long register return short sizeof static\
        struct switch typedef union unsigned while #define\
        #else #endif #if #ifdef #ifndef #include #undef \
        # define else endif if ifdef ifndef include undef:
```

Note that the first field is just the language name (and any variants of it). Thus the C language could be specified to *vgrind*(1) as "c" or "C".

Entries may continue onto multiple lines by giving a \ as the last character of a line. Capabilities in *vgrindefs* are of two types: Boolean capabilities which indicate that the language has some particular feature and string capabilities which give a regular expression or keyword list.

REGULAR EXPRESSIONS

Vgrindefs uses regular expression which are very similar to those of *ex*(1) and *lex*(1). The characters '^', '$', ':' and '\' are reserved characters and must be "quoted" with a preceding \ if they are to be included as normal characters. The metasymbols and their meanings are:

$	the end of a line
^	the beginning of a line
\d	a delimiter (space, tab, newline, start of line)
\a	matches any string of symbols (like .* in lex)
\p	matches any alphanumeric name. In a procedure definition (pb) the string that matches this symbol is used as the procedure name.
()	grouping
|	alternation
?	last item is optional
\e	preceding any string means that the string will not match an input string if the input string is preceded by an escape character (\\). This is typically used for languages (like C) which can include the string delimiter in a string b escaping it.

Unlike other regular expressions in the system, these match words and not characters. Hence something like "(tramp|steamer)flies?" would match "tramp", "steamer", "trampflies", or "steamerflies".

KEYWORD LIST

The keyword list is just a list of keywords in the language separated by spaces. If the "oc" boolean is specified, indicating that upper and lower case are equivalent, then all the keywords should be specified in lower case.

FILES

/usr/lib/vgrindefs file containing terminal descriptions

SEE ALSO

vgrind(1), troff(1)

AUTHOR

Dave Presotto

BUGS

SECTION 6

ICON/UXB OPERATING SYSTEM GAMES

ICON®

NAME
aardvark – yet another exploration game

SYNOPSIS
/usr/games/aardvark

DESCRIPTION
Aardvark is yet another computer fantasy simulation game of the adventure/zork genre. This one is written in DDL (Dungeon Definition Language) and is intended primarily as an example of how to write a dungeon in DDL.

FILES
/usr/games/lib/ddlrun ddl interpreter
/usr/games/lib/aardvarkinternal form of aardvark dungeon

AUTHOR
Mike Urban, UCLA

NAME

adventure – an exploration game

SYNOPSIS

/usr/games/adventure

DESCRIPTION

The object of the game is to locate and explore Colossal Cave, find the treasures hidden there, and bring them back to the building with you. The program is self-describing to a point, but part of the game is to discover its rules.

To terminate a game, type 'quit'; to save a game for later resumption, type 'suspend'.

BUGS

Saving a game creates a large executable file instead of just the information needed to resume the game.

NAME

arithmetic – provide drill in number facts

SYNOPSIS

/usr/games/arithmetic [**+–x/**] [range]

DESCRIPTION

Arithmetic types out simple arithmetic problems, and waits for an answer to be typed in. If the answer is correct, it types back "Right!", and a new problem. If the answer is wrong, it replies "What?", and waits for another answer. Every twenty problems, it publishes statistics on correctness and the time required to answer.

To quit the program, type an interrupt (delete).

The first optional argument determines the kind of problem to be generated; **+–x/** respectively cause addition, subtraction, multiplication, and division problems to be generated. One or more characters can be given; if more than one is given, the different types of problems will be mixed in random order; default is **+–**

Range is a decimal number; all addends, subtrahends, differences, multiplicands, divisors, and quotients will be less than or equal to the value of *range*. Default *range* is 10.

At the start, all numbers less than or equal to *range* are equally likely to appear. If the respondent makes a mistake, the numbers in the problem which was missed become more likely to reappear.

As a matter of educational philosophy, the program will not give correct answers, since the learner should, in principle, be able to calculate them. Thus the program is intended to provide drill for someone just past the first learning stage, not to teach number facts *de novo*. For almost all users, the relevant statistic should be time per problem, not percent correct.

NAME

backgammon – the game

SYNOPSIS

/usr/games/backgammon

DESCRIPTION

This program does what you expect. It will ask whether you need instructions.

NAME
banner – print large banner on printer

SYNOPSIS
/usr/games/banner [**–w***n*] message ...

DESCRIPTION
Banner prints a large, high quality banner on the standard output. If the message is omitted, it prompts for and reads one line of its standard input. If **–w** is given, the output is scrunched down from a width of 132 to *n* , suitable for a narrow terminal. If *n* is omitted, it defaults to 80.

The output should be printed on a hard-copy device, up to 132 columns wide, with no breaks between the pages. The volume is enough that you want a printer or a fast hardcopy terminal, but if you are patient, a decwriter or other 300 baud terminal will do.

BUGS
Several ASCII characters are not defined, notably <, >, [,], \, ^, _, {, }, |, and ~. Also, the characters ", ', and & are funny looking (but in a useful way.)

The **–w** option is implemented by skipping some rows and columns. The smaller it gets, the grainier the output. Sometimes it runs letters together.

AUTHOR
Mark Horton

NAME

battlestar – a tropical adventure game

SYNOPSIS

battlestar [**-r** (recover a saved game)]

DESCRIPTION

Battlestar is an adventure game in the classic style. However, It's slightly less of a puzzle and more a game of exploration. There are a few magical words in the game, but on the whole, simple English should suffice to make one's desires understandable to the parser.

THE SETTING

In the days before the darkness came, when battlestars ruled the heavens...

> Three He made and gave them to His daughters,
> Beautiful nymphs, the goddesses of the waters.
> One to bring good luck and simple feats of wonder,
> Two to wash the lands and churn the waves asunder,
> Three to rule the world and purge the skies with thunder.

In those times great wizards were known and their powers were beyond belief. They could take any object from thin air, and, uttering the word 'su' could disappear.

In those times men were known for their lust of gold and desire to wear fine weapons. Swords and coats of mail were fashioned that could withstand a laser blast.

But when the darkness fell, the rightful reigns were toppled. Swords and helms and heads of state went rolling across the grass. The entire fleet of battlestars was reduced to a single ship.

SAMPLE COMMANDS

```
take    ---     take an object
drop    ---     drop an object

wear    ---     wear an object you are holding
draw    ---     carry an object you are wearing

puton  ---      take an object and wear it
take off --     draw an object and drop it

throw  <object> <direction>

!       <shell esc>
```

IMPLIED OBJECTS

```
>-: take watermelon
watermelon:
Taken.
>-: eat
watermelon:
Eaten.
>-: take knife and sword and apple, drop all
knife:
```

```
Taken.
broadsword:
Taken.
apple:
Taken.
knife:
Dropped.
broadsword:
Dropped.
apple:
Dropped.
>-: get
knife:
Taken.
```

Notice that the "shadow" of the next word stays around if you want to take advantage of it. That is, saying "take knife" and then "drop" will drop the knife you just took.

SCORE & INVEN

The two commands "score" and "inven" will print out your current status in the game.

SAVING A GAME

The command "save" will save your game in a file called "Bstar." You can recover a saved game by using the "-r" option when you start up the game.

DIRECTIONS

The compass directions N, S, E, and W can be used if you have a compass. If you don't have a compass, you'll have to say R, L, A, or B, which stand for Right, Left, Ahead, and Back. Directions printed in room descriptions are always printed in R, L, A, & B relative directions.

HISTORY

I wrote Battlestar in 1979 in order to experiment with the niceties of the C Language. Most interesting things that happen in the game are hardwired into the code, so don't send me any hate mail about it! Instead, enjoy art for art's sake!

AUTHOR

David Riggle

INSPIRATION & ASSISTANCE

Chris Guthrie
Peter Da Silva
Kevin Brown
Edward Wang
Ken Arnold & Company

BUGS

Countless.

FAN MAIL

Send to: edward%ucbarpa@Berkeley.arpa,
chris%ucbcory@berkeley.arpa, or
riggle.pa@xerox.arpa.

NAME
bcd – convert to antique media

SYNOPSIS
/usr/games/bcd text

DESCRIPTION
Bcd converts the literal *text* into a form familiar to old-timers.

SEE ALSO
dd(1)

NAME

boggle – play the game of boggle

SYNOPSIS

/usr/games/boggle [+] [++]

DESCRIPTION

This program is intended for people wishing to sharpen their skills at Boggle (TM Parker Bros.). If you invoke the program with 4 arguments of 4 letters each, (*e.g.* "**boggle appl epie moth erhd**") the program forms the obvious Boggle grid and lists all the words from **/usr/dict/words** found therein. If you invoke the program without arguments, it will generate a board for you, let you enter words for 3 minutes, and then tell you how well you did relative to **/usr/dict/words**.

The object of Boggle is to find, within 3 minutes, as many words as possible in a 4 by 4 grid of letters. Words may be formed from any sequence of 3 or more adjacent letters in the grid. The letters may join horizontally, vertically, or diagonally. However, no position in the grid may be used more than once within any one word. In competitive play amongst humans, each player is given credit for those of his words which no other player has found.

In interactive play, enter your words separated by spaces, tabs, or newlines. A bell will ring when there is 2:00, 1:00, 0:10, 0:02, 0:01, and 0:00 time left. You may complete any word started before the expiration of time. You can surrender before time is up by hitting 'break'. While entering words, your erase character is only effective within the current word and your line kill character is ignored.

Advanced players may wish to invoke the program with 1 or 2 +'s as the first argument. The first + removes the restriction that positions can only be used once in each word. The second + causes a position to be considered adjacent to itself as well as its (up to) 8 neighbors.

NAME

canfield, cfscores – the solitaire card game canfield

SYNOPSIS

/usr/games/canfield
/usr/games/cfscores

DESCRIPTION

If you have never played solitaire before, it is recommended that you consult a solitaire instruction book. In Canfield, tableau cards may be built on each other downward in alternate colors. An entire pile must be moved as a unit in building. Top cards of the piles are available to be able to be played on foundations, but never into empty spaces.

Spaces must be filled from the stock. The top card of the stock also is available to be played on foundations or built on tableau piles. After the stock is exhausted, tableau spaces may be filled from the talon and the player may keep them open until he wishes to use them.

Cards are dealt from the hand to the talon by threes and this repeats until there are no more cards in the hand or the player quits. To have cards dealt onto the talon the player types 'ht' for his move. Foundation base cards are also automatically moved to the foundation when they become available.

The command 'c' causes *canfield* to maintain card counting statistics on the bottom of the screen. When properly used this can greatly increase ones chances of winning.

The rules for betting are somewhat less strict than those used in the official version of the game. The initial deal costs $13. You may quit at this point or inspect the game. Inspection costs $13 and allows you to make as many moves as is possible without moving any cards from your hand to the talon. (the initial deal places three cards on the talon; if all these cards are used, three more are made available.) Finally, if the game seems interesting, you must pay the final installment of $26. At this point you are credited at the rate of $5 for each card on the foundation; as the game progresses you are credited with $5 for each card that is moved to the foundation. Each run through the hand after the first costs $5. The card counting feature costs $1 for each unknown card that is identified. If the information is toggled on, you are only charged for cards that became visible since it was last turned on. Thus the maximum cost of information is $34. Playing time is charged at a rate of $1 per minute.

With no arguments, the program *cfscores* prints out the current status of your canfield account. If a user name is specified, it prints out the status of their canfield account. If the **–a** flag is specified, it prints out the canfield accounts for all users that have played the game since the database was set up.

FILES

/usr/games/canfield	the game itself
/usr/games/cfscores	the database printer
/usr/games/lib/cfscores	the database of scores

BUGS

It is impossible to cheat.

AUTHORS

Originally written: Steve Levine
Further random hacking by: Steve Feldman, Kirk McKusick, Mikey Olson, and Eric Allman.

NAME

ching – the book of changes and other cookies

SYNOPSIS

/usr/games/ching [hexagram]

DESCRIPTION

The *I Ching* or *Book of Changes* is an ancient Chinese oracle that has been in use for centuries as a source of wisdom and advice.

The text of the *oracle* (as it is sometimes known) consists of sixty-four *hexagrams,* each symbolized by a particular arrangement of six straight (—–) and broken (– –) lines. These lines have values ranging from six through nine, with the even values indicating the broken lines.

Each hexagram consists of two major sections. The **Judgement** relates specifically to the matter at hand (E.g., "It furthers one to have somewhere to go.") while the **Image** describes the general attributes of the hexagram and how they apply to one's own life ("Thus the superior man makes himself strong and untiring.").

When any of the lines have the values six or nine, they are moving lines; for each there is an appended judgement which becomes significant. Furthermore, the moving lines are inherently unstable and change into their opposites; a second hexagram (and thus an additional judgement) is formed.

Normally, one consults the oracle by fixing the desired question firmly in mind and then casting a set of changes (lines) using yarrow–stalks or tossed coins. The resulting hexagram will be the answer to the question.

Using an algorithm suggested by S. C. Johnson, the UNIX *oracle* simply reads a question from the standard input (up to an EOF) and hashes the individual characters in combination with the time of day, process id and any other magic numbers which happen to be lying around the system. The resulting value is used as the seed of a random number generator which drives a simulated coin–toss divination. The answer is then piped through **nroff** for formatting and will appear on the standard output.

For those who wish to remain steadfast in the old traditions, the oracle will also accept the results of a personal divination using, for example, coins. To do this, cast the change and then type the resulting line values as an argument.

The impatient modern may prefer to settle for Chinese cookies; try *fortune*(6).

SEE ALSO

It furthers one to see the great man.

DIAGNOSTICS

The great prince issues commands,
Founds states, vests families with fiefs.
Inferior people should not be employed.

BUGS

Waiting in the mud
Brings about the arrival of the enemy.

If one is not extremely careful,
Somebody may come up from behind and strike him.
Misfortune.

NAME

cribbage – the card game cribbage

SYNOPSIS

/usr/games/cribbage [**–req**] *name* ...

DESCRIPTION

Cribbage plays the card game cribbage, with the program playing one hand and the user the other. The program will initially ask the user if the rules of the game are needed – if so, it will print out the appropriate section from *According to Hoyle* with *more (I).*

Cribbage options include:

–e When the player makes a mistakes scoring his hand or crib, provide an explanation of the correct score. (This is especially useful for beginning players.)

–q Print a shorter form of all messages – this is only recommended for users who have played the game without specifying this option.

–r Instead of asking the player to cut the deck, the program will randomly cut the deck.

Cribbage first asks the player whether he wishes to play a short game ("once around", to 61) or a long game ("twice around", to 121). A response of 's' will result in a short game, any other response will play a long game.

At the start of the first game, the program asks the player to cut the deck to determine who gets the first crib. The user should respond with a number between 0 and 51, indicating how many cards down the deck is to be cut. The player who cuts the lower ranked card gets the first crib. If more than one game is played, the loser of the previous game gets the first crib in the current game.

For each hand, the program first prints the player's hand, whose crib it is, and then asks the player to discard two cards into the crib. The cards are prompted for one per line, and are typed as explained below.

After discarding, the program cuts the deck (if it is the player's crib) or asks the player to cut the deck (if it's its crib); in the later case, the appropriate response is a number from 0 to 39 indicating how far down the remaining 40 cards are to be cut.

After cutting the deck, play starts with the non-dealer (the person who doesn't have the crib) leading the first card. Play continues, as per cribbage, until all cards are exhausted. The program keeps track of the scoring of all points and the total of the cards on the table.

After play, the hands are scored. The program requests the player to score his hand (and the crib, if it is his) by printing out the appropriate cards (and the cut card enclosed in brackets). Play continues until one player reaches the game limit (61 or 121).

A carriage return when a numeric input is expected is equivalent to typing the lowest legal value; when cutting the deck this is equivalent to choosing the top card.

Cards are specified as rank followed by suit. The ranks may be specified as one of: 'a', '2', '3', '4', '5', '6', '7', '8', '9', 't', 'j', 'q', and 'k', or alternatively, one of: "ace", "two", "three", "four", "five", "six", "seven", "eight", "nine", "ten", "jack", "queen", and "king". Suits may be specified as: 's', 'h', 'd', and 'c', or alternatively as: "spades", "hearts", "diamonds", and "clubs". A card may be specified as: <rank> " " <suit>, or: <rank> " of " <suit>. If the single letter rank and suit designations are used, the space separating the suit and rank may be left out. Also, if only one card of the desired rank is playable, typing the rank is sufficient. For example, if your hand was "2H, 4D, 5C, 6H, JC, KD" and it was desired to discard the king of diamonds, any of the following could be typed: "k", "king", "kd", "k d", "k of d", "king d", "king of d", "k diamonds", "k of diamonds", "king diamonds", or "king of diamonds".

FILES

/usr/games/cribbage

AUTHORS

Earl T. Cohen wrote the logic. Ken Arnold added the screen oriented interface.

NAME
doctor – interact with a psychoanalyst

SYNOPSIS
/usr/games/doctor

DESCRIPTION
Doctor is a lisp-language version of the legendary ELIZA program of Joseph Weizenbaum. This script "simulates" a Rogerian psychoanalyst. Type in lower case, and when you get tired or bored, type your interrupt character (either control-C or Rubout). Remember to type two carriage returns when you want it to answer.

In order to run this you must have a Franz Lisp system in /usr/ucb/lisp.

AUTHORS
Adapted for Lisp by Jon L White, moved to Franz by John Foderaro, from an original script by Joseph Weizenbaum.

NAME

fish – play "Go Fish"

SYNOPSIS

/usr/games/fish

DESCRIPTION

Fish plays the game of "Go Fish", a childrens' card game. The Object is to accumulate 'books' of 4 cards with the same face value. The players alternate turns; each turn begins with one player selecting a card from his hand, and asking the other player for all cards of that face value. If the other player has one or more cards of that face value in his hand, he gives them to the first player, and the first player makes another request. Eventually, the first player asks for a card which is not in the second player's hand: he replies 'GO FISH!' The first player then draws a card from the 'pool' of undealt cards. If this is the card he had last requested, he draws again. When a book is made, either through drawing or requesting, the cards are laid down and no further action takes place with that face value.

To play the computer, simply make guesses by typing a, 2, 3, 4, 5, 6, 7, 8, 9, 10, j, q, or k when asked. Hitting return gives you information about the size of my hand and the pool, and tells you about my books. Saying 'p' as a first guess puts you into 'pro' level; The default is pretty dumb.

NAME

fortune – print a random, hopefully interesting, adage

SYNOPSIS

/usr/games/fortune [–] [**–wsl**]

DESCRIPTION

Fortune with no arguments prints out a random adage. The flags mean:

–w Waits before termination for an amount of time calculated from the number of characters in the message. This is useful if it is executed as part of the logout procedure to guarantee that the message can be read before the screen is cleared.

–s Short messages only.

–l Long messages only.

FILES

/usr/games/lib/fortunes.dat

AUTHOR

Ken Arnold

NAME

hangman – Computer version of the game hangman

SYNOPSIS

/usr/games/hangman

DESCRIPTION

In *hangman,* the computer picks a word from the on-line word list and you must try to guess it. The computer keeps track of which letters have been guessed and how many wrong guesses you have made on the screen in a graphic fashion.

FILES

/usr/dict/words	On-line word list

AUTHOR

Ken Arnold

NAME

hunt – a multi-player multi-terminal game

SYNOPSIS

/usr/games/hunt [-q] [**-m**] [hostname] [**-l** name]

DESCRIPTION

The object of the game *hunt* is to kill off the other players. There are no rooms, no treasures, and no monsters. Instead, you wander around a maze, find grenades, trip mines, and shoot down walls and players. The more players you kill before you die, the better your score is. If the **–m** flag is given, you enter the game as a monitor (you can see the action but you cannot play).

Hunt normally looks for an active game on the local network; if none is found, it starts one up on the local host. One may specify the location of the game by giving the *hostname* argument. The player name may be specified on the command line by using the **-l** option. This command syntax was chosen for *rlogin/rsh* compatibility. If the **–q** flag is given, *hunt* queries the network and reports if an active game were found. This is useful for .login scripts.

Hunt only works on crt (vdt) terminals with at least 24 lines, 80 columns, and cursor addressing. The screen is divided in to 3 areas. On the right hand side is the status area. It shows you how much damage you've sustained, how many charges you have left, who's in the game, who's scanning (the asterisk in front of the name), who's cloaked (the plus sign in front of the name), and other players' scores. Most of the rest of the screen is taken up by your map of the maze, except for the 24th line, which is used for longer messages that don't fit in the status area.

Hunt uses the same keys to move as *vi* does, *i.e.*, **h**,**j**,**k**, and **l** for left, down, up, right respectively. To change which direction you're facing in the maze, use the upper case version of the movement key (*i.e.*, HJKL).

Other commands are:

f	– Fire (in the direction you're facing) (Takes 1 charge)
g	– Throw grenade (in the direction you're facing) (Takes 9 charges)
F	– Throw satchel charge (Takes 25 charges)
G	– Throw bomb (Takes 49 charges)
o	– Throw small slime bomb (Takes 15 charges)
O	– Throw big slime bomb (Takes 30 charges)
s	– Scan (show where other players are) (Takes 1 charge)
c	– Cloak (hide from scanners) (Takes 1 charge)
^L	– Redraw screen
q	– Quit

Knowing what the symbols on the screen often helps:

-|+	– walls
/\	– diagonal (deflecting) walls
#	– doors (dispersion walls)
;	– small mine
g	– large mine
:	– shot
o	– grenade
O	– satchel charge

@	– bomb
s	– small slime bomb
$	– big slime bomb
> < ^ v	– you facing right, left, up, or down
} { i !	– other players facing right, left, up, or down
*	– explosion
\ | /	
– * –	– grenade and large mine explosion
/ | \	

Satchel and bomb explosions are larger than grenades (5x5, 7x7, and 3x3 respectively).

Other helpful hints:

- You can only fire in the direction you are facing.
- You can only fire three shots in a row, then the gun must cool.
- A shot only affects the square it hits.
- Shots and grenades move 5 times faster than you do.
- To stab someone, you must face that player and move at them.
- Stabbing does 2 points worth of damage and shooting does 5 points.
- Slime does 5 points of damage each time it hits.
- You start with 15 charges and get 5 more for every new player.
- A grenade affects the nine squares centered about the square it hits.
- A satchel affects the twenty-five squares centered about the square it hits.
- A bomb affects the forty-nine squares centered about the square it hits.
- Slime affects all squares it oozes over (15 or 30 respectively).
- One small mine and one large mine is placed in the maze for every new player. A mine has a 5% probability of tripping when you walk directly at it; 50% when going sideways on to it; 95% when backing up on to it. Tripping a mine costs you 5 points or 10 points respectively. Defusing a mine is worth 1 charge or 9 charges respectively.
- You cannot see behind you.
- Scanning lasts for (20 times the number of players) turns. Scanning takes 1 ammo charge, so don't waste all your charges scanning.
- Cloaking lasts for 20 turns.
- Whenever you kill someone, you get 2 more damage capacity points and 2 damage points taken away.
- Maximum typeahead is 5 characters.
- A shot destroys normal (*i.e.*, non-diagonal, non-door) walls.
- Diagonal walls deflect shots and change orientation.
- Doors disperse shots in random directions (up, down, left, right).
- Diagonal walls and doors cannot be destroyed by direct shots but may be destroyed by an adjacent grenade explosion.
- Slime goes around walls, not through them.
- Walls regenerate, reappearing in the order they were destroyed. One percent of the regenerated walls will be diagonal walls or doors. When a wall is generated directly beneath a player, he is thrown in a random direction for a random period of time. When he lands, he sustains damage (up to 20 percent of the amount of damage he had before impact); that is, the less damage he had, the more nimble he is and therefore less likely to hurt himself on landing.
- The environment variable **HUNT** is checked to get the player name. If you don't have this variable set, *hunt* will ask you what name you want to play under. If it is set, you may also set up a single character keyboard map, but then you have to enumerate the options:
  e.g. setenv HUNT "name=Sneaky,mapkey=zoFfGg1f2g3F4G"

sets the player name to Sneaky, and the maps **z** to **o**, **F** to **f**, **G** to **g**, **1** to **f**, **2** to **g**, **3** to **F**, and **4** to **G**. The *mapkey* option must be last.

- It's a boring game if you're the only one playing.

Your score is the ratio of number of kills to number of times you entered the game and is only kept for the duration of a single session of *hunt*.

Hunt normally drives up the load average to be about (number_of_players + 0.5) greater than it would be without a *hunt* game executing. A limit of three players per host and nine players total is enforced by *hunt*.

FILES

/usr/games/lib/hunt.driver game coordinator

AUTHORS

Conrad Huang, Ken Arnold, and Greg Couch; University of California, San Francisco, Computer Graphics Lab

ACKNOWLEDGEMENTS

We thank Don Kneller, John Thomason, Eric Pettersen, and Scott Weiner for providing endless hours of play-testing to improve the character of the game. We hope their significant others will forgive them; we certainly don't.

BUGS

To keep up the pace, not everything is as realistic as possible.

There were some bugs in early releases of 4.2 BSD that *hunt* helped discover; *hunt* will crash your system if those bugs haven't been fixed.

NAME

mille – play Mille Bournes

SYNOPSIS

/usr/games/mille [file]

DESCRIPTION

Mille plays a two-handed game reminiscent of the Parker Brother's game of Mille Bournes with you. The rules are described below. If a file name is given on the command line, the game saved in that file is started.

When a game is started up, the bottom of the score window will contain a list of commands. They are:

P Pick a card from the deck. This card is placed in the 'P' slot in your hand.

D Discard a card from your hand. To indicate which card, type the number of the card in the hand (or "P" for the just-picked card) followed by a <RETURN> or <SPACE>. The <RETURN or <SPACE> is required to allow recovery from typos which can be very expensive, like discarding safeties.

U Use a card. The card is again indicated by its number, followed by a <RETURN> or <SPACE>.

O Toggle ordering the hand. By default off, if turned on it will sort the cards in your hand appropriately. This is not recommended for the impatient on slow terminals.

Q Quit the game. This will ask for confirmation, just to be sure. Hitting <DELETE> (or <RUBOUT>) is equivalent.

S Save the game in a file. If the game was started from a file, you will be given an opportunity to save it on the same file. If you don't wish to, or you did not start from a file, you will be asked for the file name. If you type a <RETURN> without a name, the save will be terminated and the game resumed.

R Redraw the screen from scratch. The command ^L (control 'L') will also work.

W Toggle window type. This switches the score window between the startup window (with all the command names) and the end-of-game window. Using the end-of-game window saves time by eliminating the switch at the end of the game to show the final score. Recommended for hackers and other miscreants.

If you make a mistake, an error message will be printed on the last line of the score window, and a bell will beep.

At the end of each hand or game, you will be asked if you wish to play another. If not, it will ask you if you want to save the game. If you do, and the save is unsuccessful, play will be resumed as if you had said you wanted to play another hand/game. This allows you to use the "S" command to reattempt the save.

AUTHOR

Ken Arnold
(The game itself is a product of Parker Brothers, Inc.)

SEE ALSO

curses(3X), *Screen Updating and Cursor Movement Optimization: A Library Package*, Ken Arnold

CARDS

Here is some useful information. The number in parentheses after the card name is the number of that card in the deck:

Hazard	Repair	Safety
Out of Gas (2)	Gasoline (6)	Extra Tank (1)
Flat Tire (2)	Spare Tire (6)	Puncture Proof (1)
Accident (2)	Repairs (6)	Driving Ace (1)
Stop (4)	Go (14)	Right of Way (1)
Speed Limit (3)	End of Limit (6)	

25 – (10), 50 – (10), 75 – (10), 100 – (12), 200 – (4)

RULES

Object: The point of game is to get a total of 5000 points in several hands. Each hand is a race to put down exactly 700 miles before your opponent does. Beyond the points gained by putting down milestones, there are several other ways of making points.

Overview: The game is played with a deck of 101 cards. *Distance* cards represent a number of miles traveled. They come in denominations of 25, 50, 75, 100, and 200. When one is played, it adds that many miles to the player's trip so far this hand. *Hazard* cards are used to prevent your opponent from putting down Distance cards. They can only be played if your opponent has a *Go* card on top of the Battle pile. The cards are *Out of Gas*, *Accident*, *Flat Tire*, *Speed Limit*, and *Stop*. *Remedy* cards fix problems caused by Hazard cards played on you by your opponent. The cards are *Gasoline*, *Repairs*, *Spare Tire*, *End of Limit*, and *Go*. *Safety* cards prevent your opponent from putting specific Hazard cards on you in the first place. They are *Extra Tank*, *Driving Ace*, *Puncture Proof*, and *Right of Way*, and there are only one of each in the deck.

Board Layout: The board is split into several areas. From top to bottom, they are: **SAFETY AREA** (unlabeled): This is where the safeties will be placed as they are played. **HAND**: These are the cards in your hand. **BATTLE**: This is the Battle pile. All the Hazard and Remedy Cards are played here, except the *Speed Limit* and *End of Limit* cards. Only the top card is displayed, as it is the only effective one. **SPEED**: The Speed pile. The *Speed Limit* and *End of Limit* cards are played here to control the speed at which the player is allowed to put down miles. **MILEAGE**: Miles are placed here. The total of the numbers shown here is the distance traveled so far.

Play: The first pick alternates between the two players. Each turn usually starts with a pick from the deck. The player then plays a card, or if this is not possible or desirable, discards one. Normally, a play or discard of a single card constitutes a turn. If the card played is a safety, however, the same player takes another turn immediately.

This repeats until one of the players reaches 700 points or the deck runs out. If someone reaces 700, they have the option of going for an *Extension*, which means that the play continues until someone reaches 1000 miles.

Hazard and Remedy Cards: Hazard Cards are played on your opponent's Battle and Speed piles. Remedy Cards are used for undoing the effects of your opponent's nastyness.

Go (Green Light) must be the top card on your Battle pile for you to play any mileage, unless you have played the *Right of Way* card (see below).

Stop is played on your opponent's *Go* card to prevent them from playing mileage until they play a *Go* card.

Speed Limit is played on your opponent's Speed pile. Until they play an *End of Limit* they can only play 25 or 50 mile cards, presuming their *Go* card allows them to do even that.

End of Limit is played on your Speed pile to nullify a *Speed Limit* played by your opponent.

Out of Gas is played on your opponent's *Go* card. They must then play a *Gasoline* card, and then a *Go* card before they can play any more mileage.

Flat Tire is played on your opponent's *Go* card. They must then play a *Spare Tire* card, and then a *Go* card before they can play any more mileage.

Accident is played on your opponent's *Go* card. They must then play a *Repairs* card, and then a *Go* card before they can play any more mileage.

Safety Cards: Safety cards prevent your opponent from playing the corresponding Hazard cards on you for the rest of the hand. It cancels an attack in progress, and *always entitles the player to an extra turn.*

Right of Way prevents your opponent from playing both *Stop* and *Speed Limit* cards on you. It also acts as a permanent *Go* card for the rest of the hand, so you can play mileage as long as there is not a Hazard card on top of your Battle pile. In this case only, your opponent can play Hazard cards directly on a Remedy card besides a Go card.

Extra Tank When played, your opponent cannot play an *Out of Gas* on your Battle Pile.

Puncture Proof When played, your opponent cannot play a *Flat Tire* on your Battle Pile.

Driving Ace When played, your opponent cannot play an *Accident* on your Battle Pile.

Distance Cards: Distance cards are played when you have a *Go* card on your Battle pile, or a Right of Way in your Safety area and are not stopped by a Hazard Card. They can be played in any combination that totals exactly 700 miles, except that *you cannot play more than two 200 mile cards in one hand.* A hand ends whenever one player gets exactly 700 miles or the deck runs out. In that case, play continues until neither someone reaches 700, or neither player can use any cards in their hand. If the trip is completed after the deck runs out, this is called *Delayed Action.*

Coup Fourré: This is a French fencing term for a counter-thrust move as part of a parry to an opponents attack. In Mille Bournes, it is used as follows: If an opponent plays a Hazard card, and you have the corresponding Safety in your hand, you play it immediately, even *before* you draw. This immediately removes the Hazard card from your Battle pile, and protects you from that card for the rest of the game. This gives you more points (see "Scoring" below).

Scoring: Scores are totaled at the end of each hand, whether or not anyone completed the trip. The terms used in the Score window have the following meanings:

Milestones Played: Each player scores as many miles as they played before the trip ended.

Each Safety: 100 points for each safety in the Safety area.

All 4 Safeties: 300 points if all four safeties are played.

Each Coup Fouré: 300 points for each Coup Fouré accomplished.

The following bonus scores can apply only to the winning player.

Trip Completed: 400 points bonus for completing the trip to 700 or 1000.

Safe Trip: 300 points bonus for completing the trip without using any 200 mile cards.

Delayed Action: 300 points bonus for finishing after the deck was exhausted.
Extension: 200 points bonus for completing a 1000 mile trip.
Shut-Out: 500 points bonus for completing the trip before your opponent played any mileage cards.

Running totals are also kept for the current score for each player for the hand (**Hand Total**), the game (**Overall Total**), and number of games won (**Games**).

NAME

monop – Monopoly game

SYNOPSIS

/usr/games/monop [file]

DESCRIPTION

Monop is reminiscent of the Parker Brother's game Monopoly, and monitors a game between 1 to 9 users. It is assumed that the rules of Monopoly are known. The game follows the standard rules, with the exception that, if a property would go up for auction and there are only two solvent players, no auction is held and the property remains unowned.

The game, in effect, lends the player money, so it is possible to buy something which you cannot afford. However, as soon as a person goes into debt, he must "fix the problem", *i.e.*, make himself solvent, before play can continue. If this is not possible, the player's property reverts to his debtee, either a player or the bank. A player can resign at any time to any person or the bank, which puts the property back on the board, unowned.

Any time that the response to a question is a *string*, e.g., a name, place or person, you can type '?' to get a list of valid answers. It is not possible to input a negative number, nor is it ever necessary.

A Summary of Commands:

quit: quit game: This allows you to quit the game. It asks you if you're sure.

print: print board: This prints out the current board. The columns have the following meanings (column headings are the same for the **where**, **own holdings**, and **holdings** commands):

Name	The first ten characters of the name of the square
Own	The *number* of the owner of the property.
Price	The cost of the property (if any)
Mg	This field has a '*' in it if the property is mortgaged
#	If the property is a Utility or Railroad, this is the number of such owned by the owner. If the property is land, this is the number of houses on it.
Rent	Current rent on the property. If it is not owned, there is no rent.

where: where players are: Tells you where all the players are. A '*' indicates the current player.

own holdings:
List your own holdings, *i.e.*, money, get-out-of-jail-free cards, and property.

holdings: holdings list: Look at anyone's holdings. It will ask you whose holdings you wish to look at. When you are finished, type "done".

shell: shell escape: Escape to a shell. When the shell dies, the program continues where you left off.

mortgage:
mortgage property: Sets up a list of mortgageable property, and asks which you wish to mortgage.

unmortgage:

unmortgage property: Unmortgage mortgaged property.

buy: buy houses: Sets up a list of monopolies on which you can buy houses. If there is more than one, it asks you which you want to buy for. It then asks you how many for each piece of property, giving the current amount in parentheses after the property name. If you build in an unbalanced manner (a disparity of more than one house within the same monopoly), it asks you to re-input things.

sell: sell houses: Sets up a list of monopolies from which you can sell houses. it operates in an analogous manner to *buy*

card: card for jail: Use a get-out-of-jail-free card to get out of jail. If you're not in jail, or you don't have one, it tells you so.

pay: pay for jail: Pay $50 to get out of jail, from whence you are put on Just Visiting. Difficult to do if you're not there.

trade: This allows you to trade with another player. It asks you whom you wish to trade with, and then asks you what each wishes to give up. You can get a summary at the end, and, in all cases, it asks for confirmation of the trade before doing it.

resign: Resign to another player or the bank. If you resign to the bank, all property reverts to its virgin state, and get-out-of-jail free cards revert to the deck.

save: save game: Save the current game in a file for later play. You can continue play after saving, either by adding the file in which you saved the game after the *monop* command, or by using the *restore* command (see below). It will ask you which file you wish to save it in, and, if the file exists, confirm that you wish to overwrite it.

restore: restore game: Read in a previously saved game from a file. It leaves the file intact.

roll: Roll the dice and move forward to your new location. If you simply hit the <RETURN> key instead of a command, it is the same as typing *roll*.

AUTHOR

Ken Arnold

FILES

/usr/games/lib/cards.pck	Chance and Community Chest cards

BUGS

No command can be given an argument instead of a response to a query.

NAME

number – convert Arabic numerals to English

SYNOPSIS

/usr/games/number

DESCRIPTION

Number copies the standard input to the standard output, changing each decimal number to a fully spelled out version.

NAME

quiz – test your knowledge

SYNOPSIS

/usr/games/quiz [**–i** file] [**–t**] [category1 category2]

DESCRIPTION

Quiz gives associative knowledge tests on various subjects. It asks items chosen from *category1* and expects answers from *category2*. If no categories are specified, *quiz* gives instructions and lists the available categories.

Quiz tells a correct answer whenever you type a bare newline. At the end of input, upon interrupt, or when questions run out, *quiz* reports a score and terminates.

The **–t** flag specifies 'tutorial' mode, where missed questions are repeated later, and material is gradually introduced as you learn.

The **–i** flag causes the named file to be substituted for the default index file. The lines of these files have the syntax:

```
line      = category newline | category ':' line
category  = alternate | category '|' alternate
alternate = empty | alternate primary
primary   = character | '[' category ']' | option
option    = '{' category '}'
```

The first category on each line of an index file names an information file. The remaining categories specify the order and contents of the data in each line of the information file. Information files have the same syntax. Backslash '\' is used as with *sh*(1) to quote syntactically significant characters or to insert transparent newlines into a line. When either a question or its answer is empty, *quiz* will refrain from asking it.

FILES

/usr/games/quiz.k/*

BUGS

The construct 'a|ab' doesn't work in an information file. Use 'a{b}'.

NAME

rain – animated raindrops display

SYNOPSIS

/usr/games/rain

DESCRIPTION

Rain's display is modeled after the VAX/VMS program of the same name. The terminal has to be set for 9600 baud to obtain the proper effect.

As with all programs that use *termcap*, the TERM environment variable must be set (and exported) to the type of the terminal being used.

FILES

/etc/termcap

AUTHOR

Eric P. Scott

NAME

robots – fight off villainous robots

SYNOPSIS

/usr/games/robots [**–sjta**] [**scorefile**]

DESCRIPTION

Robots pits you against evil robots, who are trying to kill you (which is why they are evil). Fortunately for you, even though they are evil, they are not very bright and have a habit of bumping into each other, thus destroying themselves. In order to survive, you must get them to kill each other off, since you have no offensive weaponry.

Since you are stuck without offensive weaponry, you are endowed with one piece of defensive weaponry: a teleportation device. When two robots run into each other or a junk pile, they die. If a robot runs into you, you die. When a robot dies, you get 10 points, and when all the robots die, you start on the next field. This keeps up until they finally get you.

Robots are represented on the screen by a '+', the junk heaps from their collisions by a '*', and you (the good guy) by a '@'.

The commands are:

h	move one square left
l	move one square right
k	move one square up
j	move one square down
y	move one square up and left
u	move one square up and right
b	move one square down and left
n	move one square down and right
.	(also space) do nothing for one turn
HJKLBNYU	run as far as possible in the given direction
>	do nothing for as long as possible
t	teleport to a random location
w	wait until you die or they all do
q	quit
^L	redraw the screen

All commands can be preceded by a count.

If you use the '**w**' command and survive to the next level, you will get a bonus of 10% for each robot which died after you decided to wait. If you die, however, you get nothing. For all other commands, the program will save you from typos by stopping short of being eaten. However, with '**w**' you take the risk of dying by miscalculation.

Only five scores are allowed per user on the score file. If you make it into the score file, you will be shown the list at the end of the game. If an alternate score file is specified, that will be used instead of the standard file for scores.

The options are

–s Don't play, just show the score file

–j Jump, *i.e.*, when you run, don't show any intermediate positions; only show things at the end. This is useful on slow terminals.

–t	Teleport automatically when you have no other option. This is a little disconcerting until you get used to it, and then it is very nice.
–a	Advance into the higher levels directly, skipping the lower, easier levels.

AUTHOR

Ken Arnold

FILES

/usr/games/lib/robots_roll	the score file

BUGS

Bugs? You *crazy*, man?!?

NAME

rogue – Exploring The Dungeons of Doom

SYNOPSIS

/usr/games/rogue [**–r**] [*save_file*] [**–s**] [**–d**]

DESCRIPTION

Rogue is a computer fantasy game with a new twist. It is crt oriented and the object of the game is to survive the attacks of various monsters and get a lot of gold, rather than the puzzle solving orientation of most computer fantasy games.

To get started you really only need to know two commands. The command **?** will give you a list of the available commands and the command **/** will identify the things you see on the screen.

To win the game (as opposed to merely playing to beat other people high scores) you must locate the Amulet of Yendor which is somewhere below the 20th level of the dungeon and get it out. Nobody has achieved this yet and if somebody does, they will probably go down in history as a hero among heros.

When the game ends, either by your death, when you quit, or if you (by some miracle) manage to win, *rogue* will give you alist of the top-ten scorers. The scoring is based entirely upon how much gold you get. There is a 10% penalty for getting yourself killed.

If *save_file* is specified, rogue will be restored from the specified saved game file. If the **–r** option is used, the save game file is presumed to be the default.

The **–s** option will print out the list of scores.

The **–d** option will kill you and try to add you to the score file.

For more detailed directions, read the document *A Guide to the Dungeons of Doom.*

AUTHORS

Michael C. Toy, Kenneth C. R. C. Arnold, Glenn Wichman

FILES

/usr/games/lib/rogue_roll	Score file
~/rogue.save	Default save file

SEE ALSO

Michael C. Toy and Kenneth C. R. C. Arnold, *A guide to the Dungeons of Doom*

BUGS

Probably infinite. However, that Ice Monsters sometimes transfix you permanently is *not* a bug. It's a feature.

NAME

snake, snscore – display chase game

SYNOPSIS

/usr/games/snake [**–w***n*] [**–l***n*]
/usr/games/snscore

DESCRIPTION

Snake is a display-based game which must be played on a CRT terminal from among those supported by vi(1). The object of the game is to make as much money as possible without getting eaten by the snake. The **–l** and **–w** options allow you to specify the length and width of the field. By default the entire screen (except for the last column) is used.

You are represented on the screen by an I. The snake is 6 squares long and is represented by S's. The money is $, and an exit is #. Your score is posted in the upper left hand corner.

You can move around using the same conventions as vi(1), the h, j, k, and l keys work, as do the arrow keys. Other possibilities include:

sefc	These keys are like hjkl but form a directed pad around the d key.
HJKL	These keys move you all the way in the indicated direction to the same row or column as the money. This does *not* let you jump away from the snake, but rather saves you from having to type a key repeatedly. The snake still gets all his turns.
SEFC	Likewise for the upper case versions on the left.
ATPB	These keys move you to the four edges of the screen. Their position on the keyboard is the mnemonic, e.g. P is at the far right of the keyboard.
x	This lets you quit the game at any time.
p	Points in a direction you might want to go.
w	Space warp to get out of tight squeezes, at a price.
!	Shell escape
^Z	Suspend the snake game, on systems which support it. Otherwise an interactive shell is started up.

To earn money, move to the same square the money is on. A new $ will appear when you earn the current one. As you get richer, the snake gets hungrier. To leave the game, move to the exit (#).

A record is kept of the personal best score of each player. Scores are only counted if you leave at the exit, getting eaten by the snake is worth nothing.

As in pinball, matching the last digit of your score to the number which appears after the game is worth a bonus.

To see who wastes time playing snake, run */usr/games/snscore* .

FILES

/usr/games/lib/snakerawscores	database of personal bests
/usr/games/lib/snake.log	log of games played
/usr/games/busy	program to determine if system is too busy

BUGS

When playing on a small screen, it's hard to tell when you hit the edge of the screen.

The scoring function takes into account the size of the screen. A perfect function to do this equitably has not been devised.

NAME

trek – trekkie game

SYNOPSIS

/usr/games/trek [[**–a**] file]

DESCRIPTION

Trek is a game of space glory and war. Below is a summary of commands. For complete documentation, see *Trek* by Eric Allman.

If a filename is given, a log of the game is written onto that file. If the **–a** flag is given before the filename, that file is appended to, not truncated.

The game will ask you what length game you would like. Valid responses are "short", "medium", and "long". You may also type "restart", which restarts a previously saved game. You will then be prompted for the skill, to which you must respond "novice", "fair", "good", "expert", "commadore", or "impossible". You should normally start out with a novice and work up.

In general, throughout the game, if you forget what is appropriate the game will tell you what it expects if you just type in a question mark.

AUTHOR

Eric Allman

SEE ALSO

/usr/doc/trek

COMMAND SUMMARY

abandon	**ca**pture
cloak **up/down**	
computer request; ...	**da**mages
destruct	**do**ck
help	**i**mpulse course distance
lrscan	**m**ove course distance
phasers **a**utomatic amount	
phasers **m**anual amt1 course1 spread1 ...	
torpedo course [**y**es] angle/**n**o	
ram course distance	**r**est time
shell	**sh**ields **up/d**own
srscan [**y**es/**n**o]	
status	**terminate y**es/**n**o
undock	**v**isual course
warp warp_factor	

NAME

worm – Play the growing worm game

SYNOPSIS

/usr/games/worm [*size*]

DESCRIPTION

In *worm,* you are a little worm, your body is the "o"'s on the screen and your head is the "@". You move with the hjkl keys (as in the game snake). If you don't press any keys, you continue in the direction you last moved. The upper case HJKL keys move you as if you had pressed several (9 for HL and 5 for JK) of the corresponding lower case key (unless you run into a digit, then it stops).

On the screen you will see a digit, if your worm eats the digit is will grow longer, the actual amount longer depends on which digit it was that you ate. The object of the game is to see how long you can make the worm grow.

The game ends when the worm runs into either the sides of the screen, or itself. The current score (how much the worm has grown) is kept in the upper left corner of the screen.

The optional argument, if present, is the initial length of the worm.

BUGS

If the initial length of the worm is set to less than one or more than 75, various strange things happen.

NAME

worms – animate worms on a display terminal

SYNOPSIS

/usr/games/worms [**–field**] [**–length #**] [**–number #**] [**–trail**]

DESCRIPTION

Brian Horn (cithep!bdh) showed me a *TOPS-20* program on the DEC-2136 machine called *WORM*, and suggested that I write a similar program that would run under *Unix*. I did, and no apologies.

–field makes a "field" for the worm(s) to eat; **–trail** causes each worm to leave a trail behind it. You can figure out the rest by yourself.

FILES

/etc/termcap

AUTHOR

Eric P. Scott

SEE ALSO

Snails, by Karl Heuer

BUGS

The lower-right-hand character position will not be updated properly on a terminal that wraps at the right margin.

Terminal initialization is not performed.

NAME

wump – the game of hunt-the-wumpus

SYNOPSIS

/usr/games/wump

DESCRIPTION

Wump plays the game of 'Hunt the Wumpus.' A Wumpus is a creature that lives in a cave with several rooms connected by tunnels. You wander among the rooms, trying to shoot the Wumpus with an arrow, meanwhile avoiding being eaten by the Wumpus and falling into Bottomless Pits. There are also Super Bats which are likely to pick you up and drop you in some random room.

The program asks various questions which you answer one per line; it will give a more detailed description if you want.

This program is based on one described in *People's Computer Company, 2,* 2 (November 1973).

SECTION 7

ICON/UXB OPERATING SYSTEM MISCELLANEOUS COMMANDS

ICON®

NAME

miscellaneous – miscellaneous useful information pages

DESCRIPTION

This section contains miscellaneous documentation, mostly in the area of text processing macro packages for *troff*(1).

ascii	map of ASCII character set
environ	user environment
eqnchar	special character definitions for eqn
hier	file system hierarchy
mailaddr	mail addressing description
man	macros to typeset manual pages
me	macros for formatting papers
ms	macros for formatting manuscripts
term	conventional names for terminals

NAME

ascii – map of ASCII character set

SYNOPSIS

cat /usr/pub/ascii

DESCRIPTION

Ascii is a map of the ASCII character set, to be printed as needed. It contains:

000	nul	001	soh	002	stx	003	etx	004	eot	005	enq	006	ack	007	bel
010	bs	011	ht	012	nl	013	vt	014	np	015	cr	016	so	017	si
020	dle	021	dc1	022	dc2	023	dc3	024	dc4	025	nak	026	syn	027	etb
030	can	031	em	032	sub	033	esc	034	fs	035	gs	036	rs	037	us
040	sp	041	!	042	"	043	#	044	$	045	%	046	&	047	´
050	(	051	)	052	*	053	+	054	,	055	-	056	.	057	/
060	0	061	1	062	2	063	3	064	4	065	5	066	6	067	7
070	8	071	9	072	:	073	;	074	<	075	=	076	>	077	?
100	@	101	A	102	B	103	C	104	D	105	E	106	F	107	G
110	H	111	I	112	J	113	K	114	L	115	M	116	N	117	O
120	P	121	Q	122	R	123	S	124	T	125	U	126	V	127	W
130	X	131	Y	132	Z	133	[	134	\	135	]	136	^	137	_
140	`	141	a	142	b	143	c	144	d	145	e	146	f	147	g
150	h	151	i	152	j	153	k	154	l	155	m	156	n	157	o
160	p	161	q	162	r	163	s	164	t	165	u	166	v	167	w
170	x	171	y	172	z	173	{	174	|	175	}	176	~	177	del

00	nul	01	soh	02	stx	03	etx	04	eot	05	enq	06	ack	07	bel
08	bs	09	ht	0a	nl	0b	vt	0c	np	0d	cr	0e	so	0f	si
10	dle	11	dc1	12	dc2	13	dc3	14	dc4	15	nak	16	syn	17	etb
18	can	19	em	1a	sub	1b	esc	1c	fs	1d	gs	1e	rs	1f	us
20	sp	21	!	22	"	23	#	24	$	25	%	26	&	27	´
28	(	29	)	2a	*	2b	+	2c	,	2d	-	2e	.	2f	/
30	0	31	1	32	2	33	3	34	4	35	5	36	6	37	7
38	8	39	9	3a	:	3b	;	3c	<	3d	=	3e	>	3f	?
40	@	41	A	42	B	43	C	44	D	45	E	46	F	47	G
48	H	49	I	4a	J	4b	K	4c	L	4d	M	4e	N	4f	O
50	P	51	Q	52	R	53	S	54	T	55	U	56	V	57	W
58	X	59	Y	5a	Z	5b	[	5c	\	5d	]	5e	^	5f	_
60	`	61	a	62	b	63	c	64	d	65	e	66	f	67	g
68	h	69	i	6a	j	6b	k	6c	l	6d	m	6e	n	6f	o
70	p	71	q	72	r	73	s	74	t	75	u	76	v	77	w
78	x	79	y	7a	z	7b	{	7c	|	7d	}	7e	~	7f	del

FILES

/usr/pub/ascii

NAME

environ – user environment

SYNOPSIS

extern char **environ;

DESCRIPTION

An array of strings called the 'environment' is made available by *execve*(2) when a process begins. By convention these strings have the form '*name=value*'. The following names are used by various commands:

PATH	The sequence of directory prefixes that *sh, time, nice*(1), etc., apply in searching for a file known by an incomplete path name. The prefixes are separated by ':'. *Login*(1) sets PATH=:/usr/ucb:/bin:/usr/bin.
HOME	A user's login directory, set by *login*(1) from the password file *passwd*(5).
TERM	The kind of terminal for which output is to be prepared. This information is used by commands, such as *nroff* or *plot*(1G), which may exploit special terminal capabilities. See */etc/termcap* (*termcap*(5)) for a list of terminal types.
SHELL	The file name of the users login shell.
TERMCAP	The string describing the terminal in TERM, or the name of the termcap file, see *termcap*(5),*termcap*(3X).
EXINIT	A startup list of commands read by *ex*(1), *edit*(1), and *vi*(1).
USER	The login name of the user.
PRINTER	The name of the default printer to be used by *lpr*(1), *lpq*(1), and *lprm*(1).

Further names may be placed in the environment by the *export* command and 'name=value' arguments in *sh*(1), or by the *setenv* command if you use *csh*(1). Arguments may also be placed in the environment at the point of an *execve*(2). It is unwise to conflict with certain *sh*(1) variables that are frequently exported by '.profile' files: MAIL, PS1, PS2, IFS.

SEE ALSO

csh(1), ex(1), login(1), sh(1), execve(2), system(3), termcap(3X), termcap(5)

NAME

eqnchar – special character definitions for eqn

SYNOPSIS

eqn /usr/pub/eqnchar [files] **| troff** [options]

neqn /usr/pub/eqnchar [files] **| nroff** [options]

DESCRIPTION

Eqnchar contains *troff* and *nroff* character definitions for constructing characters that are not available on the Graphic Systems typesetter. These definitions are primarily intended for use with *eqn* and *neqn*. It contains definitions for the following characters

ciplus	⊕	||	‖	*square*	□
citimes	⊗	*langle*	⟨	*circle*	○
wig	~	*rangle*	⟩	*blot*	■
-wig	≃	*hbar*	ħ	*bullet*	•
>*wig*	≳	*ppd*	⊥	*prop*	∝
<*wig*	≲	<->	↔	*empty*	∅
=*wig*	≅	<=>	⇔	*member*	∈
star	*	|<	≮	*nomem*	∉
bigstar	✱	|>	≯	*cup*	∪
=*dot*	≐	*ang*	∠	*cap*	∩
orsign	∨	*rang*	∟	*incl*	⊑
andsign	∧	*3dot*	⋮	*subset*	⊂
=*del*	≜	*thf*	∴	*supset*	⊃
oppA	∀	*quarter*	¼	*!subset*	⊆
oppE	∃	*3quarter*	¾	*!supset*	⊇
angstrom	Å	*degree*	°		

FILES

/usr/pub/eqnchar

SEE ALSO

troff(1), eqn(1)

NAME

hier – file system hierarchy

DESCRIPTION

The following outline gives a quick tour through a representative directory hierarchy.

```
/       root
/vmunix
/m1unix
/dcunix
        the parts of the kernel binary (UNIX itself)
/lost+found
        directory for connecting detached files for fsck(8)
/dev/   devices (4)
        MAKEDEV
                shell script to create special files
        MAKEDEV.local
                site specific part of MAKEDEV
        console
                main console, tty(4)
        tty*    terminals, tty(4)
        sc*     disks, sc(4)
        ...
/bin/   utility programs, cf /usr/bin/ (1)
        as      assembler
        cc      C compiler executive, cf /lib/c0, /lib/c1, /lib/cpp, /lib/c2
        csh     C shell
        ...
/lib/   object libraries and other stuff, cf /usr/lib/
        libc.a  system calls, standard I/O, etc. (2,3,3S)
        ...
        c0, c1  C compiler proper
        cpp     C preprocessor
        c2      C code improver
        ...
/etc/   essential data and maintenance utilities; sect (8)
        dump    dump program dump(8)
        passwd
                password file, passwd(5)
        group   group file, group(5)
        motd    message of the day, login(1)
        termcap
                description of terminal capabilities, termcap(5)
        ttytype
                table of what kind of terminal is on each port, ttytype(5)
        mtab    mounted file table, mtab(5)
        dumpdates
                dump history, dump(8)
        fstab   file system configuration table fstab(5)
        hosts   host name to network address mapping file, hosts(5)
        networks
                network name to network number mapping file, networks(5)
        protocols
```

```
                protocol name to protocol number mapping file, protocols(5)
        services
                network services definition file, services(5)
        remote  names and description of remote hosts for tip(1C), remote(5)
        phones
                private phone numbers for remote hosts, as described in phones(5)
        ttys    properties of terminals, ttys(5)
        getty   part of login, getty(8)
        init    the parent of all processes, init(8)
        rc      shell program to bring the system up
        rc.local
                site dependent portion of rc
        cron    the clock daemon, cron(8)
        mount   mount(8)
        ...
/sys/   system  source (if available)
        h/      header (include) files
                acct.h  acct(5)
                stat.h  stat(2)
                ...
        sys/    machine independent system source
                init_main.c
                uipc_socket.c
                ufs_syscalls.c
                ...
        conf/   site configuration files
                GENERIC
                ...
        net/    general network source
        netinet/
                DARPA Internet network source
        netimp/
                network code related to use of an IMP
                if_imp.c
                if_imphost.c
                if_imphost.h
                ...
        /tmp/   temporary files, usually on a fast device, cf /usr/tmp/
                e*      used by ed(1)
                ctm*    used by cc(1)
                ...
        /usr/   general-pupose directory, usually a mounted file system
                adm/    administrative information
                        wtmp    login history, utmp(5)
                        messages
                                hardware error messages
                        tracct  phototypesetter accounting, troff(1)
                        lpacct  line printer accounting lpr(1)
                        vaacct, vpacct
                                varian and versatec accounting vpr(1), vtroff(1), pac(8)
        /usr    /bin
                utility programs, to keep /bin/ small
                tmp/    temporaries, to keep /tmp/ small
```

```
          stm*    used by sort(1)
          raster  used by plot(1G)
  dict/   word lists, etc.
          words   principal word list, used by look(1)
          spellhist
                  history file for spell(1)
  games/
          adventure
          lib/    library of stuff for the games
                  quiz.k/
                          what quiz(6) knows
                          index   category index
                          africa  countries and capitals
                          ...
                  ...
          ...
  include/
          standard #include files
          a.out.h
                  object file layout, a.out(5)
          stdio.h standard I/O, intro(3S)
          math.h
                  (3M)
          ...
          sys/    system-defined layouts, cf /sys/h
          net/    symbolic link to sys/net
          machine/
                  symbolic link to sys/machine
          ...
  lib/    object libraries and stuff, to keep /lib/ small
          atrun   scheduler for at(1)
          lint/   utility files for lint
                  lint[12]
                          subprocesses for lint(1)
                  llib-lc dummy declarations for /lib/libc.a, used by lint(1)
                  llib-lm dummy declarations for /lib/libc.m
                  ...
          struct/
                  passes of struct(1)
          ...
          tmac/   macros for troff(1)
                  tmac.an
                          macros for man(7)
                  tmac.s  macros for ms(7)
                  ...
          font/   fonts for troff(1)
                  ftR     Times Roman
                  ftB     Times Bold
                  ...
          uucp/   programs and data for uucp(1C)
                  L.sys   remote system names and numbers
                  uucico  the real copy program
                  ...
```

```
                units   conversion tables for units(1)
                eign    list of English words to be ignored by ptx(1)
/usr/   man/
        volume 1 of this manual, man(1)
                man0/   general
                        intro   introduction to volume 1, ms(7) format
                        xx      template for manual page
                man1/   chapter 1
                        as.1
                        mount.1m
                        ...
                ...
                cat1/   preformatted pages for section 1
                ...
        msgs/   messages, cf msgs(1)
                bounds
                        highest and lowest message
        new/    binaries of new versions of programs
        preserve/
                editor temporaries preserved here after crashes/hangups
        public/
                binaries of user programs - write permission to everyone
        spool/  delayed execution files
                at/     used by at(1)
                lpd/    used by lpr(1)
                        lock    present when line printer is active
                        cf*     copy of file to be printed, if necessary
                        df*     daemon control file, lpd(8)
                        tf*     transient control file, while lpr is working
                uucp/   work files and staging area for uucp(1C)
                        LOGFILE
                                summary log
                        LOG.* log file for one transaction
                mail/   mailboxes for mail(1)
                        name  mail file for user name
                        name.lock
                                lock file while name is receiving mail
                secretmail/
                        like mail/
                uucp/   work files and staging area for uucp(1C)
                        LOGFILE
                                summary log
                        LOG.* log file for one transaction
                        mqueue/
                                mail queue for sendmail(8)
        wd      initial working directory of a user, typically wd is the user's login name
                .profile
                        set environment for sh(1), environ(7)
                .project
                        what you are doing (used by ( finger(1) )
                .cshrc  startup file for csh(1)
                .exrc   startup file for ex(1)
                .plan   what your short-term plans are (used by finger(1) )
                .netrc  startup file for various network programs
                .msgsrc
                        startup file for msgs(1)
                .mailrc
                        startup file for mail(1)
                calendar
                        user's datebook for calendar(1)
        doc/    papers, mostly in volume 2 of this manual, typically in ms(7) format
                as/     assembler manual
                ...
/usr/   src/
        source programs for utilities, etc.
        (on machines with source code)
        normally /usr/src will not exist.
        bin/    source of commands in /bin
                as/     assembler
                ar.c    source for ar(1)
                ...
        usr.bin/
                source for commands in /usr/bin
                troff/  source for nroff and troff(1)
                        font/   source for font tables, /usr/lib/font/
                                ftR.c   Roman
                                ...
                        term/   terminal characteristics tables, /usr/lib/term/
                                tab300.c
                                        DASI 300
                                ...
                        ...
        ucb     source for programs in /usr/ucb
        games/
                source for /usr/games
        lib/    source for programs and archives in /lib
                libc/   C runtime library
                        csu/    startup and wrapup routines needed with every C
                                program
                                crt0.s  regular startup
                                mcrt0.s
                                        modified startup for cc -p
                        sys/    system calls (2)
                                access.s
                                brk.s
                                ...
                stdio/  standard I/O functions (3S)
                        fgets.c
                        fopen.c
                        ...
                gen/    other functions in (3)
                        abs.c
                        ...
                net/    network functions in (3N)
                        gethostbyname.c
                        ...
```

local/	source which isn't normally distributed	
new/	source for new versions of commands and library routines	
old/	source for old versions of commands and library routines	
ucb/	binaries of programs developed at UCB	
	...	
	edit	editor for beginners
	ex	command editor for experienced users
	...	
	mail	mail reading/sending subsystem
	man	on line documentation
	...	
	pi	Pascal translator
	px	Pascal interpreter
	...	
	vi	visual editor

SEE ALSO

ls(1), apropos(1), whatis(1), whereis(1), finger(1), which(1), find(1), grep(1)

BUGS

The position of files is subject to change without notice.

NAME

mailaddr – mail addressing description

DESCRIPTION

Mail addresses are based on the ARPANET protocol listed at the end of this manual page. These addresses are in the general format

    user@domain

where a domain is a hierarchical dot separated list of subdomains. For example, the address

    eric@monet.Berkeley.ARPA

is normally interpreted from right to left: the message should go to the ARPA name tables (which do not correspond exactly to the physical ARPANET), then to the Berkeley gateway, after which it should go to the local host monet. When the message reaches monet it is delivered to the user "eric".

Unlike some other forms of addressing, this does not imply any routing. Thus, although this address is specified as an ARPA address, it might travel by an alternate route if that was more convenient or efficient. For example, at Berkeley the associated message would probably go directly to monet over the Ethernet rather than going via the Berkeley ARPANET gateway.

Abbreviation. Under certain circumstances it may not be necessary to type the entire domain name. In general anything following the first dot may be omitted if it is the same as the domain from which you are sending the message. For example, a user on "calder.Berkeley.ARPA" could send to "eric@monet" without adding the ".Berkeley.ARPA" since it is the same on both sending and receiving hosts.

Certain other abbreviations may be permitted as special cases. For example, at Berkeley ARPANET hosts can be referenced without adding the ".ARPA" as long as their names do not conflict with a local host name.

Compatibility. Certain old address formats are converted to the new format to provide compatibility with the previous mail system. In particular,

    host:user

is converted to

    user@host

to be consistent with the *rcp*(1C) command.

Also, the syntax:

    host!user

is converted to:

    user@host.UUCP

This is normally converted back to the "host!user" form before being sent on for compatibility with older UUCP hosts.

The current implementation is not able to route messages automatically through the UUCP network. Until that time you must explicitly tell the mail system which hosts to send your message through to get to your final destination.

Case Distinctions. Domain names (i.e., anything after the "@" sign) may be given in any mixture of upper and lower case with the exception of UUCP hostnames. Most hosts accept any mixture of case in user names, with the notable exception of MULTICS sites.

Differences with ARPA Protocols. Although the UNIX addressing scheme is based on the ARPA mail addressing protocols, there are some significant differences.

At the time of this writing the only "top level" domain defined by ARPA is the ".ARPA" domain itself. This is further restricted to having only one level of host specifier. That is, the only addresses that ARPA accepts at this time must be in the format "user@host.ARPA" (where "host" is one word). In particular, addresses such as:

eric@monet.Berkeley.ARPA

are not currently legal under the ARPA protocols. For this reason, these addresses are converted to a different format on output to the ARPANET, typically:

eric%monet@Berkeley.ARPA

Route-addrs. Under some circumstances it may be necessary to route a message through several hosts to get it to the final destination. Normally this routing is done automatically, but sometimes it is desirable to route the message manually. An address that shows these relays are termed "route-addrs." These use the syntax:

<@hosta,@hostb:user@hostc>

This specifies that the message should be sent to hosta, from there to hostb, and finally to hostc. This path is forced even if there is a more efficient path to hostc.

Route-addrs occur frequently on return addresses, since these are generally augmented by the software at each host. It is generally possible to ignore all but the "user@host" part of the address to determine the actual sender.

Postmaster. Every site is required to have a user or user alias designated "postmaster" to which problems with the mail system may be addressed.

CSNET. Messages to CSNET sites can be sent to "user.host@UDel-Relay".

BERKELEY

The following comments apply only to the Berkeley environment.

Host Names. Many of the old familiar host names are being phased out. In particular, single character names as used in Berknet are incompatible with the larger world of which Berkeley is now a member. For this reason the following names are being obsoleted. You should notify any correspondents of your new address as soon as possible.

OLD	NEW		
p	ucbcad	r arpavax	ucbarpa
v csvax	ucbernie	n	ucbkim
y	ucbcory		
		j ingvax	ucbingres

The old addresses will be rejected as unknown hosts sometime in the near future.

What's My Address? If you are on a local machine, say monet, your address is

yourname@monet.Berkeley.ARPA

However, since most of the world does not have the new software in place yet, you will have to give correspondents slightly different addresses. From the ARPANET, your address would be:

yourname%monet@Berkeley.ARPA

From UUCP, your address would be:

ucbvax!yourname%monet

Computer Center. The Berkeley Computer Center is in a subdomain of Berkeley. Messages to the computer center should be addressed to:

user%host.CC@Berkeley.ARPA

The alternate syntax:

user@host.CC

may be used if the message is sent from inside Berkeley.

For the time being Computer Center hosts are known within the Berkeley domain, i.e., the ".CC" is optional. However, it is likely that this situation will change with time as both the Computer Science department and the Computer Center grow.

Bitnet. Hosts on bitnet may be accessed using:

user@host.BITNET

SEE ALSO

mail(1), sendmail(8); Crocker, D. H., *Standard for the Format of Arpa Internet Text Messages,* RFC822.

NAME

man – macros to typeset manual

SYNOPSIS

nroff −man file ...

troff −man file ...

DESCRIPTION

These macros are used to lay out pages of this manual. A skeleton page may be found in the file /usr/man/man0/xx.

Any text argument *t* may be zero to six words. Quotes may be used to include blanks in a 'word'. If *text* is empty, the special treatment is applied to the next input line with text to be printed. In this way .I may be used to italicize a whole line, or .SM followed by .B to make small bold letters.

A prevailing indent distance is remembered between successive indented paragraphs, and is reset to default value upon reaching a non-indented paragraph. Default units for indents *i* are ens.

Type font and size are reset to default values before each paragraph, and after processing font and size setting macros.

These strings are predefined by **−man**:

*R	'®', '(Reg)' in *nroff*.
*S	Change to default type size.

FILES

/usr/lib/tmac/tmac.an
/usr/man/man0/xx

SEE ALSO

troff(1), man(1)

BUGS

Relative indents don't nest.

REQUESTS

Request	Cause Break	If no Argument	Explanation
.B *t*	no	*t*=n.t.l.*	Text *t* is bold.
.BI *t*	no	*t*=n.t.l.	Join words of *t* alternating bold and italic.
.BR *t*	no	*t*=n.t.l.	Join words of *t* alternating bold and Roman.
.DT	no	.5i 1i...	Restore default tabs.
.HP *i*	yes	*i*=p.i.*	Set prevailing indent to *i*. Begin paragraph with hanging indent.
.I *t*	no	*t*=n.t.l.	Text *t* is italic.
.IB *t*	no	*t*=n.t.l.	Join words of *t* alternating italic and bold.
.IP *x i*	yes	*x*=""	Same as .TP with tag *x*.
.IR *t*	no	*t*=n.t.l.	Join words of *t* alternating italic and Roman.
.LP	yes	-	Same as .PP.
.PD *d*	no	*d*=.4v	Interparagraph distance is *d*.
.PP	yes	-	Begin paragraph. Set prevailing indent to .5i.

.RE	yes	-	End of relative indent. Set prevailing indent to amount of starting .RS.
.RB *t*	no	*t*=n.t.l.	Join words of *t* alternating Roman and bold.
.RI *t*	no	*t*=n.t.l.	Join words of *t* alternating Roman and italic.
.RS *i*	yes	*i*=p.i.	Start relative indent, move left margin in distance *i*. Set prevailing indent to .5i for nested indents.
.SH *t*	yes	*t*=n.t.l.	Subhead.
.SM *t*	no	*t*=n.t.l.	Text *t* is small.
.TH *n c x v m*	yes	-	Begin page named *n* of chapter *c; x* is extra commentary, e.g. 'local', for page foot center; *v* alters page foot left, e.g. '4th Berkeley Distribution'; *m* alters page head center, e.g. 'Brand X Programmer's Manual'. Set prevailing indent and tabs to .5i.
.TP *i*	yes	*i*=p.i.	Set prevailing indent to *i*. Begin indented paragraph with hanging tag given by next text line. If tag doesn't fit, place it on separate line.

* n.t.l. = next text line; p.i. = prevailing indent

NAME

me – macros for formatting papers

SYNOPSIS

nroff –me [options] file ...
troff –me [options] file ...

DESCRIPTION

This package of *nroff* and *troff* macro definitions provides a canned formatting facility for technical papers in various formats. When producing 2-column output on a terminal, filter the output through *col*(1).

The macro requests are defined below. Many *nroff* and *troff* requests are unsafe in conjunction with this package, however these requests may be used with impunity after the first .pp:

.bp	begin new page
.br	break output line here
.sp n	insert n spacing lines
.ls n	(line spacing) n=1 single, n=2 double space
.na	no alignment of right margin
.ce n	center next n lines
.ul n	underline next n lines
.sz +n	add n to point size

Output of the *eqn, neqn, refer,* and *tbl*(1) preprocessors for equations and tables is acceptable as input.

FILES

/usr/lib/tmac/tmac.e
/usr/lib/me/*

SEE ALSO

eqn(1), troff(1), refer(1), tbl(1)
–me Reference Manual, Eric P. Allman
Writing Papers with Nroff Using –me

REQUESTS

In the following list, "initialization" refers to the first .pp, .lp, .ip, .np, .sh, or .uh macro. This list is incomplete; see *The –me Reference Manual* for interesting details.

Request	Initial Value	Cause Break	Explanation
.(c	-	yes	Begin centered block
.(d	-	no	Begin delayed text
.(f	-	no	Begin footnote
.(l	-	yes	Begin list
.(q	-	yes	Begin major quote
.(x *x*	-	no	Begin indexed item in index *x*
.(z	-	no	Begin floating keep
.)c	-	yes	End centered block
.)d	-	yes	End delayed text
.)f	-	yes	End footnote
.)l	-	yes	End list

.)q	-	yes	End major quote
.)x	-	yes	End index item
.)z	-	yes	End floating keep
.++ *m H*	-	no	Define paper section. *m* defines the part of the paper, and can be **C** (chapter), **A** (appendix), **P** (preliminary, e.g., abstract, table of contents, etc.), **B** (bibliography), **RC** (chapters renumbered from page one each chapter), or **RA** (appendix renumbered from page one).
.+c *T*	-	yes	Begin chapter (or appendix, etc., as set by .++). *T* is the chapter title.
.1c	1	yes	One column format on a new page.
.2c	1	yes	Two column format.
.EN	-	yes	Space after equation produced by *eqn* or *neqn*.
.EQ *x y*	-	yes	Precede equation; break out and add space. Equation number is *y*. The optional argument *x* may be *I* to indent equation (default), *L* to left-adjust the equation, or *C* to center the equation.
.TE	-	yes	End table.
.TH	-	yes	End heading section of table.
.TS *x*	-	yes	Begin table; if *x* is *H* table has repeated heading.
.ac *A N*	-	no	Set up for ACM style output. *A* is the Author's name(s), *N* is the total number of pages. Must be given before the first initialization.
.b *x*	no	no	Print *x* in boldface; if no argument switch to boldface.
.ba *+n*	0	yes	Augments the base indent by *n*. This indent is used to set the indent on regular text (like paragraphs).
.bc	no	yes	Begin new column
.bi *x*	no	no	Print *x* in bold italics (nofill only)
.bx *x*	no	no	Print *x* in a box (nofill only).
.ef *'x'y'z'*	''''	no	Set even footer to x y z
.eh *'x'y'z'*	''''	no	Set even header to x y z
.fo *'x'y'z'*	''''	no	Set footer to x y z
.hx	-	no	Suppress headers and footers on next page.
.he *'x'y'z'*	''''	no	Set header to x y z
.hl	-	yes	Draw a horizontal line
.i *x*	no	no	Italicize *x;* if *x* missing, italic text follows.
.ip *x y*	no	yes	Start indented paragraph, with hanging tag *x*. Indentation is *y* ens (default 5).
.lp	yes	yes	Start left-blocked paragraph.
.lo	-	no	Read in a file of local macros of the form **.****x*. Must be given before initialization.
.np	1	yes	Start numbered paragraph.
.of *'x'y'z'*	''''	no	Set odd footer to x y z
.oh *'x'y'z'*	''''	no	Set odd header to x y z
.pd	-	yes	Print delayed text.
.pp	no	yes	Begin paragraph. First line indented.
.r	yes	no	Roman text follows.
.re	-	no	Reset tabs to default values.
.sc	no	no	Read in a file of special characters and diacritical marks. Must be given before initialization.
.sh *n x*	-	yes	Section head follows, font automatically bold. *n* is level of section, *x* is title of section.
.sk	no	no	Leave the next page blank. Only one page is remembered ahead.
.sz *+n*	10p	no	Augment the point size by *n* points.
.th	no	no	Produce the paper in thesis format. Must be given before initialization.
.tp	no	yes	Begin title page.
.u *x*	-	no	Underline argument (even in *troff*). (Nofill only).

.uh	-	yes	Like .sh but unnumbered.
.xp *x*	-	no	Print index *x*.

NAME

ms – text formatting macros

SYNOPSIS

nroff –ms [options] file ...
troff –ms [options] file ...

DESCRIPTION

This package of *nroff* and *troff* macro definitions provides a formatting facility for various styles of articles, theses, and books. When producing 2-column output on a terminal or line-printer, or when reverse line motions are needed, filter the output through *col*(1). All external –ms macros are defined below. Many *nroff* and *troff* requests are unsafe in conjunction with this package. However, the first four requests below may be used with impunity after initialization, and the last two may be used even before initialization:

.bp	begin new page
.br	break output line
.sp n	insert n spacing lines
.ce n	center next n lines
.ls n	line spacing: n=1 single, n=2 double space
.na	no alignment of right margin

Font and point size changes with \f and \s are also allowed; for example, "\fIword\fR" will italicize *word.* Output of the *tbl, eqn,* and *refer*(1) preprocessors for equations, tables, and references is acceptable as input.

FILES

/usr/lib/tmac/tmac.x
/usr/lib/ms/x.???

SEE ALSO

eqn(1), refer(1), tbl(1), troff(1)

REQUESTS

Macro Name	Initial Value	Break? Reset?	Explanation
.AB *x*	–	y	begin abstract; if *x*=no don't label abstract
.AE	–	y	end abstract
.AI	–	y	author's institution
.AM	–	n	better accent mark definitions
.AU	–	y	author's name
.B *x*	–	n	embolden *x*; if no *x*, switch to boldface
.B1	–	y	begin text to be enclosed in a box
.B2	–	y	end boxed text and print it
.BT	date	n	bottom title, printed at foot of page
.BX *x*	–	n	print word *x* in a box
.CM	if t	n	cut mark between pages
.CT	–	y,y	chapter title: page number moved to CF (TM only)
.DA *x*	if n	n	force date *x* at bottom of page; today if no *x*
.DE	–	y	end display (unfilled text) of any kind
.DS *x y*	I	y	begin display with keep; *x*=I,L,C,B; *y*=indent
.ID *y*	8n,.5i	y	indented display with no keep; *y*=indent

.LD	–	y	left display with no keep
.CD	–	y	centered display with no keep
.BD	–	y	block display; center entire block
.EF *x*	–	n	even page footer *x* (3 part as for .tl)
.EH *x*	–	n	even page header *x* (3 part as for .tl)
.EN	–	y	end displayed equation produced by *eqn*
.EQ *x y*	–	y	break out equation; *x*=L,I,C; *y*=equation number
.FE	–	n	end footnote to be placed at bottom of page
.FP	–	n	numbered footnote paragraph; may be redefined
.FS *x*	–	n	start footnote; *x* is optional footnote label
.HD	undef	n	optional page header below header margin
.I *x*	–	n	italicize *x*; if no *x*, switch to italics
.IP *x y*	–	y,y	indented paragraph, with hanging tag *x*; *y*=indent
.IX *x y*	–	y	index words *x y* and so on (up to 5 levels)
.KE	–	n	end keep of any kind
.KF	–	n	begin floating keep; text fills remainder of page
.KS	–	y	begin keep; unit kept together on a single page
.LG	–	n	larger; increase point size by 2
.LP	–	y,y	left (block) paragraph.
.MC *x*	–	y,y	multiple columns; *x*=column width
.ND *x*	if t	n	no date in page footer; *x* is date on cover
.NH *x y*	–	y,y	numbered header; *x*=level, *x*=0 resets, *x*=S sets to *y*
.NL	10p	n	set point size back to normal
.OF *x*	–	n	odd page footer *x* (3 part as for .tl)
.OH *x*	–	n	odd page header *x* (3 part as for .tl)
.P1	if TM	n	print header on 1st page
.PP	–	y,y	paragraph with first line indented
.PT	- % -	n	page title, printed at head of page
.PX *x*	–	y	print index (table of contents); *x*=no suppresses title
.QP	–	y,y	quote paragraph (indented and shorter)
.R	on	n	return to Roman font
.RE	5n	y,y	retreat: end level of relative indentation
.RP *x*	–	n	released paper format; *x*=no stops title on 1st page
.RS	5n	y,y	right shift: start level of relative indentation
.SH	–	y,y	section header, in boldface
.SM	–	n	smaller; decrease point size by 2
.TA	8n,5n	n	set tabs to 8n 16n ... (nroff) 5n 10n ... (troff)
.TC *x*	–	y	print table of contents at end; *x*=no suppresses title
.TE	–	y	end of table processed by *tbl*
.TH	–	y	end multi-page header of table
.TL	–	y	title in boldface and two points larger
.TM	off	n	UC Berkeley thesis mode
.TS *x*	–	y,y	begin table; if *x*=H table has multi-page header
.UL *x*	–	n	underline *x*, even in *troff*
.UX *x*	–	n	UNIX; trademark message first time; *x* appended
.XA *x y*	–	y	another index entry; *x*=page or no for none; *y*=indent
.XE	–	y	end index entry (or series of .IX entries)
.XP	–	y,y	paragraph with first line exdented, others indented
.XS *x y*	–	y	begin index entry; *x*=page or no for none; *y*=indent
.1C	on	y,y	one column format, on a new page
.2C	–	y,y	begin two column format
.]-	–	n	beginning of *refer* reference
.[0	–	n	end of unclassifiable type of reference

.[N	–	n	N= 1:journal-article, 2:book, 3:book-article, 4:report

REGISTERS

Formatting distances can be controlled in –ms by means of built-in number registers. For example, this sets the line length to 6.5 inches:

```
.nr LL 6.5i
```

Here is a table of number registers and their default values:

Name	Register Controls	Takes Effect	Default
PS	point size	paragraph	10
VS	vertical spacing	paragraph	12
LL	line length	paragraph	6i
LT	title length	next page	same as LL
FL	footnote length	next .FS	5.5i
PD	paragraph distance	paragraph	1v (if n), .3v (if t)
DD	display distance	displays	1v (if n), .5v (if t)
PI	paragraph indent	paragraph	5n
QI	quote indent	next .QP	5n
FI	footnote indent	next .FS	2n
PO	page offset	next page	0 (if n), ~1i (if t)
HM	header margin	next page	1i
FM	footer margin	next page	1i
FF	footnote format	next .FS	0 (1, 2, 3 available)

When resetting these values, make sure to specify the appropriate units. Setting the line length to 7, for example, will result in output with one character per line. Setting FF to 1 suppresses footnote superscripting; setting it to 2 also suppresses indentation of the first line; and setting it to 3 produces an .IP-like footnote paragraph.

Here is a list of string registers available in –ms; they may be used anywhere in the text:

Name	String's Function
*Q	quote (" in *nroff,* " in *troff*)
*U	unquote (" in *nroff,* " in *troff*)
*–	dash (-- in *nroff,* — in *troff*)
*(MO	month (month of the year)
*(DY	day (current date)
**	automatically numbered footnote
*´	acute accent (before letter)
*`	grave accent (before letter)
*^	circumflex (before letter)
*,	cedilla (before letter)
*:	umlaut (before letter)
*~	tilde (before letter)

When using the extended accent mark definitions available with .AM, these strings should come after, rather than before, the letter to be accented.

BUGS

Floating keeps and regular keeps are diverted to the same space, so they cannot be mixed together with predictable results.

NAME

prec – C precedence chart

SYNOPSIS

cat /usr/pub/prec

DESCRIPTION

Prec is a C precedence chart, to be printed as needed. It contains:

Operator	Associativity
() [] -> .	left to right
! ~ ++ -- - (type) * & sizeof	right to left
* / %	left to right
+ -	left to right
<< >>	left to right
< <= > >=	left to right
== !=	left to right
&	left to right
^	left to right
|	left to right
&&	left to right
||	left to right
?:	right to left
= += -= etc.	right to left
,	left to right

FILES

/usr/pub/prec

NAME

term – conventional names for terminals

DESCRIPTION

Certain commands use these terminal names. They are maintained as part of the shell environment (see *sh*(1),*environ*(7)).

adm3a	Lear Seigler Adm-3a
2621	Hewlett-Packard HP262? series terminals
hp	Hewlett-Packard HP264? series terminals
c100	Human Designed Systems Concept 100
h19	Heathkit H19
mime	Microterm mime in enhanced ACT IV mode
1620	DIABLO 1620 (and others using HyType II)
300	DASI/DTC/GSI 300 (and others using HyType I)
33	TELETYPE® Model 33
37	TELETYPE Model 37
43	TELETYPE Model 43
735	Texas Instruments TI735 (and TI725)
745	Texas Instruments TI745
dumb	terminals with no special features
dialup	a terminal on a phone line with no known characteristics
network	a terminal on a network connection with no known characteristics
4014	Tektronix 4014
vt52	Digital Equipment Corp. VT52

The list goes on and on. Consult /etc/termcap (see *termcap*(5)) for an up-to-date and locally correct list.

Commands whose behavior may depend on the terminal either consult TERM in the environment, or accept arguments of the form **–Tterm,** where *term* is one of the names given above.

SEE ALSO

stty(1), tabs(1), plot(1G), sh(1), environ(7) ex(1), clear(1), more(1), ul(1), tset(1), termcap(5), termcap(3X), ttytype(5)
troff(1) for *nroff*

BUGS

The programs that ought to adhere to this nomenclature do so only fitfully.

SECTION 8

ICON/UXB OPERATING SYSTEM MAINTENANCE

ICON®

NAME

intro – introduction to system maintenance and operation commands

DESCRIPTION

This section contains information related to system operation and maintenance. In particular, commands used to create new file systems, *newfs*, *mkfs*, and verify the integrity of the file systems, *fsck*, are described here. The section *dkfmt* should be consulted when formatting disks. The section *crash* should be consulted in understanding how to interpret system crash dumps.

LIST OF PROGRAMS

Program	*Appears on Page*	*Description*
ac	ac.8	login accounting
accton	sa.8	system accounting
adduser	adduser.8	procedure for adding new users
analyze	analyze.8	Virtual UNIX postmortem crash analyzer
arcv	arcv.8	convert archives to new format
badsect	badsect.8	create files to contain bad sectors
bugfiler	bugfiler.8	file bug reports in folders automatically
catman	catman.8	create the cat files for the manual
chown	chown.8	change owner
comsat	comsat.8c	biff server
config	config.8	build system configuration files
crash	crash.8v	what happens when the system crashes
cron	cron.8	clock daemon
dmesg	dmesg.8	collect system diagnostic messages to form error log
drtest	drtest.8	standalone disk test program
dump	dump.8	incremental file system dump
dumpfs	dumpfs.8	dump file system information
edquota	edquota.8	edit user quotas
fastboot	fastboot.8	reboot/halt the system without checking the disks
fasthalt	fastboot.8	reboot/halt the system without checking the disks
fsck	fsck.8	file system consistency check and interactive repair
ftpd	ftpd.8c	DARPA Internet File Transfer Protocol server
gettable	gettable.8c	get NIC format host tables from a host
getty	getty.8	set terminal mode
halt	halt.8	stop the processor
htable	htable.8	convert NIC standard format host tables
ifconfig	ifconfig.8c	configure network interface parameters
implog	implog.8c	IMP log interpreter
implogd	implogd.8c	IMP logger process
init	init.8	process control initialization
kgmon	kgmon.8	generate a dump of the operating systems profile buffers
lpc	lpc.8	line printer control program
lpd	lpd.8	line printer daemon
makedev	makedev.8	make system special files
makekey	makekey.8	generate encryption key
mkfs	mkfs.8	construct a file system
mklost+found	mklost+found.8	make a lost+found directory for fsck
mknod	mknod.8	build special file
mkproto	mkproto.8	construct a prototype file system
mount	mount.8	mount and dismount file system

newfs	newfs.8	construct a new file system
pac	pac.8	printer/ploter accounting information
quot	quot.8	summarize file system ownership
quotacheck	quotacheck.8	file system quota consistency checker
quotaoff	quotaon.8	turn file system quotas on and off
quotaon	quotaon.8	turn file system quotas on and off
rc	rc.8	command script for auto-reboot and daemons
rdump	rdump.8c	file system dump across the network
reboot	reboot.8	UNIX bootstrapping procedures
renice	renice.8	alter priority of running processes
repquota	repquota.8	summarize quotas for a file system
restore	restore.8	incremental file system restore
rexecd	rexecd.8c	remote execution server
rlogind	rlogind.8c	remote login server
rmt	rmt.8c	remote magtape protocol module
route	route.8c	manually manipulate the routing tables
routed	routed.8c	network routing daemon
rrestore	rrestore.8c	restore a file system dump across the network
rshd	rshd.8c	remote shell server
rwhod	rwhod.8c	system status server
sa	sa.8	system accounting
sendmail	sendmail.8	send mail over the internet
shutdown	shutdown.8	close down the system at a given time
sticky	sticky.8	executable files with persistent text
swapon	swapon.8	specify additional device for paging and swapping
sync	sync.8	update the super block
syslog	syslog.8	log systems messages
telnetd	telnetd.8c	DARPA TELNET protocol server
tftpd	tftpd.8c	DARPA Trivial File Transfer Protocol server
trpt	trpt.8c	transliterate protocol trace
umount	mount.8	mount and dismount file system
update	update.8	periodically update the super block
uuclean	uuclean.8c	uucp spool directory clean-up
uusnap	uusnap.8c	show snapshot of the UUCP system
vipw	vipw.8	edit the password file

NAME

ac – login accounting

SYNOPSIS

/etc/ac [**–w** wtmp] [**–p**] [**–d**] [people] ...

DESCRIPTION

Ac produces a printout giving connect time for each user who has logged in during the life of the current *wtmp* file. A total is also produced. **–w** is used to specify an alternate *wtmp* file. **–p** prints individual totals; without this option, only totals are printed. **–d** causes a printout for each midnight to midnight period. Any *people* will limit the printout to only the specified login names. If no *wtmp* file is given, */usr/adm/wtmp* is used.

The accounting file */usr/adm/wtmp* is maintained by *init* and *login*. Neither of these programs creates the file, so if it does not exist no connect-time accounting is done. To start accounting, it should be created with length 0. On the other hand if the file is left undisturbed it will grow without bound, so periodically any information desired should be collected and the file truncated.

FILES

/usr/adm/wtmp

SEE ALSO

init(8), sa(8), login(1), utmp(5).

NAME

adduser – procedure for adding new users

DESCRIPTION

A new user must choose a login name, which must not already appear in */etc/passwd.* An account can be added by editing a line into the passwd file; this must be done with the password file locked e.g. by using *vipw*(8).

A new user is given a group and user id. User id's should be distinct across a system, since they are used to control access to files. Typically, users working on similar projects will be put in the same group. Thus at UCB we have groups for system staff, faculty, graduate students, and a few special groups for large projects. System staff is group "10" for historical reasons, and the super-user is in this group.

A skeletal account for a new user "ernie" would look like:

```
ernie::235:20:& Kovacs,508E,7925,6428202:/mnt/grad/ernie:/bin/csh
```

The first field is the login name "ernie". The next field is the encrypted password which is not given and must be initialized using *passwd*(1). The next two fields are the user and group id's. Traditionally, users in group 20 are graduate students and have account names with numbers in the 200's. The next field gives information about ernie's real name, office and office phone and home phone. This information is used by the *finger*(1) program. From this information we can tell that ernie's real name is "Ernie Kovacs" (the & here serves to repeat "ernie" with appropriate capitalization), that his office is 508 Evans Hall, his extension is x2-7925, and this his home phone number is 642-8202. You can modify the *finger*(1) program if necessary to allow different information to be encoded in this field. The UCB version of finger knows several things particular to Berkeley – that phone extensions start "2–", that offices ending in "E" are in Evans Hall and that offices ending in "C" are in Cory Hall.

The final two fields give a login directory and a login shell name. Traditionally, user files live on a file system which has the machines single letter *net*(1) address as the first of two characters. Thus on the Berkeley CS Department VAX, whose Berknet address is "csvax" abbreviated "v" the user file systems are mounted on "/va", "/vb", etc. On each such filesystem there are subdirectories there for each group of users, i.e.: "/va/staff" and "/vb/prof". This is not strictly necessary but keeps the number of files in the top level directories reasonably small.

The login shell will default to "/bin/sh" if none is given. Most users at Berkeley choose "/bin/csh" so this is usually specified here.

It is useful to give new users some help in getting started, supplying them with a few skeletal files such as *.profile* if they use "/bin/sh", or *.cshrc* and *.login* if they use "/bin/csh". The directory "/usr/skel" contains skeletal definitions of such files. New users should be given copies of these files which, for instance, arrange to use *tset*(1) automatically at each login.

FILES

/etc/passwd	password file
/usr/skel	skeletal login directory

SEE ALSO

passwd(1), finger(1), chsh(1), chfn(1), passwd(5), vipw(8)

BUGS

User information should be stored in its own data base separate from the password file.

NAME

binstl – program to install bootloader on disk

DESCRIPTION

Binstl is a program executable only from standalone mode. It is used to install the boot loader, *bload* (8), in the first 35 sectors of a disk (hard or floppy). This must be done before the disk can be used by the system.

For a detailed description on how to install the loader refer to the *MPS020-2 Operating System Installation Guide*

FILES

/stand/binstl

SEE ALSO

bload(8), standalone(8),

NAME

bload – program to load standalone programs

DESCRIPTION

Bload is a program executable only from standalone mode. It is used to load and run standalone programs.

Bload is actually the heart of standalone mode. Bload is run when the automatic reboot procedure is overridden. The user is given the option of which device to load from, and *bload* is loaded from that device and run. *Bload* issues its prompt:

```
ICON loader -- Version 1.0
Load:
```

From here the standalone programs are run, single user mode is entered, and multi-user mode is entered.

Arguments to load may be:

- [device]:[partition] (e.g. 3:2 or 0:a)
- >standalone program (e.g. stand/fsck,stand/mkfs)
- -s (single user mode)

For a more detailed description of *bload*, see the *MPS020-2 Operating System Installation Guide*

FILES

/stand/bload

SEE ALSO

standalone(8),

NAME

bugfiler – file bug reports in folders automatically

SYNOPSIS

bugfiler [mail directory]

DESCRIPTION

Bugfiler is a program to automatically intercept bug reports, summarize them and store them in the appropriate sub directories of the mail directory specified on the command line or the (system dependent) default. It is designed to be compatible with the Rand MH mail system. *Bugfiler* is normally invoked by the mail delivery program through *aliases*(5) with a line such as the following in /usr/lib/aliases.

```
bugs:"|bugfiler /usr/bugs/mail"
```

It reads the message from the standard input or the named file, checks the format and returns mail acknowledging receipt or a message indicating the proper format. Valid reports are then summarized and filed in the appropriate folder. Users can then log onto the system and check the summary file for bugs that pertain to them. Bug reports are submitted in RFC822 format and must contain the following header lines:

```
Date: <date the report is received>
From: <valid return address>
Subject: <short summary of the problem>
Index: <source directory>/<source file> <version> [Fix]
```

In addition, the body of the message must contain a line which begins with "Description:" followed by zero or more lines describing the problem in detail and a line beginning with "Repeat-By:" followed by zero or more lines describing how to repeat the problem. If the keyword 'Fix' is specified in the 'Index' line, then there must also be a line beginning with "Fix:" followed by a diff of the old and new source files or a description of what was done to fix the problem.

The 'Index' line is the key to the filing mechanism. The source directory name must match one of the folder names in the mail directory. The message is then filed in this folder and a line appended to the summary file in the following format:

```
<folder name>/<message number>        <Index info>
                                <Subject info>
```

FILES

/usr/new/lib/mh/deliver	mail delivery program
/usr/new/lib/mh/unixtomh	converts unix mail format to mh format
maildir/.ack	the message sent in acknowledgement
maildir/.format	the message sent when format errors are detected
maildir/summary	the summary file
maildir/Bf??????	temporary copy of the input message
maildir/Rp??????	temporary file for the reply message.

SEE ALSO

mh(1), newaliases(1), aliases(5)

BUGS

Since mail can be forwarded in a number of different ways, *bugfiler* does not recognize forwarded mail and will reply/complain to the forwarder instead of the original sender unless there is a 'Reply-To' field in the header.

Duplicate messages should be discarded or recognized and put somewhere else.

NAME

catman – create the cat files for the manual

SYNOPSIS

/etc/catman [**–p**] [**–n**] [**–w**] [sections]

DESCRIPTION

Catman creates the preformatted versions of the on-line manual from the nroff input files. Each manual page is examined and those whose preformatted versions are missing or out of date are recreated. If any changes are made, *catman* will recreate the **/usr/lib/whatis** database.

If there is one parameter not starting with a '–', it is take to be a list of manual sections to look in. For example

catman 123

will cause the updating to only happen to manual sections 1, 2, and 3.

Options:

–n	prevents creations of **/usr/lib/whatis**.
–p	prints what would be done instead of doing it.
–w	causes only the **/usr/lib/whatis** database to be created. No manual reformatting is done.

FILES

/usr/man/man?/*.*	raw (nroff input) manual sections
/usr/man/cat?/*.*	preformatted manual pages
/usr/lib/makewhatis	commands to make whatis database

SEE ALSO

man(1)

BUGS

Acts oddly on nights with full moons.

NAME

chown – change owner

SYNOPSIS

/etc/chown [**–f**] owner file ...

DESCRIPTION

Chown changes the owner of the *files* to *owner*. The owner may be either a decimal UID or a login name found in the password file.

Only the super-user can change owner, in order to simplify accounting procedures. No errors are reported when the **–f** (force) option is given.

FILES

/etc/passwd

SEE ALSO

chgrp(1), chown(2), passwd(5), group(5)

NAME

cleanlpd – clean line printer daemon environment

SYNOPSIS

/usr/lib/cleanlpd

DESCRIPTION

Cleanlpd cleans the line printer daemon lock files and is normally invoked when the line printer daemon becomes out of sync with it's operating environment. If the restart command in *lpc(1)* fails to start the daemon, use *cleanlpd* to restart the daemon from scratch. This command should not normally need to be used, but will clean up after a change in the /etc/printcap file or when for some unknown reason *lpr(1)* fails to work.

FILES

/usr/spool/*	spool directories
/dev/lp*	line printer devices
/dev/printer	socket for local requests

SEE ALSO

lpc(8), pac(1), lpr(1), lpq(1), lprm(1), printcap(5)
4.2BSD Line Printer Spooler Manual

NAME

clri – clear i-node

SYNOPSIS

/etc/clri filesystem i-number ...

DESCRIPTION

N.B.: *Clri* is obsoleted for normal file system repair work by *fsck*(8).

Clri writes zeros on the i-nodes with the decimal *i-numbers* on the *filesystem.* After *clri,* any blocks in the affected file will show up as 'missing' in an *icheck*(8) of the *filesystem.*

Read and write permission is required on the specified file system device. The i-node becomes allocatable.

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

SEE ALSO

icheck(8)

BUGS

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

NAME

comsat – biff server

SYNOPSIS

/etc/comsat

DESCRIPTION

Comsat is the server process which listens for reports of incoming mail and notifies users if they have requested this service. *Comsat* listens on a datagram port associated with the "biff" service specification (see *services*(5)) for one line messages of the form

    user@mailbox-offset

If the *user* specified is logged in to the system and the associated terminal has the owner execute bit turned on (by a "biff y"), the *offset* is used as a seek offset into the appropriate mailbox file and the first 7 lines or 560 characters of the message are printed on the user's terminal. Lines which appear to be part of the message header other than the "From", "To", "Date", or "Subject" lines are not included in the displayed message.

FILES

/etc/utmp	to find out who's logged on and on what terminals

SEE ALSO

biff(1)

BUGS

The message header filtering is prone to error.

Users should be notified of mail which arrives on other machines than the one they are currently logged in to.

The notification should appear in a separate window so it does not mess up the screen.

NAME

copy – standalone copy program

DESCRIPTION

Copy is a program executable only from standalone mode. It is used to copy from one area on peripheral storage to another. These areas may be on the same or separate devices.

When *copy* is run it returns the following prompt:

```
Sanyo/Icon copy program -- Version 1.0
From:
```

Copy expects the following:

```
[device]:[partition or file]
```

in response. Currently *copy* supports the following 4 devices:

0 = first hard disk
1 = second hard disk
2 = floppy disk
3 = cassette drive

Partitions are the letters 'a' through 'h', used when you are accessing disks, and files are one up numbers, starting at '0', when you are accessing tape. Partitions on the disk are the same as those set up by *dkfmt*(8).

Thus an area specifier would be: 0:a, 2:b or 3:10.

For an example of the use of *copy*, refer to the *MPS020-2 Operating System Installation Guide*

FILES

/stand/copy

SEE ALSO

standalone(8),
dkfmt(8),

NAME

crash – what happens when the system crashes

DESCRIPTION

This section explains what happens when the system crashes.

When the system crashes voluntarily it prints a message of the form

panic: why i gave up the ghost

on the console, then sits there waiting for someone to press reset. When reset is pressed, by turning the system keyswitch to reset, the bootstrap loader starts executing reboot. The loader detects that this is a boot from a crash, or other unusual circumstance, asks the operator if he wishes to dump the system core to a diagnostic tape, responds to that choice, then continues the reboot procedure as described in *reboot*(8). Unless some unexpected inconsistency is encountered in the state of the file systems due to hardware or software failure the system will then resume multi-user operations.

The system has a large number of internal consistency checks; if one of these fails, then it will panic with a very short message indicating which one failed.

There are two basic types of system failures, those resulting from hardware failure, and those resulting from internal kernel inconsistency. Hardware errors are the most common (which is not to say that they are common), and can manifest themselves in a number of different ways.

System crashes should be rare occurences. Regular or frequent crashes could indicate a persistent hardware problem. Contact your customer service representative in such instances.

SEE ALSO

reboot(8), loader(8)

NAME
cron – clock daemon

SYNOPSIS
/etc/cron

DESCRIPTION
Cron executes commands at specified dates and times according to the instructions in the file /usr/lib/crontab. Since *cron* never exits, it should only be executed once. This is best done by running *cron* from the initialization process through the file /etc/rc; see *init*(8).

Crontab consists of lines of six fields each. The fields are separated by spaces or tabs. The first five are integer patterns to specify the minute (0-59), hour (0-23), day of the month (1-31), month of the year (1-12), and day of the week (1-7 with 1=Monday). Each of these patterns may contain a number in the range above; two numbers separated by a minus meaning a range inclusive; a list of numbers separated by commas meaning any of the numbers; or an asterisk meaning all legal values. The sixth field is a string that is executed by the Shell at the specified times. A percent character in this field 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.

Crontab is examined by *cron* every minute.

FILES
/usr/lib/crontab

NAME

dcheck – file system directory consistency check

SYNOPSIS

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

DESCRIPTION

N.B.: *Dcheck* is obsoleted for normal consistency checking by *fsck*(8).

Dcheck reads the directories in a file system and compares the link-count in each i-node with the number of directory entries by which it is referenced. If the file system is not specified, a set of default file systems is checked.

The **–i** flag is followed by a list of i-numbers; when one of those i-numbers turns up in a directory, the number, the i-number of the directory, and the name of the entry are reported.

The program is fastest if the raw version of the special file is used, since the i-list is read in large chunks.

FILES

Default file systems vary with installation.

SEE ALSO

fsck(8), icheck(8), fs(5), clri(8), ncheck(8)

DIAGNOSTICS

When a file turns up for which the link-count and the number of directory entries disagree, the relevant facts are reported. Allocated files which have 0 link-count and no entries are also listed. The only dangerous situation occurs when there are more entries than links; if entries are removed, so the link-count drops to 0, the remaining entries point to thin air. They should be removed. When there are more links than entries, or there is an allocated file with neither links nor entries, some disk space may be lost but the situation will not degenerate.

BUGS

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

Dcheck is obsoleted by *fsck* and remains for historical reasons.

NAME

dkfmt – standalone disk formatter

DESCRIPTION

Dkfmt is a program executable only from standalone mode. It is used to format the hard disk(s) at installation time and to format floppy disks.

For a detailed description of dkfmt, refer to the *MPS020-2 Operating System Installation Guide.*

FILES

/stand/dkfmt

SEE ALSO

standalone(8),

NOTE

Formatting a floppy is virtually identical to formatting a hard disk, with the exception of the size and configuration table.

NAME

dmesg – collect system diagnostic messages to form error log

SYNOPSIS

/etc/dmesg [–]

DESCRIPTION

Dmesg looks in a system buffer for recently printed diagnostic messages and prints them on the standard output. The messages are those printed by the system when device (hardware) errors occur and (occasionally) when system tables overflow non-fatally. If the – flag is given, then *dmesg* computes (incrementally) the new messages since the last time it was run and places these on the standard output. This is typically used with *cron*(8) to produce the error log */usr/adm/messages* by running the command

```
/etc/dmesg – >> /usr/adm/messages
```

every 10 minutes.

FILES

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

BUGS

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

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

NAME
doscopyd – MPS/DOS file copy daemon

SYNOPSIS
/etc/doscopyd [line]

DESCRIPTION
Doscopyd is a server process which should be started in the rc.local file. It provides support for copying files to and from MPS/DOS. The line argument may be specified to override the default data stream, which is /dev/mtty7. It is not normally necessary to specify this parameter.

Please refer to the "Technical Note on Dosc and Related Software" for a full description of *doscopyd* and its MPS/DOS clients, *UCOPY* and *TAR*.

SEE ALSO
dosc(1), "Technical Note on Dosc and Related Software"

NAME

dosdisk – program to create and display information for MPS/DOS vdisks

SYNOPSIS

/etc/dosdisk [-v "volname"] [-c clustersize] [-r #rootdirents] [path] [size]

DESCRIPTION

Dosdisk is used by the system administrator to add vdisks for use by the MPS/DOS environment, or to display the vdisks currently defined. If *dosdisk* is entered without parameters, it will list all currently defined vdisks. If parameters are specified, there are two types of vdisks which can be identified to the system. The first is a "dos partition" type vdisk, which is supported for backward compatibility. In this case the path must be either /dev/sc0d or /dev/sc1d, and none of the other parameters can be specified. The path name is simply added to /etc/dosdisks. The other type of vdisk is a MPS/UX file to be used as a vdisk. In this case, **path** specifies the pathname of a file which will be created to serve as the vdisk. This file cannot currently exist. The size must also be specified, and may be any value from 512K to 512M. (See NOTE.) The size may be specified as a number, a number followed by "k" which multiplies the value given by 1024, or a number followed by "m" which multiplies the value given by 1024*1024. The -v, -c and -r options allow specification of the volume name, cluster size, and number of directory entries in the root directory for the newly created vdisk. Users should not normally specify the clustersize for vdisks larger than 32M.

Please refer to the "Technical Note on Dosc and Related Software" for a full description of MPS/DOS vdisk support.

FILES

/etc/dosdisks vdisk description file

SEE ALSO

dosc(1), "Technical Note on Dosc and Related Software"

NOTES

Please note that in release MPS/UX release 2.15, not all sizes of vdisks have been tested. The following sizes of vdisks have been tested and appear to work successfully:

512K through 256M
500M

The next release may support up to 1G vdisks, and all sizes up to the max will be supported. Users needing sizes which have not been tested are welcome to try them; they should work.

NAME

dosprint – MPS/DOS spooler daemon

SYNOPSIS

/etc/dosprint [line [delay]]

DESCRIPTION

Dosprint is a server process which should be started in the rc.local file. It provides spooled printer support for up to eight virtual printers to MPS/DOS users. The optional arguments are to override the default input stream (/dev/mtty6), and the default timeout delay (10 seconds). If only the delay is to be changed, /dev/mtty6 must be specified as the first parameter. It should not normally be necessary to specify either of these parameters.

Please refer to the "Technical Note on Dosc and Related Software" for a full description of MPS/DOS spooled printer support.

FILES

/etc/dosprinters	MPS/DOS printer description file

SEE ALSO

dosc(1), "Technical Note on Dosc and Related Software"

NAME

dump – incremental file system dump

SYNOPSIS

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

DESCRIPTION

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

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

f Place the dump on the next *argument* file instead of the tape. If the name of the file is "–", *dump* writes to standard output.

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

s The size of the dump tape is specified in feet. The number of feet is taken from the next *argument.* When the specified size is reached, *dump* will wait for reels to be changed. The default tape size is 2300 feet.

d The density of the tape, expressed in BPI, is taken from the next *argument.* This is used in calculating the amount of tape used per reel. The default is 1600.

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

w Is like W, but prints only those filesystems which need to be dumped.

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

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

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

Since making a dump involves a lot of time and effort for full dumps, *dump* checkpoints itself at the start of each tape volume. If writing that volume fails for some reason, *dump* will, with operator permission, restart itself from the checkpoint after the old tape has been rewound and removed, and a new tape has been mounted.

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

Now a short suggestion on how to perform dumps. Start with a full level 0 dump

```
dump 0un
```

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

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

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

FILES

/dev/rrp1g	default filesystem to dump from
/dev/rmt8	default tape unit to dump to
/etc/ddate	old format dump date record (obsolete after **–J** option)
/etc/dumpdates	new format dump date record
/etc/fstab	dump table: file systems and frequency
/etc/group	to find group *operator*

SEE ALSO

restore(8), dump(5), fstab(5)

DIAGNOSTICS

Many, and verbose.

BUGS

Sizes are based on 1600 BPI blocked tape; the raw magtape device has to be used to approach these densities. Fewer than 32 read errors on the filesystem are ignored. Each reel requires a new process, so parent processes for reels already written just hang around until the entire tape is written.

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

NAME

dumpfs – dump file system information

SYNOPSIS

dumpfs *filesys*|*device*

DESCRIPTION

Dumpfs prints out the super block and cylinder group information for the file system or special device specified. The listing is very long and detailed. This command is useful mostly for finding out certain file system information such as the file system block size and minimum free space percentage.

SEE ALSO

fs(5), disktab(5), tunefs(8), newfs(8), fsck(8)

NAME

edquota – edit user quotas

SYNOPSIS

edquota [**–p** *proto-user*] *users...*

DESCRIPTION

Edquota is a quota editor. One or more users may be specified on the command line. For each user a temporary file is created with an ASCII representation of the current disc quotas for that user and an editor is then invoked on the file. The quotas may then be modified, new quotas added, etc. Upon leaving the editor, *edquota* reads the temporary file and modifies the binary quota files to reflect the changes made.

If the **–p** option is specified, *edquota* will duplicate the quotas of the prototypical user specified for each user specified. This is the normal mechanism used to initialize quotas for groups of users.

The editor invoked is *vi*(1) unless the environment variable EDITOR specifies otherwise.

Only the super-user may edit quotas.

FILES

quotas	at the root of each file system with quotas
/etc/fstab	to find file system names and locations

SEE ALSO

quota(1), quota(2), quotacheck(8), quotaon(8), repquota(8)

DIAGNOSTICS

Various messages about inaccessible files; self-explanatory.

BUGS

The format of the temporary file is inscrutable.

NAME

fastboot, fasthalt – reboot/halt the system without checking the disks

SYNOPSIS

/etc/fastboot [*boot-options*]
/etc/fasthalt [*halt-options*]

DESCRIPTION

Fastboot and *fasthalt* are shell scripts which reboot and halt the system without checking the file systems. This is done by creating a file */fastboot*, then invoking the *reboot* program. The system startup script, */etc/rc*, looks for this file and, if present, skips the normal invocation of *fsck*(8).

SEE ALSO

halt(8), reboot(8), rc(8)

NAME

fsck – file system consistency check and interactive repair

SYNOPSIS

reboot the machine
type SPACE bar to override the automatic reboot
boot from device 0
>**stand/fsck** **–p** filesystem ...

>**stand/fsck** [**–y**] [**–n**] filesystem ...

DESCRIPTION

The first form of *fsck* preens the specified file systems.

The system takes care that only a restricted class of innocuous inconsistencies can happen unless hardware or software failures intervene. These are limited to the following:

Unreferenced inodes

Link counts in inodes too large

Missing blocks in the free list

Blocks in the free list also in files

Counts in the super-block wrong

These are the only inconsistencies which *fsck* with the **–p** option will correct; if it encounters other inconsistencies, it exits with an abnormal return status. For each corrected inconsistency one or more lines will be printed identifying the file system on which the correction will take place, and the nature of the correction. After successfully correcting a file system, *fsck* will print the number of files on that file system and the number of used and free blocks.

Without the **–p** option, *fsck* audits and interactively repairs inconsistent conditions for file systems. If the file system is inconsistent the operator is prompted for concurrence before each correction is attempted. It should be noted that a number of the corrective actions which are not fixable under the **–p** option will result in some loss of data. The amount and severity of data lost may be determined from the diagnostic output. The default action for each consistency correction is to wait for the operator to respond **yes** or **no**. If the operator does not have write permission *fsck* will default to a **–n** action.

Fsck has more consistency checks than its predecessors *check, dcheck, fcheck,* and *icheck* combined.

The following flags are interpreted by *fsck*.

–y Assume a yes response to all questions asked by *fsck;* this should be used with great caution as this is a free license to continue after essentially unlimited trouble has been encountered.

–n Assume a no response to all questions asked by *fsck;* do not open the file system for writing.

Inconsistencies checked are as follows:

1. Blocks claimed by more than one inode or the free list.
2. Blocks claimed by an inode or the free list outside the range of the file system.
3. Incorrect link counts.
4. Size checks:
   Directory size not of proper format.
5. Bad inode format.
6. Blocks not accounted for anywhere.
7. Directory checks:
   File pointing to unallocated inode.
   Inode number out of range.
8. Super Block checks:

   More blocks for inodes than there are in the file system.
9. Bad free block list format.
10. Total free block and/or free inode count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with the operator's concurrence, reconnected by placing them in the **lost+found** directory. The name assigned is the inode number. The only restriction is that the directory **lost+found** must pre-exist in the root of the filesystem 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.

FILES

/etc/fstab	contains default list of file systems to check.

DIAGNOSTICS

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

SEE ALSO

fstab(5), fs(5), newfs(8), mkfs(8), crash(8V), reboot(8)

EXAMPLE

reboot the machine
type SPACE bar to override the automatic reboot
boot from device 0
>**stand/fsck –p** 0a 0b 0g

BUGS

Inode numbers for . and .. in each directory should be checked for validity.

There should be some way to start a **fsck –p** at pass *n*.

NAME

ftpd – DARPA Internet File Transfer Protocol server

SYNOPSIS

/etc/ftpd [**–d**] [**–l**] [**–t**timeout]

DESCRIPTION

Ftpd is the DARPA Internet File Transfer Protototocol server process. The server uses the TCP protocol and listens at the port specified in the "ftp" service specification; see *services*(5).

If the **–d** option is specified, each socket created will have debugging turned on (SO_DEBUG). With debugging enabled, the system will trace all TCP packets sent and received on a socket. The program *trpt*(8C) may then be used to interpret the packet traces.

If the –l option is specified, each ftp session is logged on the standard output. This allows a line of the form '/etc/ftpd -l > /tmp/ftplog" to be used to conveniently maintain a log of ftp sessions.

The ftp server will timeout an inactive session after 60 seconds. If the **–t** option is specified, the inactivity timeout period will be set to *timeout.*

The ftp server currently supports the following ftp requests; case is not distinguished.

Request	**Description**
ACCT	specify account (ignored)
ALLO	allocate storage (vacuously)
APPE	append to a file
CWD	change working directory
DELE	delete a file
HELP	give help information
LIST	give list files in a directory ("ls -lg")
MODE	specify data transfer *mode*
NLST	give name list of files in directory ("ls")
NOOP	do nothing
PASS	specify password
PORT	specify data connection port
QUIT	terminate session
RETR	retrieve a file
RNFR	specify rename-from file name
RNTO	specify rename-to file name
STOR	store a file
STRU	specify data transfer *structure*
TYPE	specify data transfer *type*
USER	specify user name
XCUP	change to parent of current working directory
XCWD	change working directory
XMKD	make a directory
XPWD	print the current working directory
XRMD	remove a directory

The remaining ftp requests specified in Internet RFC 765 are recognized, but not implemented.

Ftpd interprets file names according to the "globbing" conventions used by *csh*(1). This allows users to utilize the metacharacters "*?[]{}~".

Ftpd authenticates users according to three rules.

1) The user name must be in the password data base, */etc/passwd*, and not have a null password. In this case a password must be provided by the client before any file operations may be performed.

2) The user name must not appear in the file */etc/ftpusers*.

3) If the user name is "anonymous" or "ftp", an anonymous ftp account must be present in the password file (user "ftp"). In this case the user is allowed to log in by specifying any password (by convention this is given as the client host's name).

In the last case, *ftpd* takes special measures to restrict the client's access privileges. The server performs a *chroot*(2) command to the home directory of the "ftp" user. In order that system security is not breached, it is recommended that the "ftp" subtree be constructed with care; the following rules are recommended.

~ftp) Make the home directory owned by "ftp" and unwritable by anyone.

~ftp/bin)
Make this directory owned by the super-user and unwritable by anyone. The program *ls*(1) must be present to support the list commands. This program should have mode 111.

~ftp/etc)
Make this directory owned by the super-user and unwritable by anyone. The files *passwd*(5) and *group*(5) must be present for the *ls* command to work properly. These files should be mode 444.

~ftp/pub)
Make this directory mode 777 and owned by "ftp". Users should then place files which are to be accessible via the anonymous account in this directory.

SEE ALSO

ftp(1C),

BUGS

There is no support for aborting commands.

The anonymous account is inherently dangerous and should avoided when possible.

The server must run as the super-user to create sockets with privileged port numbers. It maintains an effective user id of the logged in user, reverting to the super-user only when binding addresses to sockets. The possible security holes have been extensively scrutinized, but are possibly incomplete.

NAME

gettable – get NIC format host tables from a host

SYNOPSIS

/etc/gettable *host*

DESCRIPTION

Gettable is a simple program used to obtain the NIC standard host tables from a "nicname" server. The indicated *host* is queried for the tables. The tables, if retrieved, are placed in the file *hosts.txt*.

Gettable operates by opening a TCP connection to the port indicated in the service specification for "nicname". A request is then made for "ALL" names and the resultant information is placed in the output file.

Gettable is best used in conjunction with the *htable*(8) program which converts the NIC standard file format to that used by the network library lookup routines.

SEE ALSO

intro(3N), htable(8)

BUGS

Should allow requests for only part of the database.

NAME

getty – set terminal mode

SYNOPSIS

/etc/getty [type]

DESCRIPTION

Getty is invoked by *init*(8) immediately after a terminal is opened, following the making of a connection. While reading the name *getty* attempts to adapt the system to the speed and type of terminal being used.

Init calls *getty* with an argument specified by the *ttys* file entry for the terminal line. The argument can be used to make *getty* treat the line specially. This argument is used as an index into the *gettytab*(5) database, to determine the characteristics of the line. If there is no argument, or there is no such table, the **default** table is used. If there is no **/etc/gettytab** a set of system defaults is used. If indicated by the table located, *getty* will clear the terminal screen, print a banner heading, and prompt for a login name. Usually either the banner of the login prompt will include the system hostname. Then the user's name is read, a character at a time. If a null character is received, it is assumed to be the result of the user pushing the 'break' ('interrupt') key. The speed is usually then changed and the 'login:' is typed again; a second 'break' changes the speed again and the 'login:' is typed once more. Successive 'break' characters cycle through the some standard set of speeds.

The user's name is terminated by a new-line or carriage-return character. The latter results in the system being set to treat carriage returns appropriately (see *tty*(4)).

The user's name is scanned to see if it contains any lower-case alphabetic characters; if not, and if the name is nonempty, the system is told to map any future upper-case characters into the corresponding lower-case characters.

Finally, login is called with the user's name as argument.

Most of the default actions of *getty* can be circumvented, or modified, by a suitable *gettytab* table.

Getty can be set to timeout after some interval, which will cause dial up lines to hang up if the login name is not entered reasonably quickly.

FILES

/etc/gettytab

SEE ALSO

gettytab(5), init(8), login(1), ioctl(2), tty(4), ttys(5).

BUGS

Currently, the format of **/etc/ttys** limits the permitted table names to a single character, this should be expanded.

/etc/ttys should be replaced completely.

NAME

halt – stop the processor

SYNOPSIS

/etc/halt [**–n**] [**–q**] [**–y**]

DESCRIPTION

Halt writes out sandbagged information to the disks and then stops the processor.

The **–n** option prevents the sync before stopping. The **–q** option causes a quick halt, no graceful shutdown is attempted. The **–y** option is needed if you are trying to halt the system from a dialup.

SEE ALSO

reboot(8), shutdown(8)

NAME

htable – convert NIC standard format host tables

SYNOPSIS

/etc/htable *file*

DESCRIPTION

Htable is used to convert host files in the format specified in Internet RFC 810 to the format used by the network library routines. Three files are created as a result of running *htable*: *hosts*, *networks*, and *gateways*. The *hosts* file is used by the *gethostent*(3N) routines in mapping host names to addresses. The *networks* file is used by the *getnetent*(3N) routines in mapping network names to numbers. The *gateways* file is used by the routing daemon in identifying "passive" Internet gateways; see *routed*(8C) for an explanation.

If any of the files *localhosts*, *localnetworks*, or *localgateways* are present in the current directory, the file's contents is prepended to the output file without interpretation. This allows sites to maintain local aliases and entries which are not normally present in the master database.

Htable is best used in conjunction with the *gettable*(8C) program which retrieves the NIC database from a host.

SEE ALSO

intro(3N), gettable(8C)

BUGS

Does not properly calculate the *gateways* file.

NAME

icheck – file system storage consistency check

SYNOPSIS

/etc/icheck [**–s**] [**–b** numbers] [filesystem]

DESCRIPTION

N.B.: *Icheck* is obsoleted for normal consistency checking by *fsck*(8).

Icheck examines a file system, builds a bit map of used blocks, and compares this bit map against the free list maintained on the file system. If the file system is not specified, a set of default file systems is checked. The normal output of *icheck* includes a report of

> The total number of files and the numbers of regular, directory, block special and character special files.
>
> The total number of blocks in use and the numbers of single-, double-, and triple-indirect blocks and directory blocks.
>
> The number of free blocks.
>
> The number of blocks missing; i.e. not in any file nor in the free list.

The **–s** option causes *icheck* to ignore the actual free list and reconstruct a new one by rewriting the super-block of the file system. The file system should be dismounted while this is done; if this is not possible (for example if the root file system has to be salvaged) care should be taken that the system is quiescent and that it is rebooted immediately afterwards so that the old, bad in-core copy of the super-block will not continue to be used. Notice also that the words in the super-block which indicate the size of the free list and of the i-list are believed. If the super-block has been curdled these words will have to be patched. The **–s** option causes the normal output reports to be suppressed.

Following the **–b** option is a list of block numbers; whenever any of the named blocks turns up in a file, a diagnostic is produced.

Icheck is faster if the raw version of the special file is used, since it reads the i-list many blocks at a time.

FILES

Default file systems vary with installation.

SEE ALSO

fsck(8), dcheck(8), ncheck(8), fs(5), clri(8)

DIAGNOSTICS

For duplicate blocks and bad blocks (which lie outside the file system) *icheck* announces the difficulty, the i-number, and the kind of block involved. If a read error is encountered, the block number of the bad block is printed and *icheck* considers it to contain 0. 'Bad freeblock' means that a block number outside the available space was encountered in the free list. '*n* dups in free' means that *n* blocks were found in the free list which duplicate blocks either in some file or in the earlier part of the free list.

BUGS

Since *icheck* is inherently two-pass in nature, extraneous diagnostics may be produced if applied to active file systems.

It believes even preposterous super-blocks and consequently can get core images.

The system should be fixed so that the reboot after fixing the root file system is not necessary.

NAME

ifconfig – configure network interface parameters

SYOPNSIS

/etc/ifconfig interface [*address*] [*parameters*]

DESCRIPTION

Ifconfig is used to assign an address to a network interface and/or configure network interface parameters. *Ifconfig* must be used at boot time to define the network address of each interface present on a machine; it may also be used at a later time to redefine an interface's address. The *interface* parameter is a string of the form "name unit", e.g. "en0", while the address is either a host name present in the host name data base, *hosts*(5), or a DARPA Internet address expressed in the Internet standard "dot notation".

The following parameters may be set with *ifconfig*:

up	Mark an interface "up".
down	Mark an interface "down". When an interface is marked "down", the system will not attempt to transmit messages through that interface.
trailers	Enable the use of a "trailer" link level encapsulation when sending (default). If a network interface supports *trailers*, the system will, when possible, encapsulate outgoing messages in a manner which minimizes the number of memory to memory copy operations performed by the receiver.
–trailers	Disable the use of a "trailer" link level encapsulation.
arp	Enable the use of the Address Resolution Protocol in mapping between network level addresses and link level addresses (default). This is currently implemented for mapping between DARPA Internet addreses and 10Mb/s Ethernet addresses.
–arp	Disable the use of the Address Resolution Protocol.

Ifconfig displays the current configuration for a network interface when no optional parameters are supplied.

Only the super-user may modify the configuration of a network interface.

DIAGNOSTICS

Messages indicating the specified interface does not exit, the requested address is unknown, the user is not privileged and tried to alter an interface's configuration.

SEE ALSO

rc(8), intro(4N), netstat(1)

NAME

implog – IMP log interpreter

SYNOPSIS

/etc/implog [**–D**] [**–f**] [**–c**] [**–l** [*link*]] [**–h** *host#*] [**–i** *imp#*] [**–t** *message-type*]

DESCRIPTION

Implog is program which interprets the message log produced by *implogd*(8C).

If no arguments are specified, *implog* interprets and prints every message present in the message file. Options may be specified to force printing only a subset of the logged messages.

–D Do not show data messages.

–f Follow the logging process in action. This flags causes *implog* to print the current contents of the log file, then check for new logged messages every 5 seconds.

–c In addition to printing any data messages logged, show the contents of the data in hexadecimal bytes.

–l [*link#*]
Show only those messages received on the specified "link". If no value is given for the link, the link number of the IP protocol is assumed.

–h *host#*
Show only those messages received from the specified host. (Usually specified in conjunction with an imp.)

–i *imp#*
Show only those messages received from the specified imp.

–t *message-type*
Show only those messages received of the specified message type.

SEE ALSO

imp(4P), implogd(8C)

BUGS

Can not specify multiple hosts, imps, etc. Can not follow reception of messages without looking at those currently in the file.

NAME

implogd – IMP logger process

SYNOPSIS

/etc/implogd [**–d**]

DESCRIPTION

Implogd is program which logs messages from the IMP, placing them in the file */usr/adm/implog.*

Entries in the file are variable length. Each log entry has a fixed length header of the form:

```
struct sockstamp {
        short   sin_family;
        u_short         sin_port;
        struct  in_addr sin_addr;
        time_t  sin_time;
        int     sin_len;
};
```

followed, possibly, by the message received from the IMP. Each time the logging process is started up it places a time stamp entry in the file (a header with *sin_len* field set to 0).

The logging process will catch only those message from the IMP which are not processed by a protocol module, e.g. IP. This implies the log should contain only status information such as "IMP going down" messages and, perhaps, stray NCP messages.

SEE ALSO

imp(4P), implog(8C)

BUGS

The messages should probably be sent to the system error logging process instead of maintaining yet another log file.

NAME

init – process control initialization

SYNOPSIS

/etc/init

DESCRIPTION

Init is invoked inside UNIX as the last step in the boot procedure. It normally then runs the automatic reboot sequence as described in *reboot*(8), and if this succeeds, begins multi-user operation. If the reboot fails, it commences single user operation by giving the super-user a shell on the console. It is possible to pass parameters from the boot program to *init* so that single user operation is commenced immediately. When such single user operation is terminated by killing the single-user shell (i.e. by hitting ^D), *init* runs */etc/rc* without the reboot parameter. This command file performs housekeeping operations such as removing temporary files, mounting file systems, and starting daemons.

In multi-user operation, *init's* role is to create a process for each terminal port on which a user may log in. To begin such operations, it reads the file */etc/ttys* and forks several times to create a process for each terminal specified in the file. Each of these processes opens the appropriate terminal for reading and writing. These channels thus receive file descriptors 0, 1 and 2, the standard input and output and the diagnostic output. Opening the terminal will usually involve a delay, since the *open* is not completed until someone is dialed up and carrier established on the channel. If a terminal exists but an error occurs when trying to open the terminal *init* complains by writing a message to the system console; the message is repeated every 10 minutes for each such terminal until the terminal is shut off in /etc/ttys and init notified (by a hangup, as described below), or the terminal becomes accessible (init checks again every minute). After an open succeeds, */etc/getty* is called with argument as specified by the second character of the *ttys* file line. *Getty* reads the user's name and invokes *login* to log in the user and execute the Shell.

Ultimately the Shell will terminate because of an end-of-file either typed explicitly or generated as a result of hanging up. The main path of *init*, which has been waiting for such an event, wakes up and removes the appropriate entry from the file *utmp*, which records current users, and makes an entry in */usr/adm/wtmp*, which maintains a history of logins and logouts. The *wtmp* entry is made only if a user logged in successfully on the line. Then the appropriate terminal is reopened and *getty* is reinvoked.

Init catches the *hangup* signal (signal SIGHUP) and interprets it to mean that the file */etc/ttys* should be read again. The Shell process on each line which used to be active in *ttys* but is no longer there is terminated; a new process is created for each added line; lines unchanged in the file are undisturbed. Thus it is possible to drop or add phone lines without rebooting the system by changing the *ttys* file and sending a *hangup* signal to the *init* process: use 'kill –HUP 1.'

Init will terminate multi-user operations and resume single-user mode if sent a terminate (TERM) signal, i.e. "kill –TERM 1". If there are processes outstanding which are deadlocked (due to hardware or software failure), *init* will not wait for them all to die (which might take forever), but will time out after 30 seconds and print a warning message.

Init will cease creating new *getty*'s and allow the system to slowly die away, if it is sent a terminal stop (TSTP) signal, i.e. "kill –TSTP 1". A later hangup will resume full multi-user operations, or a terminate will initiate a single user shell. This hook is used by *reboot*(8) and *halt*(8).

Init's role is so critical that if it dies, the system will reboot itself automatically. If, at bootstrap time, the *init* process cannot be located, the system will loop in user mode at location 0x13.

DIAGNOSTICS

init: ***tty*****: cannot open**. A terminal which is turned on in the *rc* file cannot be opened, likely because the requisite lines are either not configured into the system or the associated device was not attached during boot-time system configuration.

WARNING: Something is hung (wont die); ps axl advised. A process is hung and could not be killed when the system was shutting down. This is usually caused by a process which is stuck in a device driver due to a persistent device error condition.

FILES

/dev/console, /dev/tty*, /etc/utmp, /usr/adm/wtmp, /etc/ttys, /etc/rc

SEE ALSO

login(1), kill(1), sh(1), ttys(5), crash(8V), getty(8), rc(8), reboot(8), halt(8), shutdown(8)

NAME

kgmon – generate a dump of the operating system's profile buffers

SYNOPSIS

/etc/kgmon [**–b**] [**–h**] [**–r**] [**–p**] [system] [memory]

DESCRIPTION

Kgmon is a tool used when profiling the operating system. When no arguments are supplied, *kgmon* indicates the state of operating system profiling as running, off, or not configured. (see *config*(8)) If the **–p** flag is specified, *kgmon* extracts profile data from the operating system and produces a *gmon.out* file suitable for later analysis by *gprof*(1).

The following options may be specified:

–b	Resume the collection of profile data.
–h	Stop the collection of profile data.
–p	Dump the contents of the profile buffers into a *gmon.out* file.
–r	Reset all the profile buffers. If the **–p** flag is also specified, the *gmon.out* file is generated before the buffers are reset.

If neither **–b** nor **–h** is specified, the state of profiling collection remains unchanged. For example, if the **–p** flag is specified and profile data is being collected, profiling will be momentarily suspended, the operating system profile buffers will be dumped, and profiling will be immediately resumed.

FILES

/vmunix – the default system
/dev/kmem – the default memory

SEE ALSO

gprof(1), config(8)

DIAGNOSTICS

Users with only read permission on /dev/kmem cannot change the state of profiling collection. They can get a *gmon.out* file with the warning that the data may be inconsistent if profiling is in progress.

NAME

lpc – line printer control program

SYNOPSIS

/etc/lpc [command [argument ...]]

DESCRIPTION

Lpc is used by the system administrator to control the operation of the line printer system. For each line printer configured in /etc/printcap, *lpc* may be used to:

- disable or enable a printer,
- disable or enable a printer's spooling queue,
- rearrange the order of jobs in a spooling queue,
- find the status of printers, and their associated spooling queues and printer dameons.

Without any arguments, *lpc* will prompt for commands from the standard input. If arguments are supplied, *lpc* interprets the first argument as a command and the remaining arguments as parameters to the command. The standard input may be redirected causing *lpc* to read commands from file. Commands may be abbreviated; the following is the list of recognized commands.

? [command ...]

help [command ...]
    Print a short description of each command specified in the argument list, or, if no arguments are given, a list of the recognized commands.

abort { all | printer ... }
    Terminate an active spooling daemon on the local host immediately and then disable printing (preventing new daemons from being started by *lpr*) for the specified printers.

clean { all | printer ... }
    Remove all files beginning with "cf", "tf", or "df" from the specified printer queue(s) on the local machine.

enable { all | printer ... }
    Enable spooling on the local queue for the listed printers. This will allow *lpr* to put new jobs in the spool queue.

exit

quit
    Exit from lpc.

disable { all | printer ... }
    Turn the specified printer queues off. This prevents new printer jobs from being entered into the queue by *lpr*.

restart { all | printer ... }
    Attempt to start a new printer daemon. This is useful when some abnormal condition causes the daemon to die unexpectedly leaving jobs in the queue. *Lpq* will report that there is no daemon present when this condition occurs.

start { all | printer ... }
    Enable printing and start a spooling daemon for the listed printers.

status [all] [printer ...]
    Display the status of daemons and queues on the local machine.

stop { all | printer ... }
Stop a spooling daemon after the current job completes and disable printing.

topq printer [jobnum ...] [user ...]
Place the jobs in the order listed at the top of the printer queue.

FILES

/etc/printcap	printer description file
/usr/spool/*	spool directories
/usr/spool/*/lock	lock file for queue control

SEE ALSO

lpd(8), lpr(1), lpq(1), lprm(1), printcap(5), cleanlpd(8)

DIAGNOSTICS

?Ambiguous command	abbreviation matches more than one command
?Invalid command	no match was found
?Privileged command	command can be executed by root only

NAME

lpd – line printer daemon

SYNOPSIS

/usr/lib/lpd [-l] [-L logfile] [port #]

DESCRIPTION

Lpd is the line printer daemon (spool area handler) and is normally invoked at boot time from the *rc*(8) file. It makes a single pass through the *printcap*(5) file to find out about the existing printers and prints any files left after a crash. It then uses the system calls *listen*(2) and *accept*(2) to receive requests to print files in the queue, transfer files to the spooling area, display the queue, or remove jobs from the queue. In each case, it forks a child to handle the request so the parent can continue to listen for more requests. The Internet port number used to rendezvous with other processes is normally obtained with *getservbyname*(3) but can be changed with the *port#* argument. The **–L** option changes the file used for writing error conditions from the system console to *logfile.* The –l flag causes *lpd* to log valid requests received from the network. This can be useful for debugging purposes.

Access control is provided by two means. First, All requests must come from one of the machines listed in the file */etc/hosts.equiv*. Second, if the "rs" capability is specified in the *printcap* entry for the printer being accessed, *lpr* requests will only be honored for those users with accounts on the machine with the printer.

The file *lock* in each spool directory is used to prevent multiple daemons from becoming active simultaneously, and to store information about the daemon process for *lpr*(1), *lpq*(1), and *lprm*(1). After the daemon has successfully set the lock, it scans the directory for files beginning with *cf*. Lines in each *cf* file specify files to be printed or non-printing actions to be performed. Each such line begins with a key character to specify what to do with the remainder of the line.

J	Job Name. String to be used for the job name on the burst page.
C	Classification. String to be used for the classification line on the burst page.
L	Literal. The line contains identification info from the password file and causes the banner page to be printed.
T	Title. String to be used as the title for *pr*(1).
H	Host Name. Name of the machine where *lpr* was invoked.
P	Person. Login name of the person who invoked *lpr*. This is used to verify ownership by *lprm*.
M	Send mail to the specified user when the current print job completes.
f	Formatted File. Name of a file to print which is already formatted.
l	Like "f" but passes control characters and does not make page breaks.
p	Name of a file to print using *pr*(1) as a filter.
t	Troff File. The file contains *troff*(1) output (cat phototypesetter commands).
d	DVI File. The file contains *Tex*(l) output (DVI format from Stanford).
g	Graph File. The file contains data produced by *plot*(3X).
c	Cifplot File. The file contains data produced by *cifplot*.
v	The file contains a raster image.
r	The file contains text data with FORTRAN carriage control characters.

1	Troff Font R. Name of the font file to use instead of the default.
2	Troff Font I. Name of the font file to use instead of the default.
3	Troff Font B. Name of the font file to use instead of the default.
4	Troff Font S. Name of the font file to use instead of the default.
W	Width. Changes the page width (in characters) used by *pr*(1) and the text filters.
I	Indent. The number of characters to indent the output by (in ascii).
U	Unlink. Name of file to remove upon completion of printing.
N	File name. The name of the file which is being printed, or a blank for the standard input (when *lpr* is invoked in a pipeline).

If a file can not be opened, a message will be placed in the log file (normally the console). *Lpd* will try up to 20 times to reopen a file it expects to be there, after which it will skip the file to be printed.

Lpd uses *flock*(2) to provide exclusive access to the lock file and to prevent multiple deamons from becoming active simultaneously. If the daemon should be killed or die unexpectedly, the lock file need not be removed. The lock file is kept in a readable ASCII form and contains two lines. The first is the process id of the daemon and the second is the control file name of the current job being printed. The second line is updated to reflect the current status of *lpd* for the programs *lpq*(1) and *lprm*(1).

FILES

/etc/printcap	printer description file
/usr/spool/*	spool directories
/dev/lp*	line printer devices
/dev/printer	socket for local requests
/etc/hosts.equiv	lists machine names allowed printer access

SEE ALSO

lpc(8), pac(1), lpr(1), lpq(1), lprm(1), printcap(5), cleanlpd(8)
4.2BSD Line Printer Spooler Manual

NAME

makedev – make system special files

SYNOPSIS

/dev/MAKEDEV *device...*

DESCRIPTION

MAKEDEV is a shell script normally used to install special files. It resides in the */dev* directory, as this is the normal location of special files. Arguments to *MAKEDEV* are usually of the form ***device-name***? where ***device-name*** is one of the supported devices listed in section 4 of the manual and "?" is a logical unit number (0-9). A few special arguments create assorted collections of devices and are listed below.

std Create the *standard* devices for the system; e.g. /dev/console, /dev/tty.

local Create those devices specific to the local site. This request causes the shell file */dev/MAKEDEV.local* to be executed. Site specific commands, such as those used to setup dialup lines as "ttyd?" should be included in this file.

Since all devices are created using *mknod*(8), this shell script is useful only to the super-user.

DIAGNOSTICS

Either self-explanatory, or generated by one of the programs called from the script. Use "sh -x MAKEDEV" in case of trouble.

SEE ALSO

intro(4), config(8), mknod(8)

BUGS

When more than one piece of hardware of the same "kind" is present on a machine (for instance, a dh and a dmf), naming conflicts arise.

NAME

makekey – generate encryption key

SYNOPSIS

/usr/lib/makekey

DESCRIPTION

Makekey improves the usefulness of encryption schemes depending on a key by increasing the amount of time required to search the key space. It reads 10 bytes from its standard input, and writes 13 bytes on its standard output. The output depends on the input in a way intended to be difficult to compute (that is, to require a substantial fraction of a second).

The first eight input bytes (the ***input key***) can be arbitrary ASCII characters. The last two (the *salt*) are best chosen from the set of digits, upper- and lower-case letters, and '.' and '/'. The salt characters are repeated as the first two characters of the output. The remaining 11 output characters are chosen from the same set as the salt and constitute the *output key.*

The transformation performed is essentially the following: the salt is used to select one of 4096 cryptographic machines all based on the National Bureau of Standards DES algorithm, but modified in 4096 different ways. Using the input key as key, a constant string is fed into the machine and recirculated a number of times. The 64 bits that come out are distributed into the 66 useful key bits in the result.

Makekey is intended for programs that perform encryption (for instance, *ed* and *crypt*(1)). Usually makekey's input and output will be pipes.

SEE ALSO

crypt(1), ed(1)

NAME

mkfs – program to make UNIX file systems

DESCRIPTION

Mkfs is a program executable only from standalone mode. It is used to install the skeleton of a UNIX file system on a hard disk or a floppy disk. This must be done after the disk has been formatted so that the disk may be used.

For a detailed description of mkfs, reference the *MPS020-2 Operating System Installation Guide*

FILES

/stand/mkfs

SEE ALSO

standalone(8),

NAME

mklost+found – make a lost+found directory for fsck

SYNOPSIS

/etc/mklost+found

DESCRIPTION

A directory *lost+found* is created in the current directory and a number of empty files are created therein and then removed so that there will be empty slots for *fsck*(8). This command should not normally be needed since *mkfs*(8) automatically creates the *lost+found* directory when a new file system is created.

SEE ALSO

fsck(8), mkfs(8)

NAME

mknod – build special file

SYNOPSIS

/etc/mknod name [**c**] [**b**] major minor

DESCRIPTION

Mknod makes a special file. The first argument is the *name* of the entry. The second 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).

The assignment of major device numbers is specific to each system. They have to be dug out of the system source file *conf.c*.

SEE ALSO

mknod(2)

NAME

mkproto – construct a prototype file system

SYNOPSIS

/etc/mkproto special proto

DESCRIPTION

Mkproto is used to bootstrap a new file system. First a new file system is created using *newfs*(8). *Mkproto* is then used to copy files from the old file system into the new file system according to the directions found in the prototype file *proto*. The prototype file contains tokens separated by spaces or new lines. The first tokens comprise the specification for the root directory. File specifications consist of tokens giving the mode, the user-id, the group id, and the initial contents of the file. The syntax of the contents field depends on the mode.

The mode token for a file is a 6 character string. The first character specifies the type of the file. (The characters **–bcd** specify regular, block special, character special and directory files respectively.) The second character of the type is either **u** or – to specify set-user-id mode or not. The third is **g** or – for the set-group-id mode. The rest of the mode is a three 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 is a pathname whence the contents and size are copied.

If the file is a block or character special file, two decimal number tokens follow which give the major and minor device numbers.

If the file is a directory, *mkproto* makes the entries **.** and **..** and then reads a list of names and (recursively) file specifications for the entries in the directory. The scan is terminated with the token $.

A sample prototype specification follows:

```
d—777 3 1
usr     d—777 3 1
        sh      ——755 3 1 /bin/sh
        ken     d—755 6 1
                $
        b0      b—644 3 1 0 0
        c0      c—644 3 1 0 0
        $
$
```

SEE ALSO

fs(5), dir(5), fsck(8), newfs(8)

BUGS

There should be some way to specify links.

There should be some way to specify bad blocks.

Mkproto can only be run on virgin file systems. It should be possible to copy files into existent file systems.

NAME

mount, umount – mount and dismount file system

SYNOPSIS

/etc/mount [special name [**–r**]]

/etc/mount –a

/etc/umount special

/etc/umount –a

DESCRIPTION

Mount announces to the system that a removable file system is present on the device *special.* The file *name* must exist already; it must be a directory (unless the root of the mounted file system is not a directory). It becomes the name of the newly mounted root. The optional argument **–r** indicates that the file system is to be mounted read-only.

Umount announces to the system that the removable file system previously mounted on device *special* is to be removed.

If the **–a** option is present for either *mount* or *umount,* all of the file systems described in */etc/fstab* are attempted to be mounted or unmounted. In this case, *special* and *name* are taken from */etc/fstab.* The *special* file name from */etc/fstab* is the block special name.

These commands maintain a table of mounted devices in */etc/mtab.* If invoked without an argument, *mount* prints the table.

Physically write-protected and magnetic tape file systems must be mounted read-only or errors will occur when access times are updated, whether or not any explicit write is attempted.

FILES

/etc/mtab	mount table
/etc/fstab	file system table

SEE ALSO

mount(2), mtab(5), fstab(5)

BUGS

Mounting file systems full of garbage will crash the system.
Mounting a root directory on a non-directory makes some apparently good pathnames invalid.

NAME

ncheck – generate names from i-numbers

SYNOPSIS

/etc/ncheck [**–i** numbers] [**–a**] [**–s**] [filesystem]

DESCRIPTION

N.B.: For most normal file system maintenance, the function of *ncheck* is subsumed by *fsck*(8).

Ncheck with no argument generates a pathname vs. i-number list of all files on a set of default file systems. Names of directory files are followed by '/.'. The **–i** option reduces the report to only those files whose i-numbers follow. The **–a** option allows printing of the names '.' and '..', which are ordinarily suppressed. The **–s** option reduces the report to special files and files with set-user-ID mode; it is intended to discover concealed violations of security policy.

A file system may be specified.

The report is in no useful order, and probably should be sorted.

SEE ALSO

sort(1), dcheck(8), fsck(8), icheck(8)

DIAGNOSTICS

When the filesystem structure is improper, '??' denotes the 'parent' of a parentless file and a pathname beginning with '...' denotes a loop.

NAME

newfs – construct a new file system

SYNOPSIS

/etc/newfs [**–v**] [**–n**] [**mkfs-options**] **special disk-type**

DESCRIPTION

Newfs is a "friendly" front-end to the *mkfs*(8) program. *Newfs* will look up the type of disk a file system is being created on in the disk description file */etc/disktab*, calculate the appropriate parameters to use in calling *mkfs*, then build the file system by forking *mkfs* and, if the file system is a root partition, install the necessary bootstrap programs in the initial 8 sectors of the device. The **–n** option prevents the bootstrap programs from being installed.

If the **–v** option is supplied, *newfs* will print out its actions, including the parameters passed to *mkfs*.

Options which may be used to override default parameters passed to *mkfs* are:

–s size The size of the file system in sectors.

–b block-size
The block size of the file system in bytes.

–f frag-size
The fragment size of the file system in bytes.

–t #tracks/cylinder

–c #cylinders/group
The number of cylinders per cylinder group in a file system. The default value used is 16.

–m free space %
The percentage of space reserved from normal users; the minimum free space threshhold. The default value used is 10%.

–r revolutions/minute
The speed of the disk in revolutions per minute (normally 3600).

–S sector-size
The size of a sector in bytes (almost never anything but 512).

–i number of bytes per inode
This specifies the density of inodes in the file system. The default is to create an inode for each 2048 bytes of data space. If fewer inodes are desired, a larger number should be used; to create more inodes a smaller number should be given.

FILES

/etc/disktab	for disk geometry and file system partition information
/etc/mkfs	to actually build the file system
/usr/mdec	for boot strapping programs

SEE ALSO

disktab(5), fs(5), diskpart(8), fsck(8), format(8), mkfs(8), tunefs(8)

McKusick, Joy, Leffler; "A Fast File System for Unix", Computer Systems Research Group, Dept of EECS, Berkeley, CA 94720; TR #7, September 1982.

BUGS

Should figure out the type of the disk without the user's help.

NAME

pac – printer/plotter accounting information

SYNOPSIS

/etc/pac [**–Pprinter**] [**–pprice**] [**–s**] [**–r**] [**–c**] [name ...]

DESCRIPTION

Pac reads the printer/plotter accounting files, accumulating the number of pages (the usual case) or feet (for raster devices) of paper consumed by each user, and printing out how much each user consumed in pages or feet and dollars. If any *names* are specified, then statistics are only printed for those users; usually, statistics are printed for every user who has used any paper.

The **–P** flag causes accounting to be done for the named printer. Normally, accounting is done for the default printer (site dependent) or the value of the environment variable **PRINTER** is used.

The **–p** flag causes the value *price* to be used for the cost in dollars instead of the default value of 0.02.

The **–c** flag causes the output to be sorted by cost; usually the output is sorted alphabetically by name.

The **–r** flag reverses the sorting order.

The **–s** flag causes the accounting information to be summarized on the summary accounting file; this summarization is necessary since on a busy system, the accounting file can grow by several lines per day.

FILES

/usr/adm/?acct	raw accounting files
/usr/adm/?_sum	summary accounting files

BUGS

The relationship between the computed price and reality is as yet unknown.

NAME

park – program to park the hard disk heads

DESCRIPTION

Park is a program executable only from standalone mode. It is used to park the heads on the hard disk, so that they will not damage the disk if the machine is jarred while moving it.

It is always a good idea to park the disk heads, no matter how short a distance the machine may be moved.

For a more detailed description of park, see the *MPS020-2 Operating System Installation Guide*

FILES

/stand/park

SEE ALSO

standalone(8),

NAME

ping – send ICMP ECHO_REQUEST packets to network hosts

SYNOPSIS

/etc/ping [**–r**] [**–v**] *host* [*packetsize*] [*count*]

DESCRIPTION

The DARPA Internet is a large and complex aggregation of network hardware, connected together by gateways. Tracking a single-point hardware or software failure can often be difficult. *Ping* utilizes the ICMP protocol's mandatory ECHO_REQUEST datagram to elicit an ICMP ECHO_RESPONSE from a host or gateway. ECHO_REQUEST datagrams ("pings") have an IP and ICMP header, followed by a **struct timeval**, and then an arbitrary number of "pad" bytes used to fill out the packet. Default datagram length is 64 bytes, but this may be changed using the command-line option. Other options are:

–r Bypass the normal routing tables and send directly to a host on an attached network. If the host is not on a directly-attached network, an error is returned. This option can be used to ping a local host through an interface that has no route through it (e.g., after the interface was dropped by *routed*(8C)).

–v Verbose output. ICMP packets other than ECHO RESPONSE that are received are listed.

When using *ping* for fault isolation, it should first be run on the local host, to verify that the local network interface is up and running. Then, hosts and gateways further and further away should be "pinged". *Ping* sends one datagram per second, and prints one line of output for every ECHO_RESPONSE returned. No output is produced if there is no response. If an optional *count* is given, only that number of requests is sent. Round-trip times and packet loss statistics are computed. When all responses have been received or the program times out (with a *count* specified), or if the program is terminated with a SIGINT, a brief summary is displayed.

This program is intended for use in network testing, measurement and management. It should be used primarily for manual fault isolation. Because of the load it could impose on the network, it is unwise to use *ping* during normal operations or from automated scripts.

AUTHOR

Mike Muuss, U.S. Army Ballistic Research Laboratory

SEE ALSO

netstat(1), ifconfig(8C)

BUGS

More statistics could always be gathered.

This program must either be run by root or be SUID root to have access to the ICMP socket.

NAME

pstat – print system facts

SYNOPSIS

/etc/pstat –aixptufT [**suboptions**] [**system**] [**corefile**]

DESCRIPTION

Pstat interprets the contents of certain system tables. If *corefile* is given, the tables are sought there, otherwise in */dev/kmem.* The required namelist is taken from */vmunix* unless *system* is specified. Options are

–a Under **–p**, describe all process slots rather than just active ones.

–i Print the inode table with the these headings:

LOC The core location of this table entry.

FLAGS Miscellaneous state variables encoded thus:

L	locked
U	update time (*fs*(5)) must be corrected
A	access time must be corrected
M	file system is mounted here
W	wanted by another process (L flag is on)
T	contains a text file
C	changed time must be corrected
S	shared lock applied
E	exclusive lock applied
Z	someone waiting for an exclusive lock

CNT Number of open file table entries for this inode.

DEV Major and minor device number of file system in which this inode resides.

RDC Reference count of shared locks on the inode.

WRC Reference count of exclusive locks on the inode (this may be > 1 if, for example, a file descriptor is inherited across a fork).

INO I-number within the device.

MODE Mode bits, see *chmod*(2).

NLK Number of links to this inode.

UID User ID of owner.

SIZ/DEV
Number of bytes in an ordinary file, or major and minor device of special file.

–x Print the text table with these headings:

LOC The core location of this table entry.

FLAGS Miscellaneous state variables encoded thus:

T	*ptrace*(2) in effect
W	text not yet written on swap device
L	loading in progress
K	locked
w	wanted (L flag is on)
P	resulted from demand-page-from-inode exec format (see *execve*(2))

DADDR Disk address in swap, measured in multiples of 512 bytes.

CADDR Head of a linked list of loaded processes using this text segment.

SIZE Size of text segment, measured in multiples of 512 bytes.

IPTR Core location of corresponding inode.

CNT	Number of processes using this text segment.
CCNT	Number of processes in core using this text segment.

−p Print process table for active processes with these headings:

LOC	The core location of this table entry.
S	Run state encoded thus:

0	no process
1	waiting for some event
3	runnable
4	being created
5	being terminated
6	stopped under trace

F Miscellaneous state variables, or-ed together (hexadecimal):

000001	loaded
000002	the scheduler process
000004	locked for swap out
000008	swapped out
000010	traced
000020	used in tracing
000080	in page-wait
000100	prevented from swapping during *fork*(2)
000200	gathering pages for raw i/o
000400	exiting
001000	process resulted from a *vfork*(2) which is not yet complete
002000	another flag for *vfork*(2)
004000	process has no virtual memory, as it is a parent in the context of *vfork*(2)
008000	process is demand paging data pages from its text inode.
010000	process has advised of anomalous behavior with *vadvise*(2).
020000	process has advised of sequential behavior with *vadvise*(2).
040000	process is in a sleep which will timeout.
080000	a parent of this process has exited and this process is now considered detached.
100000	process used 4.1BSD compatibility mode signal primitives, no system calls will restart.
200000	process is owed a profiling tick.

POIP	number of pages currently being pushed out from this process.
PRI	Scheduling priority, see *setpriority*(2).
SIGNAL	Signals received (signals 1-32 coded in bits 0-31),
UID	Real user ID.
SLP	Amount of time process has been blocked.
TIM	Time resident in seconds; times over 127 coded as 127.
CPU	Weighted integral of CPU time, for scheduler.
NI	Nice level, see *setpriority*(2).
PGRP	Process number of root of process group (the opener of the controlling terminal).
PID	The process ID number.
PPID	The process ID of parent process.
ADDR	If in core, the page frame number of the first page of the 'u-area' of the process. If swapped out, the position in the swap area measured in multiples of 512 bytes.
RSS	Resident set size – the number of physical page frames allocated to this process.
SRSS	RSS at last swap (0 if never swapped).
SIZE	Virtual size of process image (data+stack) in multiples of 512 bytes.
WCHAN	Wait channel number of a waiting process.

LINK	Link pointer in list of runnable processes.
TEXTP	If text is pure, pointer to location of text table entry.
CLKT	Countdown for real interval timer, *setitimer*(2) measured in clock ticks (10 milliseconds).

–t Print table for terminals with these headings:

RAW	Number of characters in raw input queue.
CAN	Number of characters in canonicalized input queue.
OUT	Number of characters in putput queue.
MODE	See *tty*(4).
ADDR	Physical device address.
DEL	Number of delimiters (newlines) in canonicalized input queue.
COL	Calculated column position of terminal.
STATE	Miscellaneous state variables encoded thus:

W	waiting for open to complete
O	open
S	has special (output) start routine
C	carrier is on
B	busy doing output
A	process is awaiting output
X	open for exclusive use
H	hangup on close

PGRP	Process group for which this is controlling terminal.
DISC	Line discipline; blank is old tty OTTYDISC or "new tty" for NTTYDISC or "net" for NETLDISC (see *bk*(4)).

–u print information about a user process; the next argument is its address as given by *ps*(1). The process must be in main memory, or the file used can be a core image and the address 0.

–f Print the open file table with these headings:

LOC	The core location of this table entry.
TYPE	The type of object the file table entry points to.
FLG	Miscellaneous state variables encoded thus:

R	open for reading
W	open for writing
A	open for appending

CNT	Number of processes that know this open file.
INO	The location of the inode table entry for this file.
OFFS/SOCK	The file offset (see *lseek*(2)), or the core address of the associated socket structure.

–s print information about swap space usage: the number of (1k byte) pages used and free is given as well as the number of used pages which belong to text images.

–T prints the number of used and free slots in the several system tables and is useful for checking to see how full system tables have become if the system is under heavy load.

FILES

/vmunix	namelist
/dev/kmem	default source of tables

SEE ALSO

ps(1), stat(2), fs(5)
K. Thompson, *UNIX Implementation*

BUGS

It would be very useful if the system recorded "maximum occupancy" on the tables reported by **–T**; even more useful if these tables were dynamically allocated.

NAME

quot – summarize file system ownership

SYNOPSIS

/etc/quot [option] ... [filesystem]

DESCRIPTION

Quot prints the number of blocks in the named *filesystem* currently owned by each user. If no *filesystem* is named, a default name is assumed. The following options are available:

–n Cause the pipeline **ncheck filesystem | sort +0n | quot –n filesystem** to produce a list of all files and their owners.

–c Print three columns giving file size in blocks, number of files of that size, and cumulative total of blocks in that size or smaller file.

–f Print count of number of files as well as space owned by each user.

FILES

Default file system varies with system.
/etc/passwd to get user names

SEE ALSO

ls(1), du(1)

NAME

quotacheck – file system quota consistency checker

SYNOPSIS

/etc/quotacheck [**–v**] filesystem...
/etc/quotacheck [**–v**] **–a**

DESCRIPTION

Quotacheck examines each file system, builds a table of current disc usage, and compares this table against that stored in the disc quota file for the file system. If any inconsistencies are detected, both the quota file and the current system copy of the incorrect quotas are updated (the latter only occurs if an active file system is checked).

If the **–a** flag is supplied in place of any file system names, *quotacheck* will check all the file systems indicated in */etc/fstab* to be read-write with disc quotas.

Normally *quotacheck* reports only those quotas modified. If the **–v** option is supplied, *quotacheck* will indicate the calculated disc quotas for each user on a particular file system.

Quotacheck expects each file system to be checked to have a quota file named *quotas* in the root directory. If none is present, *quotacheck* will ignore the file system.

Quotacheck is normally run at boot time from the */etc/rc.local* file, see *rc*(8), before enabling disc quotas with *quotaon*(8).

Quotacheck accesses the raw device in calculating the actual disc usage for each user. Thus, the file systems checked should be quiescent while *quotacheck* is running.

FILES

/etc/fstab	default file systems

SEE ALSO

quota(2), setquota(2), quotaon(8)

NAME

quotaon, quotaoff – turn file system quotas on and off

SYNOPSIS

/etc/quotaon [**–v**] *filsys...*

/etc/quotaon [**–v**] **–a**

/etc/quotaoff [**–v**] *filsys...*

/etc/quotaoff [**–v**] **–a**

DESCRIPTION

Quotaon announces to the system that disc quotas should be enabled on one or more file systems. The file systems specified must have entries in /etc/fstab and be mounted at the time. The file system quota files must be present in the root directory of the specified file system and be named *quotas*. The optional argument **–v** causes *quotaon* to print a message for each file system where quotas are turned on. If, instead of a list of file systems, a **–a** argument is give to *quotaon*, all file systems in /etc/fstab marked read-write with quotas will have their quotas turned on. This is normally used at boot time to enable quotas.

Quotaoff announces to the system that file systems specified should have any disc quotas turned off. As above, the **–v** forces a verbose message for each file system affected; and the **–a** option forces all file systems in /etc/fstab to have their quotas disabled.

These commands update the status field of devices located in */etc/mtab* to indicate when quotas are on or off for each file system.

FILES

/etc/mtab	mount table
/etc/fstab	file system table

SEE ALSO

setquota(2), mtab(5), fstab(5)

NAME

rc – command script for auto-reboot and daemons

SYNOPSIS

/etc/rc
/etc/rc.local

DESCRIPTION

Rc is the command script which controls the automatic reboot and *rc.local* is the script holding commands which are pertinent only to a specific site.

When an automatic reboot is in progress, *rc* is invoked with the argument *autoboot* and runs a *fsck* with option **–p** to "preen" all the disks of minor inconsistencies resulting from the last system shutdown and to check for serious inconsistencies caused by hardware or software failure. If this auto-check and repair succeeds, then the second part of *rc* is run.

The second part of *rc,* which is run after a auto-reboot succeeds and also if *rc* is invoked when a single user shell terminates (see *init*(8)), starts all the daemons on the system, preserves editor files and clears the scratch directory **/tmp.** *Rc.local* is executed immediately before any other commands after a successful *fsck*. Normally, the first commands placed in the *rc.local* file define the machine's name, using *hostname*(1), and save any possible core image that might have been generated as a result of a system crash, *savecore*(8). The latter command is included in the *rc.local* file because the directory in which core dumps are saved is usually site specific.

SEE ALSO

init(8), reboot(8), savecore(8)

NAME
rdump – file system dump across the network

SYNOPSIS
/etc/rdump [key [*argument* ...] filesystem]

DESCRIPTION
Rdump copies to magnetic tape all files changed after a certain date in the *filesystem*. The command is identical in operation to *dump*(8) except the *f* key should be specified and the file supplied should be of the form *machine:device*.

Rdump creates a remote server, */etc/rmt*, on the client machine to access the tape device.

SEE ALSO
dump(8), rmt(8C)

DIAGNOSTICS
Same as *dump*(8) with a few extra related to the network.

NAME

reboot – UNIX bootstrapping procedures

SYNOPSIS

/etc/reboot [**–n**] [**–q**]

DESCRIPTION

UNIX is started by placing it in memory at location zero and transferring to zero. Since the system is not reenterable, it is necessary to read it in from disk or tape each time it is to be bootstrapped.

Rebooting a running system. When a UNIX is running and a reboot is desired, *shutdown*(8) is normally used. If there are no users then **/etc/reboot** can be used. Reboot causes the disks to be synced, and then a multi-user reboot (as described below) is initiated. This causes a system to be booted and an automatic disk check to be performed. If all this succeeds without incident, the system is then brought up for many users.

Options to reboot are:

–n option avoids the sync. It can be used if a disk or the processor is on fire.

–q reboots quickly and ungracefully, without shutting down running processes first.

Power fail and crash recovery. Normally, the system will reboot itself at power-up. When the system crashes, it will hang up with an error message, generally, being printed on the console. The system must then be rebooted by turning the keyswitch to the reset position (this is a spring loaded position).

When the system is being rebooted, the user will be given the option to forgo the automatic reboot process, and interact with the loader(8), to, perhaps, load standalone programs, or other versions of the UNIX kernel.

In an emergency, the bootstrap methods described in the paper "Installing and Operating 4.2bsd" can be used to boot from a distribution tape.

FILES

/vmunix	virtual memory system code
/m1unix	main system code
/dcunix	disk cache code

SEE ALSO

crash(8V), fsck(8), init(8), rc(8), shutdown(8), halt(8), newfs(8)

NAME

renice – alter priority of running processes

SYNOPSIS

/etc/renice priority [[**–p**] pid ...] [[**–g**] pgrp ...] [[**–u**] user ...]

DESCRIPTION

Renice alters the scheduling priority of one or more running processes. The *who* parameters are interpreted as process ID's, process group ID's, or user names. *Renice*'ing a process group causes all processes in the process group to have their scheduling priority altered. *Renice*'ing a user causes all processes owned by the user to have their scheduling priority altered. By default, the processes to be affected are specified by their process ID's. To force *who* parameters to be interpreted as process group ID's, a **–g** may be specified. To force the *who* parameters to be interpreted as user names, a **–u** may be given. Supplying **–p** will reset *who* interpretation to be (the default) process ID's. For example,

```
/etc/renice +1 987 -u daemon root -p 32
```

would change the priority of process ID's 987 and 32, and all processes owned by users daemon and root.

Users other than the super-user may only alter the priority of processes they own, and can only monotonically increase their "nice value" within the range 0 to PRIO_MIN (20). (This prevents overriding administrative fiats.) The super-user may alter the priority of any process and set the priority to any value in the range PRIO_MAX (–20) to PRIO_MIN. Useful priorities are: 19 (the affected processes will run only when nothing else in the system wants to), 0 (the "base" scheduling priority), anything negative (to make things go very fast).

FILES

/etc/passwd to map user names to user ID's

SEE ALSO

getpriority(2), setpriority(2)

BUGS

If you make the priority very negative, then the process cannot be interrupted. To regain control you make the priority greater than zero. Non super-users can not increase scheduling priorities of their own processes, even if they were the ones that decreased the priorities in the first place.

NAME

repquota – summarize quotas for a file system

SYNOPSIS

repquota *filesys*...

DESCRIPTION

Repquota prints a summary of the disc usage and quotas for the specified file systems. For each user the current number files and amount of space (in kilobytes) is printed, along with any quotas created with *edquota*(8).

Only the super-user may view quotas which are not their own.

FILES

quotas	at the root of each file system with quotas
/etc/fstab	for file system names and locations

SEE ALSO

quota(1), quota(2), quotacheck(8), quotaon(8), edquota(8)

DIAGNOSTICS

Various messages about inaccessible files; self-explanatory.

NAME

restore – incremental file system restore

SYNOPSIS

/etc/restore key [name ...]

DESCRIPTION

Restore reads tapes dumped with the *dump*(8) command. Its actions are controlled by the *key* argument. The *key* is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to the command are file or directory names specifying the files that are to be restored. Unless the **h** key is specified (see below), the appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

The function portion of the key is specified by one of the following letters:

r The tape is read and loaded into the current directory. This should not be done lightly; the **r** key should only be used to restore a complete dump tape onto a clear file system or to restore an incremental dump tape after a full level zero restore. Thus

```
/etc/newfs /dev/rrp0g eagle
/etc/mount /dev/rp0g /mnt
cd /mnt
restore r
```

is a typical sequence to restore a complete dump. Another *restore* can be done to get an incremental dump in on top of this. Note that *restore* leaves a file *restoresymtab* in the root directory to pass information between incremental restore passes. This file should be removed when the last incremental tape has been restored.
A *dump*(8) followed by a *newfs*(8) and a *restore* is used to change the size of a file system.

R *Restore* requests a particular tape of a multi volume set on which to restart a full restore (see the **r** key above). This allows *restore* to be interrupted and then restarted.

x The named files are extracted from the tape. If the named file matches a directory whose contents had been written onto the tape, and the **h** key is not specified, the directory is recursively extracted. The owner, modification time, and mode are restored (if possible). If no file argument is given, then the root directory is extracted, which results in the entire content of the tape being extracted, unless the **h** key has been specified.

t The names of the specified files are listed if they occur on the tape. If no file argument is given, then the root directory is listed, which results in the entire content of the tape being listed, unless the **h** key has been specified. Note that the **t** key replaces the function of the old *dumpdir* program.

i This mode allows interactive restoration of files from a dump tape. After reading in the directory information from the tape, *restore* provides a shell like interface that allows the user to move around the directory tree selecting files to be extracted. The available commands are given below; for those commands that require an argument, the default is the current directory.

ls [arg] – List the current or specified directory. Entries that are directories are appended with a "/". Entries that have been marked for extraction are prepended with a "*". If the verbose key is set the inode number of each entry is also listed.

cd arg – Change the current working directory to the specified argument.

pwd – Print the full pathname of the current working directory.

add [arg] – The current directory or specified argument is added to the list of files to be extracted. If a directory is specified, then it and all its descendents are added to the extraction list (unless the **h** key is specified on the command line). Files that are on the extraction list are prepended with a "*" when they are listed by **ls**.

delete [arg] – The current directory or specified argument is deleted from the list of files to be extracted. If a directory is specified, then it and all its descendents are deleted from the extraction list (unless the **h** key is specified on the command line). The most expedient way to extract most of the files from a directory is to add the directory to the extraction list and then delete those files that are not needed.

extract – All the files that are on the extraction list are extracted from the dump tape. *Restore* will ask which volume the user wishes to mount. The fastest way to extract a few files is to start with the last volume, and work towards the first volume.

verbose – The sense of the **v** key is toggled. When set, the verbose key causes the **ls** command to list the inode numbers of all entries. It also causes *restore* to print out information about each file as it is extracted.

help – List a summary of the available commands.

quit – Restore immediately exits, even if the extraction list is not empty.

The following characters may be used in addition to the letter that selects the function desired.

v Normally *restore* does its work silently. The **v** (verbose) key causes it to type the name of each file it treats preceded by its file type.

f The next argument to *restore* is used as the name of the archive instead of /dev/rmt?. If the name of the file is "–", *restore* reads from standard input. Thus, *dump*(8) and *restore* can be used in a pipeline to dump and restore a file system with the command

```
dump 0f - /usr | (cd /mnt; restore xf -)
```

y *Restore* will not ask whether it should abort the restore if gets a tape error. It will always try to skip over the bad tape block(s) and continue as best it can.

m *Restore* will extract by inode numbers rather than by file name. This is useful if only a few files are being extracted, and one wants to avoid regenerating the complete pathname to the file.

h *Restore* extracts the actual directory, rather than the files that it references. This prevents hierarchical restoration of complete subtrees from the tape.

DIAGNOSTICS

Complaints about bad key characters.

Complaints if it gets a read error. If **y** has been specified, or the user responds "y", *restore* will attempt to continue the restore.

If the dump extends over more than one tape, *restore* will ask the user to change tapes. If the **x** or **i** key has been specified, *restore* will also ask which volume the user wishes to mount. The fastest way to extract a few files is to start with the last volume, and work towards the first volume.

There are numerous consistency checks that can be listed by *restore*. Most checks are self-explanatory or can "never happen". Common errors are given below.

Converting to new file system format.
: A dump tape created from the old file system has been loaded. It is automatically converted to the new file system format.

<filename>: not found on tape
: The specified file name was listed in the tape directory, but was not found on the tape. This is caused by tape read errors while looking for the file, and from using a dump tape created on an active file system.

expected next file <inumber>, got <inumber>
: A file that was not listed in the directory showed up. This can occur when using a dump tape created on an active file system.

Incremental tape too low
: When doing incremental restore, a tape that was written before the previous incremental tape, or that has too low an incremental level has been loaded.

Incremental tape too high
: When doing incremental restore, a tape that does not begin its coverage where the previous incremental tape left off, or that has too high an incremental level has been loaded.

Tape read error while restoring <filename>
Tape read error while skipping over inode <inumber>
Tape read error while trying to resynchronize
: A tape read error has occurred. If a file name is specified, then its contents are probably partially wrong. If an inode is being skipped or the tape is trying to resynchronize, then no extracted files have been corrupted, though files may not be found on the tape.

resync restore, skipped <num> blocks
: After a tape read error, *restore* may have to resynchronize itself. This message lists the number of blocks that were skipped over.

FILES

/dev/rmt?	the default tape drive
/tmp/rstdir*	file containing directories on the tape.
/tmp/rstmode*	owner, mode, and time stamps for directories.
./restoresymtab	information passed between incremental restores.

SEE ALSO

rrestore(8C) dump(8), newfs(8), mount(8), mkfs(8)

BUGS

Restore can get confused when doing incremental restores from dump tapes that were made on active file systems.

A level zero dump must be done after a full restore. Because restore runs in user code, it has no control over inode allocation; thus a full restore must be done to get a new set of directories reflecting the new inode numbering, even though the contents of the files is unchanged.

NAME

rexecd – remote execution server

SYNOPSIS

/etc/rexecd

DESCRIPTION

Rexecd is the server for the *rexec*(3X) routine. The server provides remote execution facilities with authentication based on user names and encrypted passwords.

Rexecd listens for service requests at the port indicated in the "exec" service specification; see *services*(5). When a service request is received the following protocol is initiated:

1) The server reads characters from the socket up to a null ('\0') byte. The resultant string is interpreted as an ASCII number, base 10.
2) If the number received in step 1 is non-zero, it is interpreted as the port number of a secondary stream to be used for the **stderr**. A second connection is then created to the specified port on the client's machine.
3) A null terminated user name of at most 16 characters is retrieved on the initial socket.
4) A null terminated, encrypted, password of at most 16 characters is retrieved on the initial socket.
5) A null terminated command to be passed to a shell is retrieved on the initial socket. The length of the command is limited by the upper bound on the size of the system's argument list.
6) *Rexecd* then validates the user as is done at login time and, if the authentication was successful, changes to the user's home directory, and establishes the user and group protections of the user. If any of these steps fail the connection is aborted with a diagnostic message returned.
7) A null byte is returned on the connection associated with the **stderr** and the command line is passed to the normal login shell of the user. The shell inherits the network connections established by *rexecd*.

DIAGNOSTICS

All diagnostic messages are returned on the connection associated with the **stderr**, after which any network connections are closed. An error is indicated by a leading byte with a value of 1 (0 is returned in step 7 above upon successful completion of all the steps prior to the command execution).

"username too long"
The name is longer than 16 characters.

"password too long"
The password is longer than 16 characters.

"command too long "
The command line passed exceeds the size of the argument list (as configured into the system).

"Login incorrect."
No password file entry for the user name existed.

"Password incorrect."
The wrong was password supplied.

"No remote directory."
The *chdir* command to the home directory failed.

"Try again."
A *fork* by the server failed.

"/bin/sh: ..."
The user's login shell could not be started.

BUGS

Indicating "Login incorrect" as opposed to "Password incorrect" is a security breach which allows people to probe a system for users with null passwords.

A facility to allow all data exchanges to be encrypted should be present.

NAME

rlogind – remote login server

SYNOPSIS

/etc/rlogind [**–d**]

DESCRIPTION

Rlogind is the server for the *rlogin*(1C) program. The server provides a remote login facility with authentication based on privileged port numbers.

Rlogind listens for service requests at the port indicated in the "login" service specification; see *services*(5). When a service request is received the following protocol is initiated:

1) The server checks the client's source port. If the port is not in the range 0-1023, the server aborts the connection.

2) The server checks the client's source address. If the address is associated with a host for which no corresponding entry exists in the host name data base (see *hosts*(5)), the server aborts the connection.

Once the source port and address have been checked, *rlogind* allocates a pseudo terminal (see *pty*(4)), and manipulates file descriptors so that the slave half of the pseudo terminal becomes the **stdin** , **stdout** , and **stderr** for a login process. The login process is an instance of the *login*(1) program, invoked with the **–r** option. The login process then proceeds with the authentication process as described in *rshd*(8C), but if automatic authentication fails, it reprompts the user to login as one finds on a standard terminal line.

The parent of the login process manipulates the master side of the pseduo terminal, operating as an intermediary between the login process and the client instance of the *rlogin* program. In normal operation, the packet protocol described in *pty*(4) is invoked to provide ^S/^Q type facilities and propagate interrupt signals to the remote programs. The login process propagates the client terminal's baud rate and terminal type, as found in the environment variable, "TERM"; see *environ*(7).

DIAGNOSTICS

All diagnostic messages are returned on the connection associated with the **stderr**, after which any network connections are closed. An error is indicated by a leading byte with a value of 1.

"Hostname for your address unknown."
No entry in the host name database existed for the client's machine.

"Try again."
A *fork* by the server failed.

"/bin/sh: ..."
The user's login shell could not be started.

BUGS

The authentication procedure used here assumes the integrity of each client machine and the connecting medium. This is insecure, but is useful in an "open" environment.

A facility to allow all data exchanges to be encrypted should be present.

NAME

rmt – remote magtape protocol module

SYNOPSIS

/etc/rmt

DESCRIPTION

Rmt is a program used by the remote dump and restore programs in manipulating a magnetic tape drive through an interprocess communication connection. *Rmt* is normally started up with an *rexec*(3X) or *rcmd*(3X) call.

The *rmt* program accepts requests specific to the manipulation of magnetic tapes, performs the commands, then responds with a status indication. All responses are in ASCII and in one of two forms. Successful commands have responses of

A*number*\n

where *number* is an ASCII representation of a decimal number. Unsuccessful commands are responded to with

E*error-number*\n*error-message*\n,

where *error-number* is one of the possible error numbers described in *intro*(2) and *error-message* is the corresponding error string as printed from a call to *perror*(3). The protocol is comprised of the following commands (a space is present between each token).

O device mode
: Open the specified *device* using the indicated *mode*. *Device* is a full pathname and *mode* is an ASCII representation of a decimal number suitable for passing to *open*(2). If a device had already been opened, it is closed before a new open is performed.

C device
: Close the currently open device. The *device* specified is ignored.

L whence offset
: Perform an *lseek*(2) operation using the specified parameters. The response value is that returned from the *lseek* call.

W count
: Write data onto the open device. *Rmt* reads *count* bytes from the connection, aborting if a premature end-of-file is encountered. The response value is that returned from the *write*(2) call.

R count
: Read *count* bytes of data from the open device. If *count* exceeds the size of the data buffer (10 kilobytes), it is truncated to the data buffer size. *Rmt* then performs the requested *read*(2) and responds with **A***count-read*\n if the read was successful; otherwise an error in the standard format is returned. If the read was successful, the data read is then sent.

I operation count
: Perform a MTIOCOP *ioctl*(2) command using the specified parameters. The parameters are interpreted as the ASCII representations of the decimal values to place in the *mt_op* and *mt_count* fields of the structure used in the *ioctl* call. The return value is the *count* parameter when the operation is successful.

S
: Return the status of the open device, as obtained with a MTIOCGET *ioctl* call. If the operation was successful, an "ack" is sent with the size of the status buffer, then the status buffer is sent (in binary).

Any other command causes *rmt* to exit.

DIAGNOSTICS

All responses are of the form described above.

SEE ALSO

rcmd(3X), rexec(3X), mtio(4), rdump(8C), rrestore(8C)

BUGS

People tempted to use this for a remote file access protocol are discouraged.

NAME

route – manually manipulate the routing tables

SYNOPSIS

/etc/route [**–f**] [*command args*]

DESCRIPTION

Route is a program used to manually manipulate the network routing tables. It normally is not needed, as the system routing table management daemon, *routed*(8C), should tend to this task.

Route accepts three commands: *add*, to add a route; *delete*, to delete a route; and *change*, to modify an existing route.

All commands have the following syntax:

> **/etc/route** *command* **destination gateway** [**metric**]

where *destination* is a host or network for which the route is "to", *gateway* is the gateway to which packets should be addressed, and *metric* is an optional count indicating the number of hops to the *destination*. If no metric is specified, *route* assumes a value of 0. Routes to a particular host are distinguished from those to a network by interpreting the Internet address associated with *destination*. If the *destination* has a "local address part" of INADDR_ANY, then the route is assumed to be to a network; otherwise, it is presumed to be a route to a host. If the route is to a destination connected via a gateway, the *metric* should be greater than 0. All symbolic names specified for a *destination* or *gateway* are looked up first in the host name database, *hosts*(5). If this lookup fails, the name is then looked for in the network name database, *networks*(5).

Route uses a raw socket and the SIOCADDRT and SIOCDELRT *ioctl*'s to do its work. As such, only the super-user may modify the routing tables.

If the **–f** option is specified, *route* will "flush" the routing tables of all gateway entries. If this is used in conjunction with one of the commands described above, the tables are flushed prior to the command's application.

DIAGNOSTICS

"add %s: gateway %s flags %x"
The specified route is being added to the tables. The values printed are from the routing table entry supplied in the *ioctl* call.

"delete %s: gateway %s flags %x"
As above, but when deleting an entry.

"%s %s done"
When the **–f** flag is specified, each routing table entry deleted is indicated with a message of this form.

"not in table"
A delete operation was attempted for an entry which wasn't present in the tables.

"routing table overflow"
An add operation was attempted, but the system was low on resources and was unable to allocate memory to create the new entry.

SEE ALSO

intro(4N), routed(8C)

BUGS

The change operation is not implemented, one should add the new route, then delete the old one.

NAME

routed – network routing daemon

SYNOPSIS

/etc/routed [**–s**] [**–q**] [**–t**] [*logfile*]

DESCRIPTION

Routed is invoked at boot time to manage the network routing tables. The routing daemon uses a variant of the Xerox NS Routing Information Protocol in maintaining up to date kernel routing table entries.

In normal operation *routed* listens on *udp*(4P) socket 520 (decimal) for routing information packets. If the host is an internetwork router, it periodically supplies copies of its routing tables to any directly connected hosts and networks.

When *routed* is started, it uses the SIOCGIFCONF *ioctl* to find those directly connected interfaces configured into the system and marked "up" (the software loopback interface is ignored). If multiple interfaces are present, it is assumed the host will forward packets between networks. *Routed* then transmits a *request* packet on each interface (using a broadcast packet if the interface supports it) and enters a loop, listening for *request* and *response* packets from other hosts.

When a *request* packet is received, *routed* formulates a reply based on the information maintained in its internal tables. The *response* packet generated contains a list of known routes, each marked with a "hop count" metric (a count of 16, or greater, is considered "infinite"). The metric associated with each route returned provides a metric *relative to the sender*.

Response packets received by *routed* are used to update the routing tables if one of the following conditions is satisfied:

(1) No routing table entry exists for the destination network or host, and the metric indicates the destination is "reachable" (i.e. the hop count is not infinite).

(2) The source host of the packet is the same as the router in the existing routing table entry. That is, updated information is being received from the very internetwork router through which packets for the destination are being routed.

(3) The existing entry in the routing table has not been updated for some time (defined to be 90 seconds) and the route is at least as cost effective as the current route.

(4) The new route describes a shorter route to the destination than the one currently stored in the routing tables; the metric of the new route is compared against the one stored in the table to decide this.

When an update is applied, *routed* records the change in its internal tables and generates a *response* packet to all directly connected hosts and networks. *Routed* waits a short period of time (no more than 30 seconds) before modifying the kernel's routing tables to allow possible unstable situations to settle.

In addition to processing incoming packets, *routed* also periodically checks the routing table entries. If an entry has not been updated for 3 minutes, the entry's metric is set to infinity and marked for deletion. Deletions are delayed an additional 60 seconds to insure the invalidation is propagated throughout the internet.

Hosts acting as internetwork routers gratuitously supply their routing tables every 30 seconds to all directly connected hosts and networks.

Supplying the **–s** option forces *routed* to supply routing information whether it is acting as an internetwork router or not. The **–q** option is the opposite of the **–s** option. If the **–t** option is specified, all packets sent or received are printed on the standard output. In addition, *routed* will not divorce itself from the controlling terminal so that interrupts from the keyboard will kill the process. Any other argument supplied is interpreted as the name of file in which *routed*'s actions should be logged. This log contains information about any changes to the routing tables and a history of recent messages sent and received which are related to the changed route.

In addition to the facilities described above, *routed* supports the notion of "distant" *passive* and *active* gateways. When *routed* is started up, it reads the file */etc/gateways* to find gateways which may not be identified using the SIOGIFCONF *ioctl*. Gateways specified in this manner should be marked passive if they are not expected to exchange routing information, while gateways marked active should be willing to exchange routing information (i.e. they should have a *routed* process running on the machine). Passive gateways are maintained in the routing tables forever and information regarding their existence is included in any routing information transmitted. Active gateways are treated equally to network interfaces. Routing information is distributed to the gateway and if no routing information is received for a period of the time, the associated route is deleted.

The */etc/gateways* is comprised of a series of lines, each in the following format:

< **net** | **host** > *name1* **gateway** *name2*
metric *value* < **passive** | **active** >

The **net** or **host** keyword indicates if the route is to a network or specific host.

Name1 is the name of the destination network or host. This may be a symbolic name located in */etc/networks* or */etc/hosts*, or an Internet address specified in "dot" notation; see *inet*(3N).

Name2 is the name or address of the gateway to which messages should be forwarded.

Value is a metric indicating the hop count to the destination host or network.

The keyword **passive** or **active** indicates if the gateway should be treated as *passive* or *active* (as described above).

FILES

/etc/gateways for distant gateways

SEE ALSO

"Internet Transport Protocols", XSIS 028112, Xerox System Integration Standard.
udp(4P)

BUGS

The kernel's routing tables may not correspond to those of *routed* for short periods of time while processes utilizing existing routes exit; the only remedy for this is to place the routing process in the kernel.

Routed should listen to intelligent interfaces, such as an IMP, and to error protocols, such as ICMP, to gather more information.

NAME
rrestore – restore a file system dump across the network

SYNOPSIS
/etc/rrestore [key [name ...]

DESCRIPTION
Rrestore obtains from magnetic tape files saved by a previous *dump*(8). The command is identical in operation to *restore*(8) except the *f* key should be specified and the file supplied should be of the form *machine:device*.

Rrestore creates a remote server, */etc/rmt*, on the client machine to access the tape device.

SEE ALSO
restore(8), rmt(8C)

DIAGNOSTICS
Same as *restore*(8) with a few extra related to the network.

BUGS

NAME

rshd – remote shell server

SYNOPSIS

/etc/rshd

DESCRIPTION

Rshd is the server for the *rcmd*(3X) routine and, consequently, for the *rsh*(1C) program. The server provides remote execution facilities with authentication based on privileged port numbers.

Rshd listens for service requests at the port indicated in the "cmd" service specification; see *services*(5). When a service request is received the following protocol is initiated:

1) The server checks the client's source port. If the port is not in the range 0-1023, the server aborts the connection.

2) The server reads characters from the socket up to a null ('\0') byte. The resultant string is interpreted as an ASCII number, base 10.

3) If the number received in step 1 is non-zero, it is interpreted as the port number of a secondary stream to be used for the **stderr**. A second connection is then created to the specified port on the client's machine. The source port of this second connection is also in the range 0-1023.

4) The server checks the client's source address. If the address is associated with a host for which no corresponding entry exists in the host name data base (see *hosts*(5)), the server aborts the connection.

5) A null terminated user name of at most 16 characters is retrieved on the initial socket. This user name is interpreted as a user identity to use on the **server**'s machine.

6) A null terminated user name of at most 16 characters is retrieved on the initial socket. This user name is interpreted as the user identity on the **client**'s machine.

7) A null terminated command to be passed to a shell is retrieved on the initial socket. The length of the command is limited by the upper bound on the size of the system's argument list.

8) *Rshd* then validates the user according to the following steps. The remote user name is looked up in the password file and a *chdir* is performed to the user's home directory. If either the lookup or *chdir* fail, the connection is terminated. If the user is not the super-user, (user id 0), the file */etc/hosts.equiv* is consulted for a list of hosts considered "equivalent". If the client's host name is present in this file, the authentication is considered successful. If the lookup fails, or the user is the super-user, then the file *.rhosts* in the home directory of the remote user is checked for the machine name and identity of the user on the client's machine. If this lookup fails, the connection is terminated.

9) A null byte is returned on the connection associated with the **stderr** and the command line is passed to the normal login shell of the user. The shell inherits the network connections established by *rshd*.

DIAGNOSTICS

All diagnostic messages are returned on the connection associated with the **stderr**, after which any network connections are closed. An error is indicated by a leading byte with a value of 1 (0 is returned in step 9 above upon successful completion of all the steps prior to the command execution).

"locuser too long"
The name of the user on the client's machine is longer than 16 characters.

"remuser too long"
The name of the user on the remote machine is longer than 16 characters.

"command too long "
The command line passed exceeds the size of the argument list (as configured into the system).

"Hostname for your address unknown."
No entry in the host name database existed for the client's machine.

"Login incorrect."
No password file entry for the user name existed.

"No remote directory."
The *chdir* command to the home directory failed.

"Permission denied."
The authentication procedure described above failed.

"Can't make pipe."
The pipe needed for the **stderr**, wasn't created.

"Try again."
A *fork* by the server failed.

"/bin/sh: ..."
The user's login shell could not be started.

SEE ALSO

rsh(1C), rcmd(3X)

BUGS

The authentication procedure used here assumes the integrity of each client machine and the connecting medium. This is insecure, but is useful in an "open" environment.

A facility to allow all data exchanges to be encrypted should be present.

NAME

rwhod – system status server

SYNOPSIS

/etc/rwhod

DESCRIPTION

Rwhod is the server which maintains the database used by the *rwho*(1C) and *ruptime*(1C) programs. Its operation is predicated on the ability to *broadcast* messages on a network.

Rwhod operates as both a producer and consumer of status information. As a producer of information it periodically queries the state of the system and constructs status messages which are broadcast on a network. As a consumer of information, it listens for other *rwhod* servers' status messages, validating them, then recording them in a collection of files located in the directory */usr/spool/rwho*.

The *rwho* server transmits and receives messages at the port indicated in the "rwho" service specification, see *services*(5). The messages sent and received, are of the form:

```
struct outmp {
        char    out_line[8];/* tty name */
        char    out_name[8];/* user id */
        long    out_time;/* time on */
};

struct whod {
        char    wd_vers;
        char    wd_type;
        char    wd_fill[2];
        int     wd_sendtime;
        int     wd_recvtime;
        char    wd_hostname[32];
        int     wd_loadav[3];
        int     wd_boottime;
        struct  whoent {
                struct  outmp we_utmp;
                int     we_idle;
        } wd_we[1024 / sizeof (struct whoent)];
};
```

All fields are converted to network byte order prior to transmission. The load averages are as calculated by the *w*(1) program, and represent load averages over the 5, 10, and 15 minute intervals prior to a server's transmission. The host name included is that returned by the *gethostname*(2) system call. The array at the end of the message contains information about the users logged in to the sending machine. This information includes the contents of the *utmp*(5) entry for each non-idle terminal line and a value indicating the time since a character was last received on the terminal line.

Messages received by the *rwho* server are discarded unless they originated at a *rwho* server's port. In addition, if the host's name, as specified in the message, contains any unprintable ASCII characters, the message is discarded. Valid messages received by *rwhod* are placed in files named *whod.hostname* in the directory */usr/spool/rwho*. These files contain only the most recent message, in the format described above.

Status messages are generated approximately once every 60 seconds. *Rwhod* performs an *nlist*(3) on /vmunix every 10 minutes to guard against the possibility that this file is not the system image currently operating.

SEE ALSO

rwho(1C), ruptime(1C)

BUGS

Should relay status information between networks. People often interpret the server dieing as a machine going down.

NAME

sa, accton – system accounting

SYNOPSIS

/etc/sa [**–abcdDfijkKlnrstuv**] [file]

/etc/accton [file]

DESCRIPTION

With an argument naming an existing *file, accton* causes system accounting information for every process executed to be placed at the end of the file. If no argument is given, accounting is turned off.

Sa reports on, cleans up, and generally maintains accounting files.

Sa is able to condense the information in */usr/adm/acct* into a summary file */usr/adm/savacct* which contains a count of the number of times each command was called and the time resources consumed. This condensation is desirable because on a large system */usr/adm/acct* can grow by 100 blocks per day. The summary file is normally read before the accounting file, so the reports include all available information.

If a file name is given as the last argument, that file will be treated as the accounting file; */usr/adm/acct* is the default.

Output fields are labeled: "cpu" for the sum of user+system time (in minutes), "re" for real time (also in minutes), "k" for cpu-time averaged core usage (in 1k units), "avio" for average number of i/o operations per execution. With options fields labeled "tio" for total i/o operations, "k*sec" for cpu storage integral (kilo-core seconds), "u" and "s" for user and system cpu time alone (both in minutes) will sometimes appear.

There are near a googol of options:

a	Place all command names containing unprintable characters and those used only once under the name '***other.'
b	Sort output by sum of user and system time divided by number of calls. Default sort is by sum of user and system times.
c	Besides total user, system, and real time for each command print percentage of total time over all commands.
d	Sort by average number of disk i/o operations.
D	Print and sort by total number of disk i/o operations.
f	Force no interactive threshold compression with –v flag.
i	Don't read in summary file.
j	Instead of total minutes time for each category, give seconds per call.
k	Sort by cpu-time average memory usage.
K	Print and sort by cpu-storage integral.
l	Separate system and user time; normally they are combined.
m	Print number of processes and number of CPU minutes for each user.
n	Sort by number of calls.
r	Reverse order of sort.
s	Merge accounting file into summary file */usr/adm/savacct* when done.
t	For each command report ratio of real time to the sum of user and system times.

u	Superseding all other flags, print for each command in the accounting file the user ID and command name.
v	Followed by a number *n*, types the name of each command used *n* times or fewer. Await a reply from the terminal; if it begins with 'y', add the command to the category '**junk**.' This is used to strip out garbage.

FILES

/usr/adm/acct	raw accounting
/usr/adm/savacct	summary
/usr/adm/usracct	per-user summary

SEE ALSO

ac(8), acct(2)

BUGS

The number of options to this program is absurd.

NAME

sccstorcs – build RCS file from SCCS file

SYNOPSIS

sccstorcs [**–t**] [**–v**] *s.file* ...

DESCRIPTION

sccstorcs builds an RCS file from each SCCS file argument. The deltas and comments for each delta are preserved and installed into the new RCS file in order. Also preserved are the user access list and descriptive text, if any, from the SCCS file.

The following flags are meaningful:

–t Trace only. Prints detailed information about the SCCS file and lists the commands that would be executed to produce the RCS file. No commands are actually executed and no RCS file is made.

–v Verbose. Prints each command that is run while it is building the RCS file.

FILES

For each *s.somefile, sccstorcs* writes the files *somefile* and *somefile.v* which should not already exist. *sccstorcs* will abort, rather than overwrite those files if they do exist.

SEE ALSO

ci (1), co (1), rcs (1).
Walter F. Tichy, "Design, Implementation, and Evaluation of a Revised Control System," in *Proceedings of the 6th International Conference on Software Engineering,* IEEE, Tokyo, Sept. 1982.

DIAGNOSTICS

All diagnostics are written to stderr. Non-zero exit status on error.

BUGS

sccstorcs does not preserve all SCCS options specified in the SCCS file. Most notably, it does not preserve removed deltas, MR numbers, and cutoff points.

AUTHOR

Ken Greer
Copyright © 1983 by Kenneth L. Greer

NAME

sendmail – send mail over the internet

SYNOPSIS

/usr/lib/sendmail [flags] [address ...]

newaliases

mailq

DESCRIPTION

Sendmail sends a message to one or more people, routing the message over whatever networks are necessary. *Sendmail* does internetwork forwarding as necessary to deliver the message to the correct place.

Sendmail is not intended as a user interface routine; other programs provide user-friendly front ends; *sendmail* is used only to deliver pre-formatted messages.

With no flags, *sendmail* reads its standard input up to a control-D or a line with a single dot and sends a copy of the letter found there to all of the addresses listed. It determines the network to use based on the syntax and contents of the addresses.

Local addresses are looked up in a file and aliased appropriately. Aliasing can be prevented by preceding the address with a backslash. Normally the sender is not included in any alias expansions, e.g., if 'john' sends to 'group', and 'group' includes 'john' in the expansion, then the letter will not be delivered to 'john'.

Flags are:

–ba	Go into ARPANET mode. All input lines must end with a CR-LF, and all messages will be generated with a CR-LF at the end. Also, the "From:" and "Sender:" fields are examined for the name of the sender.
–bd	Run as a daemon. This requires Berkeley IPC.
–bi	Initialize the alias database.
–bm	Deliver mail in the usual way (default).
–bp	Print a listing of the queue.
–bs	Use the SMTP protocol as described in RFC821. This flag implies all the operations of the **–ba** flag that are compatible with SMTP.
–bt	Run in address test mode. This mode reads addresses and shows the steps in parsing; it is used for debugging configuration tables.
–bv	Verify names only – do not try to collect or deliver a message. Verify mode is normally used for validating users or mailing lists.
–bz	Create the configuration freeze file.
–C*file*	Use alternate configuration file.
–d*X*	Set debugging value to *X*.
–F*fullname*	Set the full name of the sender.
–f*name*	Sets the name of the "from" person (i.e., the sender of the mail). **–f** can only be used by the special users *root, daemon,* and *network,* or if the person you are trying to become is the same as the person you are.
–h*N*	Set the hop count to *N*. The hop count is incremented every time the mail is processed. When it reaches a limit, the mail is returned with an error

	message, the victim of an aliasing loop.
-n	Don't do aliasing.
-o*x value*	Set option *x* to the specified *value*. Options are described below.
-q[*time*]	Processed saved messages in the queue at given intervals. If is omitted, process the queue once. is given as a tagged number, with 's' being seconds, 'm' being minutes, 'h' being hours, 'd' being days, and 'w' being weeks. For example, "-q1h30m" or "-q90m" would both set the timeout to one hour thirty minutes.
-r*name*	An alternate and obsolete form of the **-f** flag.
-t	Read message for recipients. To:, Cc:, and Bcc: lines will be scanned for people to send to. The Bcc: line will be deleted before transmission. Any addresses in the argument list will be suppressed.
-v	Go into verbose mode. Alias expansions will be announced, etc.

There are also a number of processing options that may be set. Normally these will only be used by a system administrator. Options may be set either on the command line using the **-o** flag or in the configuration file. These are described in detail in the *Installation and Operation Guide*. The options are:

A*file*	Use alternate alias file.
c	On mailers that are considered "expensive" to connect to, don't initiate immediate connection. This requires queueing.
d*x*	Set the delivery mode to *x*. Delivery modes are 'i' for interactive (synchronous) delivery, 'b' for background (asynchronous) delivery, and 'q' for queue only – i.e., actual delivery is done the next time the queue is run.
D	Try to automatically rebuild the alias database if necessary.
e*x*	Set error processing to mode *x*. Valid modes are 'm' to mail back the error message, 'w' to "write" back the error message (or mail it back if the sender is not logged in), 'p' to print the errors on the terminal (default), 'q' to throw away error messages (only exit status is returned), and 'e' to do special processing for the BerkNet. If the text of the message is not mailed back by modes 'm' or 'w' and if the sender is local to this machine, a copy of the message is appended to the file "dead.letter" in the sender's home directory.
F*mode*	The mode to use when creating temporary files.
f	Save UNIX-style From lines at the front of messages.
g*N*	The default group id to use when calling mailers.
H*file*	The SMTP help file.
i	Do not take dots on a line by themselves as a message terminator.
L*n*	The log level.
m	Send to "me" (the sender) also if I am in an alias expansion.
o	If set, this message may have old style headers. If not set, this message is guaranteed to have new style headers (i.e., commas instead of spaces between addresses). If set, an adaptive algorithm is used that will correctly determine the header format in most cases.
Q*queuedir*	Select the directory in which to queue messages.

r*timeout*	The timeout on reads; if none is set, *sendmail* will wait forever for a mailer.
S*file*	Save statistics in the named file.
s	Always instantiate the queue file, even under circumstances where it is not strictly necessary.
T*time*	Set the timeout on messages in the queue to the specified time. After sitting in the queue for this amount of time, they will be returned to the sender. The default is three days.
t*stz,dtz*	Set the name of the time zone.
u*N*	Set the default user id for mailers.

If the first character of the user name is a vertical bar, the rest of the user name is used as the name of a program to pipe the mail to. It may be necessary to quote the name of the user to keep *sendmail* from suppressing the blanks from between arguments.

Sendmail returns an exit status describing what it did. The codes are defined in <*sysexits.h*>

EX_OK	Successful completion on all addresses.
EX_NOUSER	User name not recognized.
EX_UNAVAILABLE	Catchall meaning necessary resources were not available.
EX_SYNTAX	Syntax error in address.
EX_SOFTWARE	Internal software error, including bad arguments.
EX_OSERR	Temporary operating system error, such as "cannot fork".
EX_NOHOST	Host name not recognized.
EX_TEMPFAIL	Message could not be sent immediately, but was queued.

If invoked as *newaliases, sendmail* will rebuild the alias database. If invoked as *mailq, sendmail* will print the contents of the mail queue.

FILES

Except for /usr/lib/sendmail.cf, these pathnames are all specified in /usr/lib/sendmail.cf. Thus, these values are only approximations.

/usr/lib/aliases	raw data for alias names
/usr/lib/aliases.pag	
/usr/lib/aliases.dir	data base of alias names
/usr/lib/sendmail.cf	configuration file
/usr/lib/sendmail.fc	frozen configuration
/usr/lib/sendmail.hf	help file
/usr/lib/sendmail.st	collected statistics
/usr/bin/uux	to deliver uucp mail
/usr/net/bin/v6mail	to deliver local mail
/usr/net/bin/sendberkmail	to deliver Berknet mail
/usr/lib/mailers/arpa	to deliver ARPANET mail
/usr/spool/mqueue/*	temp files

SEE ALSO

biff(1), binmail(1), mail(1), aliases(5), sendmail.cf(5), rmail(1), mailaddr(7);
DARPA Internet Request For Comments RFC819, RFC821, RFC822;
Sendmail - An Internetwork Mail Router;
Sendmail Installation and Operation Guide.

BUGS

Sendmail converts blanks in addresses to dots. This is incorrect according to the old ARPANET mail protocol RFC733 (NIC 41952), but is consistent with the new protocols (RFC822).

NAME

shutdown – close down the system at a given time

SYNOPSIS

/etc/shutdown [**–k**] [**–r**] [**–h**] time [warning-message ...]

DESCRIPTION

Shutdown provides an automated shutdown procedure which a super-user can use to notify users nicely when the system is shutting down, saving them from system administrators, hackers, and gurus, who would otherwise not bother with niceties.

Time is the time at which *shutdown* will bring the system down and may be the word **now** (indicating an immediate shutdown) or specify a future time in one of two formats: +number and hour:min. The first form brings the system down in *number* minutes and the second brings the system down at the time of day indicated (as a 24–hour clock).

At intervals which get closer together as apocalypse approaches, warning messages are displayed at the terminals of all users on the system. Five minutes before shutdown, or immediately if shutdown is in less than 5 minutes, logins are disabled by creating /etc/nologin and writing a message there. If this file exists when a user attempts to log in, *login*(1) prints its contents and exits. The file is removed just before *shutdown* exits.

At shutdown time a message is written in the file /usr/adm/shutdownlog, containing the time of shutdown, who ran shutdown and the reason. Then a terminate signal is sent at *init* to bring the system down to single-user state. Alternatively, if **–r, –h,** or **–k** was used, then *shutdown* will exec *reboot*(8), *halt*(8), or avoid shutting the system down (respectively). (If it isn't obvious, **–k** is to make people *think* the system is going down!)

The time of the shutdown and the warning message are placed in /etc/nologin and should be used to inform the users about when the system will be back up and why it is going down (or anything else).

If a shutdown is cancelled and killed (by killing the shutdown process ID number) before the shutdown has been executed, the file */etc/nologin* must be removed to allow users who may have logged off the system to log back in.

FILES

/etc/nologin	tells login not to let anyone log in
/usr/adm/shutdownlog	log file for successful shutdowns.

SEE ALSO

login(1), reboot(8)

BUGS

Only allows you to kill the system between now and 23:59 if you use the absolute time for shutdown.

NAME

slattach – attach serial lines as network interfaces

SYOPNSIS

/etc/slattach ttyname [*baudrate*]

DESCRIPTION

Slattach is used to assign a tty line to a network interface, and to define the network source and destination addresses. The *ttyname* parameter is a string of the form "ttyXX", or "/dev/ttyXX". The optional *baudrate* parameter is used to set the speed of the connection. If not specified, the default of 9600 is used.

Slattach uses *ioctl(2)* to change the line discipline for the tty to **SLIPDISC** and set the tty speed to *baudrate*. The operating system assigns the first available network interface to the tty. Currently, there are a maximum of five serial line interfaces available, named *sl0* through *sl4*.

Only the super-user may attach a network interface.

To detach the interface, use '**ifconfig** *interface-name* **down**' after killing off the *slattach* process. *interface-name* is the name that is shown by *netstat(1)*.

EXAMPLES

```
/etc/slattach ttya8
/etc/slattach /dev/ttyb0 19200
```

DIAGNOSTICS

Messages indicating the specified interface does not exit, the requested address is unknown, the user is not privileged and tried to alter an interface's configuration.

SEE ALSO

rc(8), intro(4N), netstat(1), ifconfig(8C), tty(4), ioctl(2)

NAME

Standalone mode -- definition of this Sanyo/ICON machine operation mode.

DESCRIPTION

Standalone mode is a special single user mode used at initial system configuration time, or subsequently to run special standalone programs, e.g. format a floppy, or check the integrity of the file system.

Standalone mode centers around the loader, bload(8), and is entered by overriding the automatic boot procedure, when prompted. You will be asked to choose a boot device:

```
Specify boot device:
        Type '0' for first hard disk
        Type '1' for second hard disk
        Type '2' for floppy disk
        Type '3' for cassette tape drive
```

Normally you will choose '0',to boot from the system's default hard disk drive. The loader program will then start running and issue the prompt:

```
ICON loader -- Version 1.0
Load:
```

To run a standalone program you would type:

>stand/(program name) [options]

The '>' tells the loader that this is a standalone program. The loader assumes that the base directory is called "/". Thus "stand/(program name)" is a program in the directory "stand", which is in the directory "/".

SEE ALSO

bload(8), binstl(8), copy(8), dkfmt(8), fsck(8), mkfs(8), park(8)

NAME

sticky – executable files with persistent text

DESCRIPTION

While the ‘sticky bit’, mode 01000 (see *chmod*(2)), is set on a sharable executable file, the text of that file will not be removed from the system swap area. Thus the file does not have to be fetched from the file system upon each execution. As long as a copy remains in the swap area, the original text cannot be overwritten in the file system, nor can the file be deleted. (Directory entries can be removed so long as one link remains.)

Sharable files are made by the **–n** and **–z** options of *ld*(1).

To replace a sticky file that has been used do: (1) Clear the sticky bit with *chmod*(1). (2) Execute the old program to flush the swapped copy. This can be done safely even if others are using it. (3) Overwrite the sticky file. If the file is being executed by any process, writing will be prevented; it suffices to simply remove the file and then rewrite it, being careful to reset the owner and mode with *chmod* and *chown*(2). (4) Set the sticky bit again.

Only the super-user can set the sticky bit.

BUGS

Are self-evident.

Is largely unnecessary on the VAX; matters only for large programs that will page heavily to start, since text pages are normally cached incore as long as possible after all instances of a text image exit.

NAME

swapon – specify additional device for paging and swapping

SYNOPSIS

/etc/swapon –a
/etc/swapon name ...

DESCRIPTION

Swapon is used to specify additional devices on which paging and swapping are to take place. The system begins by swapping and paging on only a single device so that only one disk is required at bootstrap time. Calls to *swapon* normally occur in the system multi-user initialization file */etc/rc* making all swap devices available, so that the paging and swapping activity is interleaved across several devices.

Normally, the **–a** argument is given, causing all devices marked as "sw" swap devices in **/etc/fstab** to be made available.

The second form gives individual block devices as given in the system swap configuration table. The call makes only this space available to the system for swap allocation.

SEE ALSO

swapon(2), init(8)

FILES

/dev/[ru][pk]?b	normal paging devices

BUGS

There is no way to stop paging and swapping on a device. It is therefore not possible to make use of devices which may be dismounted during system operation.

NAME

sync – update the super block

SYNOPSIS

/etc/sync

DESCRIPTION

Sync executes the *sync* system primitive. *Sync* can be called to insure all disk writes have been completed before the processor is halted in a way not suitably done by *reboot*(8) or *halt*(8).

See *sync*(2) for details on the system primitive.

SEE ALSO

sync(2), fsync(2), halt(8), reboot(8), update(8)

NAME

syslog – log systems messages

SYNOPSIS

/etc/syslog [**-m***N*] [**-f***name*] [**-d**]

DESCRIPTION

Syslog reads a datagram socket and logs each line it reads into a set of files described by the configuration file /etc/syslog.conf. *Syslog* configures when it starts up and whenever it receives a hangup signal.

Each message is one line. A message can contain a priority code, marked by a digit in angle braces at the beginning of the line. Priorities are defined in <*syslog.h*>, as follows:

LOG_ALERT	this priority should essentially never be used. It applies only to messages that are so important that every user should be aware of them, e.g., a serious hardware failure.
LOG_SALERT	messages of this priority should be issued only when immediate attention is needed by a qualified system person, e.g., when some valuable system resource dissappears. They get sent to a list of system people.
LOG_EMERG	Emergency messages are not sent to users, but represent major conditions. An example might be hard disk failures. These could be logged in a separate file so that critical conditions could be easily scanned.
LOG_ERR	these represent error conditions, such as soft disk failures, etc.
LOG_CRIT	such messages contain critical information, but which can not be classed as errors, for example, 'su' attempts. Messages of this priority and higher are typically logged on the system console.
LOG_WARNING	issued when an abnormal condition has been detected, but recovery can take place.
LOG_NOTICE	something that falls in the class of "important information"; this class is informational but important enough that you don't want to throw it away casually. Messages without any priority assigned to them are typically mapped into this priority.
LOG_INFO	information level messages. These messages could be thrown away without problems, but should be included if you want to keep a close watch on your system.
LOG_DEBUG	it may be useful to log certain debugging information. Normally this will be thrown away.

It is expected that the kernel will not log anything below LOG_ERR priority.

The configuration file is in two sections separated by a blank line. The first section defines files that *syslog* will log into. Each line contains a single digit which defines the lowest priority (highest numbered priority) that this file will receive, an optional asterisk which guarantees that something gets output at least every 20 minutes, and a pathname. The second part of the file contains a list of users that will be informed on SALERT level messages. For example, the configuration file:

```
5*/dev/console
8/usr/spool/adm/syslog
3/usr/adm/critical
```

```
eric
kridle
kalash
```

logs all messages of priority 5 or higher onto the system console, including timing marks every 20 minutes; all messages of priority 8 or higher into the file /usr/spool/adm/syslog; and all messages of priority 3 or higher into /usr/adm/critical. The users "eric", "kridle", and "kalash" will be informed on any subalert messages.

The flags are:

–m Set the mark interval to *N* (default 20 minutes).

–f Specify an alternate configuration file.

–d Turn on debugging (if compiled in).

To bring *syslog* down, it should be sent a terminate signal. It logs that it is going down and then waits approximately 30 seconds for any additional messages to come in.

There are some special messages that cause control functions. "<*>N" sets the default message priority to *N*. "<$>" causes *syslog* to reconfigure (equivalent to a hangup signal). This can be used in a shell file run automatically early in the morning to truncate the log.

Syslog creates the file /etc/syslog.pid if possible containing a single line with its process id. This can be used to kill or reconfigure *syslog*.

FILES

/etc/syslog.conf – the configuration file
/etc/syslog.pid – the process id

BUGS

LOG_ALERT and LOG_SUBALERT messages should only be allowed to privileged programs.

Actually, *syslog* is not clever enough to deal with kernel error messages in the current implementation.

SEE ALSO

syslog(3)

NAME
talkd – remote user communication server

SYNOPSIS
/etc/talkd

DESCRIPTION
Talkd is the server that notifies a user that somebody else wants to initiate a conversation. It acts a repository of invitations, responding to requests by clients wishing to rendezvous to hold a conversation. In normal operation, a client, the caller, initiates a rendezvous by sending a CTL_MSG to the server of type LOOK_UP (see <*protocols/talkd.h*>). This causes the server to search its invitation tables to check if an invitation currently exists for the caller (to speak to the callee specified in the message). If the lookup fails, the caller then sends an ANNOUNCE message causing the server to broadcast an announcement on the callee's login ports requesting contact. When the callee responds, the local server uses the recorded invitation to respond with the appropriate rendezvous address and the caller and callee client programs establish a stream connection through which the conversation takes place.

SEE ALSO
talk(1), write(1)

NAME

telnetd – DARPA TELNET protocol server

SYNOPSIS

/etc/telnetd [**–d**] [*port*]

DESCRIPTION

Telnetd is a server which supports the DARPA standard TELNET virtual terminal protocol. The TELNET server operates at the port indicated in the "telnet" service description; see *services*(5). This port number may be overridden (for debugging purposes) by specifying a port number on the command line. If the **–d** option is specified, each socket created by *telnetd* will have debugging enabled (see SO_DEBUG in *socket*(2)).

Telnetd operates by allocating a pseudo-terminal device (see *pty*(4)) for a client, then creating a login process which has the slave side of the pseudo-terminal as **stdin**, **stdout**, and **stderr**. *Telnetd* manipulates the master side of the pseudo terminal, implementing the TELNET protocol and passing characters between the client and login process.

When a TELNET session is started up, *telnetd* sends a TELNET option to the client side indicating a willingness to do "remote echo" of characters. The pseudo terminal allocated to the client is configured to operate in "cooked" mode, and with XTABS and CRMOD enabled (see *tty*(4)). Aside from this initial setup, the only mode changes *telnetd* will carry out are those required for echoing characters at the client side of the connection.

Telnetd supports binary mode, and most of the common TELNET options, but does not, for instance, support timing marks. Consult the source code for an exact list of which options are not implemented.

SEE ALSO

telnet(1C)

BUGS

A complete list of the options supported should be given here.

NAME

tftpd – DARPA Trivial File Transfer Protocol server

SYNOPSIS

/etc/tftpd [**–d**] [*port*]

DESCRIPTION

Tftpd is a server which supports the DARPA Trivial File Transfer Protocol. The TFTP server operates at the port indicated in the "tftp" service description; see *services*(5). This port number may be overridden (for debugging purposes) by specifying a port number on the command line. If the **–d** option is specified, each socket created by *tftpd* will have debugging enabled (see SO_DEBUG in *socket*(2)).

The use of *tftp* does not require an account or password on the remote system. Due to the lack of authentication information, *tftpd* will allow only publicly readable files to be accessed. Note that this extends the concept of "public" to include all users on all hosts that can be reached through the network; this may not be appropriate on all systems, and its implications should be considered before enabling tftp service.

SEE ALSO

tftp(1C)

BUGS

This server is known only to be self consistent (i.e. it operates with the user TFTP program, *tftp*(1C)). Due to the unreliability of the transport protocol (UDP) and the scarcity of TFTP implementations, it is uncertain whether it really works.

The search permissions of the directories leading to the files accessed are not checked.

NAME

trpt – transliterate protocol trace

SYNOPSIS

trpt [**–a**] [**–s**] [**–t**] [**–j**] [**–p** hex-address] [system [core]]

DESCRIPTION

Trpt interrogates the buffer of TCP trace records created when a socket is marked for "debugging" (see *setsockopt*(2)), and prints a readable description of these records. When no options are supplied, *trpt* prints all the trace records found in the system grouped according to TCP connection protocol control block (PCB). The following options may be used to alter this behavior.

–s in addition to the normal output, print a detailed description of the packet sequencing information,

–t in addition to the normal output, print the values for all timers at each point in the trace,

–j just give a list of the protocol control block addresses for which there are trace records,

–p show only trace records associated with the protocol control block who's address follows,

–a in addition to the normal output, print the values of the source and destination addresses for each packet recorded.

The recommended use of *trpt* is as follows. Isolate the problem and enable debugging on the socket(s) involved in the connection. Find the address of the protocol control blocks associated with the sockets using the **–A** option to *netstat*(1). Then run *trpt* with the **–p** option, supplying the associated protocol control block addresses. If there are many sockets using the debugging option, the **–j** option may be useful in checking to see if any trace records are present for the socket in question.

If debugging is being performed on a system or core file other than the default, the last two arguments may be used to supplant the defaults.

FILES

/vmunix
/dev/kmem

SEE ALSO

setsockopt(2), netstat(1)

DIAGNOSTICS

"no namelist" when the system image doesn't contain the proper symbols to find the trace buffer; others which should be self explanatory.

BUGS

Should also print the data for each input or output, but this is not saved in the race record.

The output format is inscrutable and should be described here.

NAME

tunefs – tune up an existing file system

SYNOPSIS

/etc/tunefs *tuneup-options special|filesys*

DESCRIPTION

Tunefs is designed to change the dynamic parameters of a file system which affect the layout policies. The parameters which are to be changed are indicated by the flags given below:

–a maxcontig
This specifies the maximum number of contiguous blocks that will be laid out before forcing a rotational delay (see –d below). The default value is one, since most device drivers require an interrupt per disk transfer. Device drivers that can chain several buffers together in a single transfer should set this to the maximum chain length.

–d rotdelay
This specifies the expected time (in milliseconds) to service a transfer completion interrupt and initiate a new transfer on the same disk. It is used to decide how much rotational spacing to place between successive blocks in a file.

–e maxbpg
This indicates the maximum number of blocks any single file can allocate out of a cylinder group before it is forced to begin allocating blocks from another cylinder group. Typically this value is set to about one quarter of the total blocks in a cylinder group. The intent is to prevent any single file from using up all the blocks in a single cylinder group, thus degrading access times for all files subsequently allocated in that cylinder group. The effect of this limit is to cause big files to do long seeks more frequently than if they were allowed to allocate all the blocks in a cylinder group before seeking elsewhere. For file systems with exclusively large files, this parameter should be set higher.

–m minfree
This value specifies the percentage of space held back from normal users; the minimum free space threshold. The default value used is 10%. This value can be set to zero, however up to a factor of three in throughput will be lost over the performance obtained at a 10% threshold. Note that if the value is raised above the current usage level, users will be unable to allocate files until enough files have been deleted to get under the higher threshold.

SEE ALSO

fs(5), newfs(8), mkfs(8)

McKusick, Joy, Leffler; "A Fast File System for Unix", Computer Systems Research Group, Dept of EECS, Berkeley, CA 94720; TR #7, September 1982.

BUGS

This program should work on mounted and active file systems. Because the super-block is not kept in the buffer cache, the program will only take effect if it is run on dismounted file systems. (if run on the root file system, the system must be rebooted)

You can tune a file system, but you can't tune a fish.

NAME

update – periodically update the super block

SYNOPSIS

/etc/update [seconds]

DESCRIPTION

Update is a program that executes the *sync*(2) primitive every 60 seconds. This insures that the file system is fairly up to date in case of a crash. This command should not be executed directly, but should be executed out of the initialization shell command file. If seconds are specified, updating occurs that often instead of every 60 seconds.

SEE ALSO

sync(2), sync(8), init(8), rc(8)

NAME

uuclean – uucp spool directory clean-up

SYNOPSIS

uuclean [option] ...

DESCRIPTION

Uuclean will scan the spool directory for files with the specified prefix and delete all those which are older than the specified number of hours.

The following options are available.

–p*pre*	Scan for files with *pre* as the file prefix. Up to 10 **–p** arguments may be specified. A **–p** without any *pre* following will cause all files older than the specified time to be deleted.
–n*time*	Files whose age is more than *time* hours will be deleted if the prefix test is satisfied. (default time is 72 hours)
–m	Send mail to the owner of the file when it is deleted.

This program will typically be started by *cron*(8).

FILES

/usr/lib/uucp	directory with commands used by uuclean internally
/usr/lib/uucp/spool	spool directory

SEE ALSO

uucp(1C), uux(1C)

NAME
uusnap – show snapshot of the UUCP system

SYNOPSIS
uusnap

DESCRIPTION
Uusnap displays in tabular format a synopsis of the current UUCP situation. The format of each line is as follows:

```
site   N Cmds   N Data   N Xqts   Message
```

Where "site" is the name of the site with work, "N" is a count of each of the three possible types of work (command, data, or remote execute), and "Message" is the current status message for that site as found in the STST file.

Included in "Message" may be the time left before UUCP can re-try the call, and the count of the number of times that UUCP has tried to reach the site.

SEE ALSO
uucp(1C), UUCP Implementation Guide

AUTHOR
Randy King

NAME

uxrc – configuration file for kernel

SYNOPSIS

/etc/uxrc

DESCRIPTION

Uxrc is the configuration file which is used to set kernel variables at boot time.

When a reboot is in progress, */etc/uxrc* is read to change kernel variables from their default state. These kernel variables can be changed by creating a file called /etc/uxrc and adding the appropriate configuration statements.

mot_mode

The tty line driver behavior on the Motorola MC68681 main board ports may be modified by changing the mot_mode variable. The lower 4 bits enable hardware RTS/CTS handshaking and the upper 4 bits enable modem control. To enable RTS/CTS handshaking on port 01 and modem control on port 02 the following line should be added to /etc/uxrc.

```
mot_mode      42
```

pcp_mode0 pcp_mode1

The tty line driver behavior on the pcp board ports may be modified by changing the pcp_mode1 and pcp_mode2 variables. The lower 16 bits enable hardware RTS/CTS handshaking and the upper 16 bits enable modem control. To enable RTS/CTS handshaking on port 09 and modem control on port 0e on pcp board 0 the following line should be added to /etc/uxrc.

```
pcp_mode0     40000200
```

autoboot

To force the machine to reboot automatically on a panic, autoboot must be set. This should not be done unless absolutely necessary because the panic message may scroll off of the console and release 2.15 does not save the panic message before rebooting. To enable this feature, the following line would be added to /etc/uxrc.

```
autoboot      1
```

Other configuration features will be added as the need arises.

BUGS

As of release 2.15, both RTS/CTS handshaking and modem control may not be enabled at the same time.

NAME

vipw – edit the password file

SYNOPSIS

vipw

DESCRIPTION

Vipw edits the password file while setting the appropriate locks, and does any necessary processing after the password file is unlocked. If the password file is already being edited, then you will be told to try again later. The *vi* editor will be used unless the environment variable EDITOR indicates an alternate editor. *Vipw* performs a number of consistency checks on the password entry for *root*, and will not allow a password file with a "mangled" root entry to be installed.

SEE ALSO

chfn(1), chsh(1), passwd(1), passwd(5), adduser(8)

FILES

/etc/ptmp

COMMENTS

ICON/UXB REFERENCE MANUAL Volume 1B **P/N 172-022-002**

Your comments and suggestions are appreciated and will help us to provide you with the very best in system and application documentation. Send your comments to the address at the bottom of this page. Users who respond will be entitled to free updates of this manual for one year.

1. How would you rate this manual for COMPLETENESS? (Please Circle)
   Excellent Poor
   5 ------------- 4 -------------- 3 ------------- 2 -------------- 1 ------------- 0

2. Is there any information that you feel should be included or removed?
   ____
   ____
   ____

3. How would you rate this manual for ACCURACY? (Please Circle)
   Excellent Poor
   5 ------------- 4 -------------- 3 ------------- 2 -------------- 1 ------------- 0

4. Indicate the page number and nature of any error(s) found in this manual.
   ____
   ____
   ____

5. How would you rate this manual for USABILITY? (Please Circle)
   Excellent Poor
   5 ------------- 4 -------------- 3 ------------- 2 -------------- 1 ------------- 0

6. Describe any format or packaging problems you have experienced with this manual and/or binder.
   ____
   ____
   ____
   ____

7. Do you have any general comments or suggestions regarding this publication or future publications?
   ____
   ____
   ____
   ____

Your Name ______________________________
Company ______________________________
Address ______________________________ Phone (____)__________
City & State ______________________________ Zip Code __________
Job Function ______________________________
Type of Equipment Installed: ______________________________