Customer Order Number 424510771-210
NSC Publication Number 424510771-210A
December 1986

Series 32000®

GENIX V.3™ Administrator's Reference Manual

REVISION RECORD

REVISION	RELEASE DATE	SUMMARY OF CHANGES
A	12/86	First Release. *Series 32000*® GENIX V.3™ Administrator's Reference Manual NSC Publication Number 424510771-210A.

Series 32000®

GENIX V.3™ Administrator's Reference Manual

Customer Order Number 424510771-210
NSC Publication Number 424510771-210A
December 1986

PREFACE

This *Administrator's Reference Manual* describes the commands that constitute the basic software running on a *Series 32000®* Computer. This manual is intended to supplement information contained in the *User's Reference Manual* and the *Programmer's Reference Manual* to provide an easy reference volume for those who must administer a GENIX V.3™ system.

The information contained in this manual is for reference only and is subject to change without notice.

Chapter 1

INTRODUCTION

1.1 OVERVIEW

This manual is intended to supplement information contained in the *User's Reference Manual* and the *Programmer's Reference Manual* to provide an easy reference volume for those who must administer a GENIX V.3 system.

The manual is divided into three sections:

1. System Maintenance Commands and Application Programs
7. Special Files
8. System Maintenance Procedures

Throughout this volume each reference of the form *name*(1M), *name*(7), or *name*(8), refers to entries in this manual, while all other references to entries of the form *name*(N), where *N* is a number possibly followed by a letter, refer to entry *name* in Section *N* of the *User's Reference Manual* or the *Programmer's Reference Manual.*

Section 1 (*System Maintenance Commands and Application Programs*) contains commands and programs that are used in administering a GENIX V.3 system. These entries carry a sub-class designation of "1M" for cross-referencing reasons.

Section 7 (*Special Files*) discusses the characteristics of system files that refer to input/output devices. The names in this section generally refer to device names for the hardware, rather than to the names of the special files themselves.

Section 8 (*System Maintenance Procedures*) discusses crash recovery, firmware programs, boot procedures, facility descriptions, etc.

Each section begins with a page labelled *intro.* Entries following the *intro* page are arranged alphabetically and may consist of more than one page. Some entries describe several routines, commands, etc. In such cases, the entry appears only once, alphabetized under its "primary" name. An example of such an entry is **mount**(1M), which also describes the **umount** command.

All entries are based on a common format, not all of whose parts always appear:

- The NAME part gives the name(s) of the entry and briefly states its purpose.
- The SYNOPSIS part summarizes the use of the program being described. A few conventions are used, particularly in Section 1M (*Commands*):
  - **Boldface** strings are literals and are to be typed just as they appear.
  - *Italic* strings usually represent substitutable argument prototypes and program names found elsewhere in the manual. (They are underlined in the typed version of the entries.)
  - Square brackets [ ] around an argument prototype indicate that the argument is optional. When an argument prototype is given as "name" or "file," it always refers to a *file* name.

— Ellipses ... are used to show that the previous argument prototype may be repeated.

- The **DESCRIPTION** provides an overview of the command.
- The **EXAMPLE(S)** part gives example(s) of usage, where appropriate.
- The **FILES** part gives the file names that are built into the program.
- The **SEE ALSO** part gives pointers to related information.
- The **DIAGNOSTICS** part discusses the diagnostic indications that may be produced. Messages that are intended to be self-explanatory are not listed.
- The **WARNINGS** part points out potential pitfalls.
- The **BUGS** part gives known bugs and sometimes deficiencies.

A "Table of Contents" and a "Permuted Index" derived from that table precede Section 1. The "Permuted Index" is a list of keywords, given in the second of three columns, together with the context in which the keyword is found. Keywords are either topical keywords or the names of manual entries. Entries are identified with their section numbers shown in parentheses. The right column lists the name of the manual page on which each keyword may be found. The left column contains useful information about the keyword.

TABLE OF CONTENTS
ADMINISTRATOR'S REFERENCE MANUAL

PERMUTED INDEX — ADMINISTRATOR'S REFERENCE MANUAL

/umountall [—k	]	umountall(1M)
/umountall	[—k]	umountall(1M)
/dsd: *Series*	*32000* computer Integral Disk Subsystem.	dsd(7)
	accept, reject: allow or prevent LP requests.	accept(1M)
/adv: advertise a directory for remote	access.	adv(1M)
/fusage: disk	access profiler.	fusage(1M)
/sadp: disk	access profiler.	sadp(1M)
/dcopy: copy file systems for optimal	access time.	dcopy(1M)
/acctcon1, acctcon2: connect-time	accounting.	acctcon(1M)
/acctprc1, acctprc2: process	accounting.	acctprc(1M)
shutacct, startup, turnacct: shell procedures for	accounting. /prctmp, prdaily, prtacct, runacct,	acctsh(1M)
/runacct: run daily	accounting.	runacct(1M)
/acctdisk, acctdusg, accton, acctwtmp: overview of	accounting and miscellaneous accounting commands.	acct(1M)
acctwtmp: overview of accounting and miscellaneous	accounting commands. /acctdisk, acctdusg, accton,	acct(1M)
/diskusg: generate disk	accounting data by user ID.	diskusg(1M)
/acctmerg: merge or add total	accounting files.	acctmerg(1M)
/acctcms: command summary from per-process	accounting records.	acctcms(1M)
/fwtmp, wtmpfix: manipulate connect	accounting records.	fwtmp(1M)
accounting records.	acctcms: command summary from per-process	acctcms(1M)
	acctcon1, acctcon2: connect-time accounting.	acctcon(1M)
/acctcon1,	acctcon2: connect-time accounting.	acctcon(1M)
accounting and miscellaneous accounting commands.	acctdisk, acctdusg, accton, acctwtmp: overview of	acct(1M)
and miscellaneous accounting commands. /acctdisk,	acctdusg, accton, acctwtmp: overview of accounting	acct(1M)
	acctmerg: merge or add total accounting files.	acctmerg(1M)
miscellaneous accounting/ /acctdisk, acctdusg,	accton, acctwtmp: overview of accounting and	acct(1M)
	acctprc1, acctprc2: process accounting.	acctprc(1M)
/acctprc1,	acctprc2: process accounting.	acctprc(1M)
accounting commands. /acctdisk, acctdusg, accton,	acctwtmp: overview of accounting and miscellaneous	acct(1M)
/killall: kill all	active processes.	killall(1M)
/sar: sa1, sa2, sadc: system	activity report package.	sar(1M)
/acctmerg: merge or	add total accounting files.	acctmerg(1M)
/hdeadd:	add/delete hdelog (Hard Disk Error Log) reports	hdeadd(1M)
/SA: devices	administered by System Administration.	sa(7)
/nlsadmin: network listener service	administration.	nlsadmin(1M)
/rfadmin: Remote File Sharing domain	administration.	rfadmin(1M)
/SA: devices administered by System	Administration.	sa(7)
/sysadm: menu interface to do system	administration.	sysadm(1)
/uadmin:	administrative control.	uadmin(1M)
/swap: swap	administrative interface.	swap(1M)
	adv: advertise a directory for remote access.	adv(1M)
/adv:	advertise a directory for remote access.	adv(1M)
/fumount: forced unmount of an	advertised resource.	fumount(1M)
/intro: introduction to maintenance commands and	application programs.	intro(1M)
/finc: fast incremental	backup	finc(1M)
/ckbupscd: check file system	backup schedule.	ckbupscd(1M)
/frec: recover files from a	backup tape.	frec(1M)
/hdefix: report or change	bad block mapping.	hdefix(1M)
/brc,	bcheckrc: system initialization procedures.	brc(1M)
/su:	become super-user or another user.	su(1M)
/mk: remake the	binary system and commands from source code.	mk(8)
/sync: update the super	block.	sync(1M)
/hdefix: report or change bad	block mapping.	hdefix(1M)
/df: report number of free disk	blocks and i-nodes.	df(1M)
	brc, bcheckrc: system initialization procedures.	brc(1M)
/mknod:	build special file.	mknod(1M)
terminfo description.	captoinfo: convert a termcap description into a	captoinfo(1M)
/hdefix: report or	change bad block mapping.	hdefix(1M)
/rfpasswd:	change Remote File Sharing host password.	rfpasswd(1M)
/chroot:	change root directory for a command.	chroot(1M)
/shutdown: shut down system,	change system state.	shutdown(1M)
/helpadm: make	changes to the Help Facility database.	helpadm(1M)
nulladm, prctmp, prdaily, prtacct, runacct,/	chargefee, ckpacct, dodisk, lastlogin, monacct,	acctsh(1M)
/checkfsys:	check a file system on a removable disk.	checkfsys(1M)
/fsck, dfsck:	check and repair file systems	fsck(1M)
/ckbupscd:	check file system backup schedule.	ckbupscd(1M)
/uucheck:	check the uucp directories and permissions file.	uucheck(1M)
	checkall: faster file system checking procedure.	checkall(1M)

/pwck, grpck: password/group file	checkers.	pwck(1M)
	checkfsys: check a file system on a removable disk.	checkfsys(1M)
/checkall: faster file system	checking procedure.	checkall(1M)
	chroot: change root directory for a command.	chroot(1M)
	ckbupscd: check file system backup schedule.	ckbupscd(1M)
prctmp, prdaily, prtacct, runacct,/ /chargefee,	ckpacct, dodisk, lastlogin, monacct, nulladm,	acctsh(1M)
/uucleanup: uucp spool directory	clean-up.	uucleanup(1M)
/strclean: STREAMS error logger	cleanup program.	strclean(1M)
/clri:	clear i-node.	clri(1M)
/cron:	clock daemon.	cron(1M)
	clone: open any minor device on a STREAMS driver.	clone(7)
	clri: clear i-node.	clri(1M)
remake the binary system and commands from source	code. /mk:	mk(8)
/chroot: change root directory for a	command.	chroot(1M)
/hdelogger: Hard Disk Error status report	command and Log Daemon.	hdelogger(1M)
/uuxqt: execute remote	command requests.	uuxqt(1M)
records. /acctcms:	command summary from per-process accounting	acctcms(1M)
overview of accounting and miscellaneous accounting	commands. /acctdisk, acctdusg, accton, acctwtmp:	acct(1M)
/install: install	commands.	install(1M)
/streamio: STREAMS ioctl	commands.	streamio(7)
/intro: introduction to maintenance	commands and application programs.	intro(1M)
/mk: remake the binary system and	commands from source code.	mk(8)
/rc2: run	commands performed for multi-user environment.	rc2(1M)
/rc0: run	commands performed to stop the operating system.	rc0(1M)
/infocmp:	compare or print out terminfo descriptions.	infocmp(1M)
/tic: terminfo	compiler.	tic(1M)
/dsd: *Series 32000*	computer Integral Disk Subsystem.	dsd(7)
	config - configure a System.	config(1M)
/hdeupdate: display/modify physical disk	configuration.	hdeupdate(1M)
/config -	configure a System.	config(1M)
/lpadmin:	configure the LP spooling system.	lpadmin(1M)
/fwtmp, wtmpfix: manipulate	connect accounting records.	fwtmp(1M)
/acctcon1, acctcon2:	connect-time accounting.	acctcon(1M)
	console: console interface.	console(7)
/console:	console interface.	console(7)
/mkfs:	construct a file system.	mkfs(1M)
/Uutry: try to	contact remote system with debugging on.	Uutry(1M)
/uadmin: administrative	control.	uadmin(1M)
/init, telinit: process	control initialization.	init(1M)
/tty:	controlling terminal interface.	tty(7)
description. /captoinfo:	convert a termcap description into a terminfo	captoinfo(1M)
/dd:	convert and copy a file.	dd(1M)
/timod: Transport Interface	cooperating STREAMS module.	timod(7)
/dd: convert and	copy a file.	dd(1M)
/dcopy:	copy file systems for optimal access time.	dcopy(1M)
/volcopy: make literal	copy of file system.	volcopy(1M)
/mem, kmem:	core memory.	mem(7)
	crash: examine system images.	crash(1M)
/makefsys:	create a file system on a diskette.	makefsys(1M)
	cron: clock daemon.	cron(1M)
/cron: clock	daemon.	cron(1M)
Hard Disk Error status report command and Log	Daemon. /hdelogger:	hdelogger(1M)
/strerr: STREAMS error logger	daemon.	strerr(1M)
/rfudaemon: Remote File Sharing	daemon process.	rfudaemon(1M)
/runacct: run	daily accounting.	runacct(1M)
/diskusg: generate disk accounting	data by user ID.	diskusg(1M)
/helpadm: make changes to the Help Facility	database.	helpadm(1M)
	dcopy: copy file systems for optimal access time.	dcopy(1M)
	dd: convert and copy a file.	dd(1M)
/fsdb: file system	debugger.	fsdb(1M)
/Uutry: try to contact remote system with	debugging on.	Uutry(1M)
/sysdef: output system	definition.	sysdef(1M)
convert a termcap description into a terminfo	description. /captoinfo:	captoinfo(1M)
/captoinfo: convert a termcap	description into a terminfo description.	captoinfo(1M)
/infocmp: compare or print out terminfo	descriptions.	infocmp(1M)
/fstyp:	determine file system identifier.	fstyp(1M)
/devnm:	device name.	devnm(1M)
/clone: open any minor	device on a STREAMS driver.	clone(7)
/SA:	devices administered by System Administration.	sa(7)
	devnm: device name.	devnm(1M)
	df: report number of free disk blocks and i-nodes.	df(1M)
/fsck,	dfsck: check and repair file systems	fsck(1M)

/link, unlink: link and unlink files and	directories.	link(1M)
/uucheck: check the uucp	directories and permissions file.	uucheck(1M)
/mvdir: move a	directory.	mvdir(1M)
/uucleanup: uucp spool	directory clean-up.	uucleanup(1M)
/chroot: change root	directory for a command.	chroot(1M)
/adv: advertise a	directory for remote access.	adv(1M)
/getty: set terminal type, modes, speed, and line	discipline.	getty(1M)
/uugetty: set terminal type, modes, speed, and line	discipline.	uugetty(1M)
/checkfsys: check a file system on a removable	disk.	checkfsys(1M)
/fusage:	disk access profiler.	fusage(1M)
/sadp:	disk access profiler.	sadp(1M)
/diskusg: generate	disk accounting data by user ID.	diskusg(1M)
/df: report number of free	disk blocks and i-nodes.	df(1M)
/hdeupdate: display/modify physical	disk configuration.	hdeupdate(1M)
/hdelog: hard	disk error log interface file.	hdelog(7)
/hdeadd: add/delete hdelog (Hard	Disk Error Log) reports .	hdeadd(1M)
/hdelogger: Hard	Disk Error status report command and Log Daemon.	hdelogger(1M)
/dsd: *Series 32000* computer Integral	Disk Subsystem.	dsd(7)
/du: summarize	disk usage.	du(1M)
/makefsys: create a file system on a	diskette.	makefsys(1M)
/mountfsys, umountfsys: mount, unmount a	diskette file system.	mountfsys(1M)
/fmtflop: physically format	diskettes.	fmtflop(1M)
/ldsysdump: load system dump from a tape or floppy	diskettes.	ldsysdump(1M)
/sysdump: dump system memory image to floppy	disk(s) or tape.	sysdump(8)
	diskusg: generate disk accounting data by user ID.	diskusg(1M)
/rmntstat:	display mounted resource information.	rmntstat(1M)
/hdeupdate:	display/modify physical disk configuration.	hdeupdate(1M)
names.	dname: Print Remote File Sharing domain and network	dname(1M)
prdaily, prtacct, runacct,/ /chargefee, ckpacct,	dodisk, lastlogin, monacct, nulladm, prctmp,	acctsh(1M)
/whodo: who is	doing what.	whodo(1M)
/rfadmin: Remote File Sharing	domain administration.	rfadmin(1M)
/dname: Print Remote File Sharing	domain and network names.	dname(1M)
/clone: open any minor device on a STREAMS	driver.	clone(7)
/sxt: pseudo-device	driver.	sxt(7)
Subsystem.	dsd: *Series 32000* computer Integral Disk	dsd(7)
	du: summarize disk usage.	du(1M)
/s5dump: incremental file system	dump.	s5dump(1M)
/ldsysdump: load system	dump from a tape or floppy diskettes.	ldsysdump(1M)
/sysdump:	dump system memory image to floppy disk(s) or tape.	sysdump(8)
/rc2: run commands performed for multi-user	environment.	rc2(1M)
/rfstop: stop the Remote File Sharing	environment.	rfstop(1M)
/hdelog: hard disk	error log interface file.	hdelog(7)
/hdeadd: add/delete hdelog (Hard Disk	Error Log) reports .	hdeadd(1M)
/strclean: STREAMS	error logger cleanup program.	strclean(1M)
/strerr: STREAMS	error logger daemon.	strerr(1M)
/log: interface to STREAMS	error logging and event tracing.	log(7)
/hdelogger: Hard Disk	Error status report command and Log Daemon.	hdelogger(1M)
/setmnt:	establish mount table.	setmnt(1M)
/log: interface to STREAMS error logging and	event tracing.	log(7)
/crash:	examine system images.	crash(1M)
/uuxqt:	execute remote command requests.	uuxqt(1M)
/helpadm: make changes to the Help	Facility database.	helpadm(1M)
/finc:	fast incremental backup .	finc(1M)
/checkall:	faster file system checking procedure.	checkall(1M)
system.	ff: list file names and statistics for a file	ff(1M)
/dd: convert and copy a	file.	dd(1M)
/hdelog: hard disk error log interface	file.	hdelog(7)
/mknod: build special	file.	mknod(1M)
/null: the null	file.	null(7)
/uucheck: check the uucp directories and permissions	file.	uucheck(1M)
/pwck, grpck: password/group	file checkers.	pwck(1M)
/ff: list	file names and statistics for a file system.	ff(1M)
/fuser: identify processes using a	file or file structure.	fuser(1M)
/rfstart: start Remote	File Sharing.	rfstart(1M)
/rfudaemon: Remote	File Sharing daemon process.	rfudaemon(1M)
/rfadmin: Remote	File Sharing domain administration.	rfadmin(1M)
/dname: Print Remote	File Sharing domain and network names.	dname(1M)
/rfstop: stop the Remote	File Sharing environment.	rfstop(1M)
/rfpasswd: change Remote	File Sharing host password.	rfpasswd(1M)
/nsquery: Remote	File Sharing name server query.	nsquery(1M)
/rfuadmin: Remote	File Sharing notification shell script.	rfuadmin(1M)
/unadv: unadvertise a Remote	File Sharing resource.	unadv(1M)

/sysadm:	menu interface to do system administration.	sysadm(1)
/acctmerg:	merge or add total accounting files.	acctmerg(1M)
/strace: print STREAMS trace	messages.	strace(1M)
/clone: open any	minor device on a STREAMS driver.	clone(7)
accton, acctwtmp: overview of accounting and	miscellaneous accounting commands. /acctdusg,	acct(1M)
source code.	mk: remake the binary system and commands from	mk(8)
	mkfs: construct a file system.	mkfs(1M)
	mknod: build special file.	mknod(1M)
/getty: set terminal type,	modes, speed, and line discipline.	getty(1M)
/uugetty: set terminal type,	modes, speed, and line discipline.	uugetty(1M)
/timod: Transport Interface cooperating STREAMS	module.	timod(7)
Transport Interface read/write interface STREAMS	module. /tirdwr:	tirdwr(7)
runacct,/ /chargefee, ckpacct, dodisk, lastlogin,	monacct, nulladm, prctmp, prdaily, prtacct,	acctsh(1M)
resources. /mount, umount:	mount and unmount file systems and remote	mount(1M)
/setmnt: establish	mount table.	setmnt(1M)
remote resources.	mount, umount: mount and unmount file systems and	mount(1M)
/mountfsys, umountfsys:	mount, unmount a diskette file system.	mountfsys(1M)
/mountall, umountall:	mount, unmount multiple file systems .	mountall(1M)
/rmountall, rumountall:	mount, unmount Remote File Sharing resources.	rmountall(1M)
systems .	mountall, umountall: mount, unmount multiple file	mountall(1M)
/rmntstat: display	mounted resource information.	rmntstat(1M)
file system.	mountfsys, umountfsys: mount, unmount a diskette	mountfsys(1M)
/rmount: retry remote resource	mounts.	rmount(1M)
/mvdir:	move a directory.	mvdir(1M)
lpshut, lpmove: start/stop the LP scheduler and	move requests. /lpsched,	lpsched(1M)
	mt: tape interface.	mt(7)
/mountall, umountall: mount, unmount	multiple file systems .	mountall(1M)
/rc2: run commands performed for	multi-user environment.	rc2(1M)
	mvdir: move a directory.	mvdir(1M)
/devnm: device	name.	devnm(1M)
/nsquery: Remote File Sharing	name server query.	nsquery(1M)
/dname: Print Remote File Sharing domain and network	names.	dname(1M)
/id: print user and group IDs and	names.	id(1M)
/ff: list file	names and statistics for a file system.	ff(1M)
/ncheck: generate path	names from i-numbers.	ncheck(1M)
	ncheck: generate path names from i-numbers.	ncheck(1M)
/nlsadmin:	network listener service administration.	nlsadmin(1M)
/dname: Print Remote File Sharing domain and	network names.	dname(1M)
	newgrp: log in to a new group.	newgrp(1M)
	nlsadmin: network listener service administration.	nlsadmin(1M)
/rfuadmin: Remote File Sharing	notification shell script.	rfuadmin(1M)
	nsquery: Remote File Sharing name server query.	nsquery(1M)
/null: the	null file.	null(7)
	null: the null file.	null(7)
/chargefee, ckpacct, dodisk, lastlogin, monacct,	nulladm, prctmp, prdaily, prtacct, runacct,/	acctsh(1M)
/df: report	number of free disk blocks and i-nodes.	df(1M)
/clone:	open any minor device on a STREAMS driver.	clone(7)
/rc0: run commands performed to stop the	operating system.	rc0(1M)
/prf:	operating system profiler.	prf(7)
/dcopy: copy file systems for	optimal access time.	dcopy(1M)
/sysdef:	output system definition.	sysdef(1M)
commands. /acctdisk, acctdusg, accton, acctwtmp:	overview of accounting and miscellaneous accounting	acct(1M)
/rfpasswd: change Remote File Sharing host	password.	rfpasswd(1M)
/pwck, grpck:	password/group file checkers.	pwck(1M)
/ncheck: generate	path names from i-numbers.	ncheck(1M)
/rc2: run commands	performed for multi-user environment.	rc2(1M)
/rc0: run commands	performed to stop the operating system.	rc0(1M)
/uucheck: check the uucp directories and	permissions file.	uucheck(1M)
/acctcms: command summary from	per-process accounting records.	acctcms(1M)
/hdeupdate: display/modify	physical disk configuration.	hdeupdate(1M)
/fmtflop:	physically format diskettes.	fmtflop(1M)
/powerdown: stop all processes and turn off the	power.	powerdown(1M)
power.	powerdown: stop all processes and turn off the	powerdown(1M)
/ckpacct, dodisk, lastlogin, monacct, nulladm,	prctmp, prdaily, prtacct, runacct, shutacct,/	acctsh(1M)
/dodisk, lastlogin, monacct, nulladm, prctmp,	prdaily, prtacct, runacct, shutacct, startup,/	acctsh(1M)
/accept, reject: allow or	prevent LP requests.	accept(1M)
	prf: operating system profiler.	prf(7)
/profiler: prfld, prfstat,	prfdc, prfsnap, prfpr: system profiler.	profiler(1M)
profiler. /profiler:	prfld, prfstat, prfdc, prfsnap, prfpr: system	profiler(1M)
/profiler: prfld, prfstat, prfdc, prfsnap,	prfpr: system profiler.	profiler(1M)
/profiler: prfld, prfstat, prfdc,	prfsnap, prfpr: system profiler.	profiler(1M)
profiler. /profiler: prfld,	prfstat, prfdc, prfsnap, prfpr: system	profiler(1M)

/infocmp: compare or	print out terminfo descriptions.	infocmp(1M)
/dname:	Print Remote File Sharing domain and network names.	dname(1M)
/strace:	print STREAMS trace messages.	strace(1M)
/id:	print user and group IDs and names.	id(1M)
/checkall: faster file system checking	procedure.	checkall(1M)
/brc, bcheckrc: system initialization	procedures.	brc(1M)
/intro: introduction to system maintenance	procedures.	intro(8)
runacct, shutacct, startup, turnacct: shell	procedures for accounting. /prdaily, prtacct,	acctsh(1M)
/rfudaemon: Remote File Sharing daemon	process.	rfudaemon(1M)
/acctprc1, acctprc2:	process accounting.	acctprc(1M)
/init, telinit:	process control initialization.	init(1M)
/killall: kill all active	processes.	killall(1M)
/powerdown: stop all	processes and turn off the power.	powerdown(1M)
/fuser: identify	processes using a file or file structure.	fuser(1M)
/fusage: disk access	profiler.	fusage(1M)
/prf: operating system	profiler.	prf(7)
prfld, prfstat, prfdc, prfsnap, prfpr: system	profiler. /profiler:	profiler(1M)
/sadp: disk access	profiler.	sadp(1M)
system profiler.	profiler: prfld, prfstat, prfdc, prfsnap, prfpr:	profiler(1M)
/strclean: STREAMS error logger cleanup	program.	strclean(1M)
/uusched: the scheduler for the uucp file transport	program.	uusched(1M)
/uucico: file transport	program for the uucp system.	uucico(1M)
to maintenance commands and application	programs. /intro: introduction	intro(1M)
/labelit:	provide labels for file systems.	labelit(1M)
/lastlogin, monacct, nulladm, prctmp, prdaily,	prtacct, runacct, shutacct, startup, turnacct:/	acctsh(1M)
/sxt:	pseudo-device driver.	sxt(7)
	pwck, grpck: password/group file checkers.	pwck(1M)
/nsquery: Remote File Sharing name server	query.	nsquery(1M)
system.	rc0: run commands performed to stop the operating	rc0(1M)
environment.	rc2: run commands performed for multi-user	rc2(1M)
/tirdwr: Transport Interface	read/write interface STREAMS module.	tirdwr(7)
command summary from per-process accounting	records. /acctcms:	acctcms(1M)
/fwtmp, wtmpfix: manipulate connect accounting	records.	fwtmp(1M)
/frec:	recover files from a backup tape.	frec(1M)
/accept,	reject: allow or prevent LP requests.	accept(1M)
code. /mk:	remake the binary system and commands from source	mk(8)
/adv: advertise a directory for	remote access.	adv(1M)
/uuxqt: execute	remote command requests.	uuxqt(1M)
/rfstart: start	Remote File Sharing.	rfstart(1M)
/rfudaemon:	Remote File Sharing daemon process.	rfudaemon(1M)
/rfadmin:	Remote File Sharing domain administration.	rfadmin(1M)
/dname: Print	Remote File Sharing domain and network names.	dname(1M)
/rfstop: stop the	Remote File Sharing environment.	rfstop(1M)
/rfpasswd: change	Remote File Sharing host password.	rfpasswd(1M)
/nsquery:	Remote File Sharing name server query.	nsquery(1M)
/rfuadmin:	Remote File Sharing notification shell script.	rfuadmin(1M)
/unadv: unadvertise a	Remote File Sharing resource.	unadv(1M)
/rmountall, rumountall: mount, unmount	Remote File Sharing resources.	rmountall(1M)
/idload:	Remote File Sharing user and group mapping.	idload(1M)
/rmount: retry	remote resource mounts.	rmount(1M)
/mount, umount: mount and unmount file systems and	remote resources.	mount(1M)
/Uutry: try to contact	remote system with debugging on.	Uutry(1M)
/checkfsys: check a file system on a	removable disk.	checkfsys(1M)
/fsck, dfsck: check and	repair file systems.	fsck(1M)
/hdelogger: Hard Disk Error status	report command and Log Daemon.	hdelogger(1M)
/fsstat:	report file system status.	fsstat(1M)
/df:	report number of free disk blocks and i-nodes.	df(1M)
/hdefix:	report or change bad block mapping.	hdefix(1M)
/sar: sa1, sa2, sadc: system activity	report package.	sar(1M)
/hdeadd: add/delete hdelog (Hard Disk Error Log)	reports.	hdeadd(1M)
/accept, reject: allow or prevent LP	requests.	accept(1M)
lpmove: start/stop the LP scheduler and move	requests. /lpsched, lpshut,	lpsched(1M)
/uuxqt: execute remote command	requests.	uuxqt(1M)
/fumount: forced unmount of an advertised	resource.	fumount(1M)
/unadv: unadvertise a Remote File Sharing	resource.	unadv(1M)
/rmntstat: display mounted	resource information.	rmntstat(1M)
/rmount: retry remote	resource mounts.	rmount(1M)
umount: mount and unmount file systems and remote	resources. /mount,	mount(1M)
rumountall: mount, unmount Remote File Sharing	resources. /rmountall,	rmountall(1M)
/s5restor: incremental file system	restore.	s5restor(1M)
/rmount:	retry remote resource mounts.	rmount(1M)
	rfadmin: Remote File Sharing domain administration.	rfadmin(1M)

	rfpasswd: change Remote File Sharing host password.	rfpasswd(1M)
	rfstart: start Remote File Sharing.	rfstart(1M)
	rfstop: stop the Remote File Sharing environment.	rfstop(1M)
script.	rfuadmin: Remote File Sharing notification shell	rfuadmin(1M)
	rfudaemon: Remote File Sharing daemon process.	rfudaemon(1M)
	rmntstat: display mounted resource information.	rmntstat(1M)
	rmount: retry remote resource mounts.	rmount(1M)
Sharing resources.	rmountall, rumountall: mount, unmount Remote File	rmountall(1M)
/chroot: change	root directory for a command.	chroot(1M)
resources. /rmountall,	rumountall: mount, unmount Remote File Sharing	rmountall(1M)
/rc2:	run commands performed for multi-user environment.	rc2(1M)
system. /rc0:	run commands performed to stop the operating	rc0(1M)
/runacct:	run daily accounting.	runacct(1M)
	runacct: run daily accounting.	runacct(1M)
/monacct, nulladm, prctmp, prdaily, prtacct,	runacct, shutacct, startup, turnacct: shell/	acctsh(1M)
	s5dump: incremental file system dump.	s5dump(1M)
	s5restor: incremental file system restore.	s5restor(1M)
	SA: devices administered by System Administration.	sa(7)
/sar:	sa1, sa2, sadc: system activity report package.	sar(1M)
/sar: sa1,	sa2, sadc: system activity report package.	sar(1M)
/sar: sa1, sa2,	sadc: system activity report package.	sar(1M)
	sadp: disk access profiler.	sadp(1M)
/sanityck: set/check file system	sanity flag.	sanityck(1M)
	sanityck: set/check file system sanity flag.	sanityck(1M)
package.	sar: sa1, sa2, sadc: system activity report	sar(1M)
/ckbupscd: check file system backup	schedule.	ckbupscd(1M)
/lpsched, lpshut, lpmove: start/stop the LP	scheduler and move requests.	lpsched(1M)
/uusched: the	scheduler for the uucp file transport program.	uusched(1M)
/rfuadmin: Remote File Sharing notification shell	script.	rfuadmin(1M)
/nsquery: Remote File Sharing name	server query.	nsquery(1M)
discipline. /getty:	set terminal type, modes, speed, and line	getty(1M)
discipline. /uugetty:	set terminal type, modes, speed, and line	uugetty(1M)
/sanityck:	set/check file system sanity flag.	sanityck(1M)
	setmnt: establish mount table.	setmnt(1M)
/rfstart: start Remote File	Sharing.	rfstart(1M)
/rfudaemon: Remote File	Sharing daemon process.	rfudaemon(1M)
/rfadmin: Remote File	Sharing domain administration.	rfadmin(1M)
/dname: Print Remote File	Sharing domain and network names.	dname(1M)
/rfstop: stop the Remote File	Sharing environment.	rfstop(1M)
/rfpasswd: change Remote File	Sharing host password.	rfpasswd(1M)
/nsquery: Remote File	Sharing name server query.	nsquery(1M)
/rfuadmin: Remote File	Sharing notification shell script.	rfuadmin(1M)
/unadv: unadvertise a Remote File	Sharing resource.	unadv(1M)
/rmountall, rumountall: mount, unmount Remote File	Sharing resources.	rmountall(1M)
/idload: Remote File	Sharing user and group mapping.	idload(1M)
prtacct, runacct, shutacct, startup, turnacct:	shell procedures for accounting. /prctmp, prdaily,	acctsh(1M)
/rfuadmin: Remote File Sharing notification	shell script.	rfuadmin(1M)
/shutdown:	shut down system, change system state.	shutdown(1M)
/nulladm, prctmp, prdaily, prtacct, runacct,	shutacct, startup, turnacct: shell procedures for/	acctsh(1M)
	shutdown: shut down system, change system state.	shutdown(1M)
/mk: remake the binary system and commands from	source code.	mk(8)
/getty: set terminal type, modes,	speed, and line discipline.	getty(1M)
/uugetty: set terminal type, modes,	speed, and line discipline.	uugetty(1M)
/uucleanup: uucp	spool directory clean-up.	uucleanup(1M)
/lpadmin: configure the LP	spooling system.	lpadmin(1M)
/rfstart:	start Remote File Sharing.	rfstart(1M)
/lpsched, lpshut, lpmove:	start/stop the LP scheduler and move requests.	lpsched(1M)
/prctmp, prdaily, prtacct, runacct, shutacct,	startup, turnacct: shell procedures for accounting.	acctsh(1M)
/ff: list file names and	statistics for a file system.	ff(1M)
/fsstat: report file system	status.	fsstat(1M)
/hdelogger: Hard Disk Error	status report command and Log Daemon.	hdelogger(1M)
/powerdown:	stop all processes and turn off the power.	powerdown(1M)
/rc0: run commands performed to	stop the operating system.	rc0(1M)
/rfstop:	stop the Remote File Sharing environment.	rfstop(1M)
	strace: print STREAMS trace messages.	strace(1M)
	strclean: STREAMS error logger cleanup program.	strclean(1M)
	streamio: STREAMS ioctl commands.	streamio(7)
/clone: open any minor device on a	STREAMS driver.	clone(7)
/strclean:	STREAMS error logger cleanup program.	strclean(1M)
/strerr:	STREAMS error logger daemon.	strerr(1M)
/log: interface to	STREAMS error logging and event tracing.	log(7)
/streamio:	STREAMS ioctl commands.	streamio(7)

/timod: Transport Interface cooperating	STREAMS module.	timod(7)
/tirdwr: Transport Interface read/write interface	STREAMS module.	tirdwr(7)
/strace: print	STREAMS trace messages.	strace(1M)
	strerr: STREAMS error logger daemon.	strerr(1M)
/fuser: identify processes using a file or file	structure.	fuser(1M)
	su: become super-user or another user.	su(1M)
/dsd: *Series 32000* computer Integral Disk	Subsystem.	dsd(7)
/du:	summarize disk usage.	du(1M)
/acctcms: command	summary from per-process accounting records.	acctcms(1M)
/sync: update the	super block.	sync(1M)
/su: become	super-user or another user.	su(1M)
/swap:	swap administrative interface.	swap(1M)
	swap: swap administrative interface.	swap(1M)
	sxt: pseudo-device driver.	sxt(7)
	sync: update the super block.	sync(1M)
	sysadm: menu interface to do system administration.	sysadm(1)
	sysdef: output system definition.	sysdef(1M)
or tape.	sysdump: dump system memory image to floppy disk(s)	sysdump(8)
/labelit: provide labels for file	systems.	labelit(1M)
/fsck, dfsck: check and repair file	systems.	fsck(1M)
/mountall, umountall: mount, unmount multiple file	systems.	mountall(1M)
/mount, umount: mount and unmount file	systems and remote resources.	mount(1M)
/dcopy: copy file	systems for optimal access time.	dcopy(1M)
/setmnt: establish mount	table.	setmnt(1M)
/frec: recover files from a backup	tape.	frec(1M)
dump system memory image to floppy disk(s) or	tape. /sysdump:	sysdump(8)
/mt:	tape interface.	mt(7)
/ldsysdump: load system dump from a	tape or floppy diskettes.	ldsysdump(1M)
/init,	telinit: process control initialization.	init(1M)
/captoinfo: convert a	termcap description into a terminfo description.	captoinfo(1M)
/termio: general	terminal interface.	termio(7)
/tty: controlling	terminal interface.	tty(7)
/getty: set	terminal type, modes, speed, and line discipline.	getty(1M)
/uugetty: set	terminal type, modes, speed, and line discipline.	uugetty(1M)
/tic:	terminfo compiler.	tic(1M)
/captoinfo: convert a termcap description into a	terminfo description.	captoinfo(1M)
/infocmp: compare or print out	terminfo descriptions.	infocmp(1M)
	termio: general terminal interface.	termio(7)
	tic: terminfo compiler.	tic(1M)
module.	timod: Transport Interface cooperating STREAMS	timod(7)
STREAMS module.	tirdwr: Transport Interface read/write interface	tirdwr(7)
/acctmerg: merge or add	total accounting files.	acctmerg(1M)
/strace: print STREAMS	trace messages.	strace(1M)
/log: interface to STREAMS error logging and event	tracing.	log(7)
/timod:	Transport Interface cooperating STREAMS module.	timod(7)
module. /tirdwr:	Transport Interface read/write interface STREAMS	tirdwr(7)
/uusched: the scheduler for the uucp file	transport program.	uusched(1M)
/uucico: file	transport program for the uucp system.	uucico(1M)
/Uutry:	try to contact remote system with debugging on.	Uutry(1M)
	tty: controlling terminal interface.	tty(7)
prdaily, prtacct, runacct, shutacct, startup,	turnacct: shell procedures for accounting. /prctmp,	acctsh(1M)
/getty: set terminal	type, modes, speed, and line discipline.	getty(1M)
/uugetty: set terminal	type, modes, speed, and line discipline.	uugetty(1M)
	uadmin: administrative control.	uadmin(1M)
resources. /mount,	umount: mount and unmount file systems and remote	mount(1M)
	umountall [—k].	umountall(1M)
/mountall,	umountall: mount, unmount multiple file systems.	mountall(1M)
/mountfsys,	umountfsys: mount, unmount a diskette file system.	mountfsys(1M)
	unadv: unadvertise a Remote File Sharing resource.	unadv(1M)
/unadv:	unadvertise a Remote File Sharing resource.	unadv(1M)
/link, unlink: link and	unlink files and directories.	link(1M)
/link,	unlink: link and unlink files and directories.	link(1M)
/mountfsys, umountfsys: mount,	unmount a diskette file system.	mountfsys(1M)
/mount, umount: mount and	unmount file systems and remote resources.	mount(1M)
/mountall, umountall: mount,	unmount multiple file systems.	mountall(1M)
/fumount: forced	unmount of an advertised resource.	fumount(1M)
/rmountall, rumountall: mount,	unmount Remote File Sharing resources.	rmountall(1M)
/sync:	update the super block.	sync(1M)
/du: summarize disk	usage.	du(1M)
/su: become super-user or another	user.	su(1M)
/id: print	user and group IDs and names.	id(1M)
/idload: Remote File Sharing	user and group mapping.	idload(1M)

/diskusg: generate disk accounting data by	user ID.	diskusg(1M)
/fuser: identify processes	using a file or file structure.	fuser(1M)
file.	uucheck: check the uucp directories and permissions	uucheck(1M)
	uucico: file transport program for the uucp system.	uucico(1M)
	uucleanup: uucp spool directory clean-up.	uucleanup(1M)
/uucheck: check the	uucp directories and permissions file.	uucheck(1M)
/uusched: the scheduler for the	uucp file transport program.	uusched(1M)
/uucleanup:	uucp spool directory clean-up.	uucleanup(1M)
/uucico: file transport program for the	uucp system.	uucico(1M)
discipline.	uugetty: set terminal type, modes, speed, and line	uugetty(1M)
program.	uusched: the scheduler for the uucp file transport	uusched(1M)
on.	Uutry: try to contact remote system with debugging	Uutry(1M)
	uuxqt: execute remote command requests.	uuxqt(1M)
	volcopy: make literal copy of file system.	volcopy(1M)
	whodo: who is doing what.	whodo(1M)
/fwtmp,	wtmpfix: manipulate connect accounting records.	fwtmp(1M)

NAME

intro — introduction to maintenance commands and application programs

DESCRIPTION

This section describes, in alphabetical order, commands that are used chiefly for system maintenance and administration purposes. The commands in this section should be used along with those listed in Section 1 of the *User's Reference Manual* and Sections 1, 2, 3, 4, and 5 of the *Programmer's Reference Manual.* References of the form *name*(1), (2), (3), (4) and (5) refer to entries in the above manuals. References of the form *name*(1M), *name*(7) or *name*(8) refer to entries in this manual.

COMMAND SYNTAX

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

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

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

SEE ALSO

getopt(1) in the *User's Reference Manual.*
getopt(3C) in the *Programmer's Reference Manual.*

DIAGNOSTICS

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

BUGS

Regrettably, not all commands adhere to the aforementioned syntax.

NAME

accept, reject — allow or prevent LP requests

SYNOPSIS

/usr/lib/accept destinations
/usr/lib/reject [—r[reason]] destinations

DESCRIPTION

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

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

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

FILES

/usr/spool/lp/*

SEE ALSO

lpadmin(1M), lpsched(1M).
enable(1), lp(1), lpstat(1) in the *User's Reference Manual.*

NAME

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

SYNOPSIS

/usr/lib/acct/acctdisk

/usr/lib/acct/acctdusg [**—u** file] [**—p** file]

/usr/lib/acct/accton [file]

/usr/lib/acct/acctwtmp "reason"

DESCRIPTION

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

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

Process accounting is performed by the GENIX V.3 system kernel. Upon termination of a process, one record per process is written to a file (normally **/usr/adm/pacct**). The programs in *acctprc*(1M) summarize this data for charging purposes; *acctcms*(1M) is used to summarize command usage. Current process data may be examined using *acctcom*(1).

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

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

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

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

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

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

FILES

/etc/passwd	used for login name to user ID conversions
/usr/lib/acct	holds all accounting commands listed in sub-class 1M of this manual
/usr/adm/pacct	current process accounting file
/etc/wtmp	login/logoff history file

SEE ALSO

acctcms(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), diskusg(1M), fwtmp(1M),

runacct(1M)
acctcom(1) in the *User's Reference Manual*
acct(2), acct(4), utmp(4) in the *Programmer's Reference Manual*

NAME

acctcms — command summary from per-process accounting records

SYNOPSIS

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

DESCRIPTION

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

—a	Print output in ASCII rather than in the internal summary format. The output includes command name, number of times executed, total kcore-minutes, total CPU minutes, total real minutes, mean size (in K), mean CPU minutes per invocation, "hog factor," characters transferred, and blocks read and written, as in *acctcom*(1). Output is normally sorted by total kcore-minutes.
—c	Sort by total CPU time, rather than total kcore-minutes.
—j	Combine all commands invoked only once under "***other".
—n	Sort by number of command invocations.
—s	Any file names encountered hereafter are already in internal summary format.
—t	Process all records as total accounting records. The default internal summary format splits each field into prime and non-prime time parts. This option combines the prime and non-prime time parts into a single field that is the total of both, and provides upward compatibility with old (*i.e.*, GENIX V.3 System) style **acctcms** internal summary format records.

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

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

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

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

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

SEE ALSO

acct(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M)
acctcom(1) in the *User's Reference Manual*
acct(2), acct(4), utmp(4) in the *Programmer's Reference Manual*

BUGS

Unpredictable output results if **—t** is used on new style internal summary format files, or if it is not used with old style internal summary format files.

NAME

acctcon1, acctcon2 — connect-time accounting

SYNOPSIS

/usr/lib/acct/acctcon1 [options]

/usr/lib/acct/acctcon2

DESCRIPTION

Acctcon1 converts a sequence of login/logoff records read from its standard input to a sequence of records, one per login session. Its input should normally be redirected from **/etc/wtmp**. Its output is ASCII, giving device, user ID, login name, prime connect time (seconds), non-prime connect time (seconds), session starting time (numeric), and starting date and time. The *options* are:

—p Print input only, showing line name, login name, and time (in both numeric and date/time formats).

—t *Acctcon1* maintains a list of lines on which users are logged in. When it reaches the end of its input, it emits a session record for each line that still appears to be active. It normally assumes that its input is a current file, so that it uses the current time as the ending time for each session still in progress. The **—t** flag causes it to use, instead, the last time found in its input, thus assuring reasonable and repeatable numbers for non-current files.

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

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

Acctcon2 expects as input a sequence of login session records and converts them into total accounting records (see **tacct** format in *acct*(4)).

EXAMPLES

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

```
acctcon1 —t —l lineuse —o reboots <wtmp | sort +1n +2 >ctmp
acctcon2 <ctmp | acctmerg >ctacct
```

FILES

/etc/wtmp

SEE ALSO

acct(1M), acctcms(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), init(1M), runacct(1M)
acctcom(1), login(1) in the *User's Reference Manual*
acct(2), acct(4), utmp(4) in the *Programmer's Reference Manual*

BUGS

The line usage report is confused by date changes. Use *wtmpfix* (see *fwtmp*(1M)) to correct this situation.

NAME

acctmerg — merge or add total accounting files

SYNOPSIS

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

DESCRIPTION

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

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

EXAMPLES

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

```
acctmerg —v <file1 >file2
        edit file2 as desired ...
acctmerg —i <file2 >file1
```

SEE ALSO

acct(1M), acctcms(1M), acctcon(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M)
acctcom(1) in the *User's Reference Manual*
acct(2), acct(4), utmp(4) in the *Programmer's Reference Manual*

NAME

acctprc1, acctprc2 — process accounting

SYNOPSIS

/usr/lib/acct/acctprc1 [ctmp]

/usr/lib/acct/acctprc2

DESCRIPTION

Acctprc1 reads input in the form described by *acct*(4), adds login names corresponding to user IDs, then writes for each process an ASCII line giving user ID, login name, prime CPU time (tics), non-prime CPU time (tics), and mean memory size (in memory segment units). If **ctmp** is given, it is expected to contain a list of login sessions, in the form described in *acctcon*(1M), sorted by user ID and login name. If this file is not supplied, it obtains login names from the password file. The information in **ctmp** helps it distinguish among different login names that share the same user ID.

Acctprc2 reads records in the form written by *acctprc1*, summarizes them by user ID and name, then writes the sorted summaries to the standard output as total accounting records.

These commands are typically used as shown below:

```
acctprc1 ctmp </usr/adm/pacct | acctprc2 >ptacct
```

FILES

/etc/passwd

SEE ALSO

acct(1M), acctcms(1M), acctcon(1M), acctmerg(1M), acctsh(1M), cron(1M), fwtmp(1M), runacct(1M)
acctcom(1) in the *User's Reference Manual*
acct(2), acct(4), utmp(4) in the *Programmer's Reference Manual*

BUGS

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

CAVEAT

A memory segment of the mean memory size is a unit of measure for the number of bytes in a logical memory segment on a particular processor. For example, on a PDP-11/70 this measure would be in 64-byte units, while on a VAX11/780 it would be in 512-byte units.

NAME

chargefee, ckpacct, dodisk, lastlogin, monacct, nulladm, prctmp, prdaily, prtacct, runacct, shutacct, startup, turnacct — shell procedures for accounting

SYNOPSIS

/usr/lib/acct/chargefee login-name number

/usr/lib/acct/ckpacct [blocks]

/usr/lib/acct/dodisk [-o] [files ...]

/usr/lib/acct/lastlogin

/usr/lib/acct/monacct number

/usr/lib/acct/nulladm file

/usr/lib/acct/prctmp

/usr/lib/acct/prdaily [-l] [-c] [mmdd]

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

/usr/lib/acct/runacct [mmdd] [mmdd state]

/usr/lib/acct/shutacct ["reason"]

/usr/lib/acct/startup

/usr/lib/acct/turnacct on | **off** | **switch**

/etc/init.d/acct

DESCRIPTION

Chargefee can be invoked to charge a *number* of units to *login-name*. A record is written to **/usr/adm/fee**, to be merged with other accounting records during the night.

Ckpacct should be initiated via *cron*(1M). It periodically checks the size of **/usr/adm/pacct**. If the size exceeds *blocks*, 1000 by default, *turnacct* will be invoked with argument *switch*. If the number of free disk blocks in the **/usr** file system falls below 500, *ckpacct* will automatically turn off the collection of process accounting records via the *off* argument to *turnacct*. When at least this number of blocks is restored, the accounting will be activated again. This feature is sensitive to the frequency at which *ckpacct* is executed, usually by *cron*.

Dodisk should be invoked by *cron* to perform the disk accounting functions. By default, it will do disk accounting on the special files in **/etc/fstab**. If the **—o** flag is used, it will do a slower version of disk accounting by login directory. *Files* specify the one or more filesystem names where disk accounting will be done. If *files* are used, disk accounting will be done on these filesystems only. If the **—o** flag is used, *files* should be mount points of mounted filesystem. If omitted, they should be the special file names of mountable filesystems.

Lastlogin is invoked by *runacct* to update **/usr/adm/acct/sum/loginlog**, which shows the last date on which each person logged in.

Monacct should be invoked once each month or each accounting period. *Number* indicates which month or period it is. If *number* is not given, it defaults to the current month (01—12). This default is useful if *monacct* is to executed via *cron*(1M) on the first day of each month. *Monacct* creates summary files in **/usr/adm/acct/fiscal** and restarts summary files in **/usr/adm/acct/sum**.

Nulladm creates *file* with mode 664 and ensures that owner and group are **adm**. It is called by various accounting shell procedures.

Prctmp can be used to print the session record file (normally **/usr/adm/acct/nite/ctmp** created by *acctcon*(1M).

Prdaily is invoked by *runacct* to format a report of the previous day's accounting data. The report resides in **/usr/adm/acct/sum/rprt***mmdd* where *mmdd* is the month and day of the report. The current daily accounting reports may be printed by typing *prdaily* . Previous days' accounting reports can be printed by using the *mmdd* option and specifying the exact report date desired. The —l flag prints a report of exceptional usage by login ID for the specifed date. Previous daily reports are cleaned up and therefore inaccessible after each invocation of *monacct* . The —*c* flag prints a report of exceptional resource usage by command, and may be used on current day's accounting data only.

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

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

Shutacct is invoked during a system shutdown to turn process accounting off and append a "reason" record to **/etc/wtmp.**

Startup is called by **/etc/init.d/acct** to turn the accounting on whenever the system is brought to a multi-user state.

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

FILES

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

SEE ALSO

acct(1M), acctcms(1M), acctcon(1M), acctmerg(1M), acctprc(1M), cron(1M), diskusg(1M), fwtmp(1M), runacct(1M)
acctcom(1) in the *User's Reference Manual*
acct(2), acct(4), utmp(4) in the *Programmer's Reference Manual*

NAME

adv — advertise a directory for remote access

SYNOPSIS

adv [-r] [-d *description*] *resource pathname* [*clients*...]
adv -m *resource* -d *description* | [*clients*...]
adv -m *resource* [-d *description*] | *clients*...
adv

DESCRIPTION

Adv is the Remote File Sharing command used to make a resource from one computer available for use on other computers. The machine that advertises the resource is called the *server*, while computers that mount and use the resource are *clients*. [See *mount*(1M).] (A resource represents a directory, which could contain files, subdirectories, named pipes and devices.)

There are three ways *adv* is used: 1) to advertise the directory *pathname* under the name *resource* so it is available to Remote File Sharing *clients*; 2) to modify *client* and *description* fields for currently advertised resources; or 3) to print a list of all locally-advertised resources.

The following options are available:

-r	Restricts access to the resource to a read-only basis. The default is read-write access.
-d *description*	Provides brief textual information about the advertised resource. *Description* is a single argument surrounded by double quotes (") and has a maximum length of 32 characters.
resource	This is the symbolic name used by the server and all authorized clients to identify the resource. It is limited to a maximum of 14 characters and must be different from every other resource name in the domain. All characters must be printable ASCII characters but must not include periods (.), slashes (/), or white space.
pathname	This is the local pathname of the advertised resource. It is limited to a maximum of 64 characters. This pathname cannot be the mount point of a remote resource and it can only be advertised under one resource name.
clients	These are the names of all clients that are authorized to remotely mount the resource. The default is that all machines that can connect to the server are authorized to access the resource. Valid input is of the form *nodename*, *domain.nodename*, *domain.*, or an alias that represents a list of client names. A domain name must be followed by a period (.) to distinguish it from a host name. The aliases are defined in **/etc/host.alias** and must conform to the alias capability in *mailx*(1).
-m	This option modifies information for a resource that has already been advertised. The resource is identified by a *resource* name. Only the *clients* and *description* fields can be modified. (To change the *pathname*, *resource* name, or read/write permissions, you must unadvertise and re-advertise the resource,)

When used with no options, *adv* displays all local resources that have been advertised; this includes the resource name, the pathname, the description, the read-write status, and the list of authorized clients. The resource field has a fixed length of 14 characters; all others are of variable length. Fields are separated by two white spaces, double quotes (") surround the description, and blank lines separate each resource entry.

This command may be used without options by any user; otherwise it is restricted to the superuser.

Remote File Sharing must be running before *adv* can be used to advertise or modify a resource entry.

EXIT STATUS

If there is at least one syntactically valid entry in the *clients* field, a warning will be issued for each invalid entry and the command will return a successful exit status. A non-zero exit status will be returned if the command fails.

ERRORS

If (1) the network is not up and running, (2) *pathname* is not a directory, (3) *pathname* isn't on a file system mounted locally, or (4) there is at least one entry in the *clients* field but none are syntactically valid, an error message will be sent to standard error.

FILES

/etc/host.alias

SEE ALSO

mount(1M), rfstart(1M), unadv(1M).
mailx(1) in the *User's Reference Manual.*

NAME

brc, bcheckrc — system initialization procedures

SYNOPSIS

/etc/brc

/etc/bcheckrc

DESCRIPTION

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

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

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

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

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

SEE ALSO

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

NAME

captoinfo — convert a termcap description into a terminfo description

SYNOPSIS

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

DESCRIPTION

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

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

—v	print out tracing information on standard error as the program runs. Specifying additional **—v** options will cause more detailed information to be printed.
—V	print out the version of the program in use on standard error and exit.
—1	cause the fields to print out one to a line. Otherwise, the fields will be printed several to a line to a maximum width of 60 characters.
—w	change the output to *width* characters.

FILES

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

CAVEATS

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

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

The short two-letter name at the beginning of the list of names in a *termcap* entry, a hold-over from an earlier version of the GENIX V.3 system, has been removed.

DIAGNOSTICS

tgetent failed with return code n (reason).
: The termcap entry is not valid. In particular, check for an invalid 'tc=' entry.

unknown type given for the termcap code *cc*.
: The termcap description had an entry for *cc* whose type was not boolean, numeric or string.

wrong type given for the boolean (numeric, string) termcap code *cc*.
: The boolean *termcap* entry *cc* was entered as a numeric or string capability.

the boolean (numeric, string) termcap code *cc* is not a valid name.
: An unknown *termcap* code was specified.

tgetent failed on TERM=term.
: The terminal type specified could not be found in the *termcap* file.

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

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

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

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

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

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

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

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

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

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

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

> *terminfo* assumes that the **sg** (now **xmc**) and **ug** values were the same.

Warning: the string produced for *ii* may be inefficient.

> The parameterized string being created should be rewritten by hand.

Null termname given.

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

cannot open *file* for reading.

> The specified file could not be opened.

SEE ALSO

infocmp(1M), tic(1M).
curses (3X), terminfo(4) in the *Programmer's Reference Manual.*
Chapter 10 in the *Programmer's Guide*.

NOTES

Captoinfo should be used to convert *termcap* entries to *terminfo*(4) entries because the *termcap* database (from earlier versions of GENIX V.3) may not be supplied in future releases.

NAME

checkall — faster file system checking procedure

SYNOPSIS

/etc/checkall

DESCRIPTION

The *checkall* procedure is a prototype and must be modified to suit local conditions. The following will serve as an example:

```
# check the root file system by itself
fsck /dev/dsk/c0d0s0

# dual fsck of (integral hard disk)
dfsck /dev/rdsk/c0d0s[123]
```

The *checkall* procedure takes 11 minutes.

Dfsck is a program that permits an operator to interact with two *fsck*(1M) programs at once. To aid in this, *dfsck* will print the file system name for each message to the operator. When answering a question from *dfsck*, the operator must prefix the response with a **1** or a **2** (indicating that the answer refers to the first or second file system group).

Due to the file system load balancing required for dual checking, the *dfsck*(1M) command should always be executed through the *checkall* shell procedure.

In a practical sense, the file systems are divided as follows:

```
dfsck file_systems_on_drive_0 — file_systems_on_drive_1
  . . .
```

WARNINGS

1. Do not use *dfsck* to check the *root* file system.
2. The *dfsck* procedure is useful only if the system is set up for multiple physical I/O buffers.

SEE ALSO

fsck(1M).

NAME

checkfsys — check a file system on a removable disk

SYNOPSIS

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

The user is asked one of the following three functions:

check the file system
: No repairs are attempted.

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

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

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

sysadm checkfsys

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

WARNING

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

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

SEE ALSO

fsck(1M), fsdb(1M), makefsys(1M), mountfsys(1M).
sysadm(1) in the *User's Reference Manual.*

NAME

chroot — change root directory for a command

SYNOPSIS

/etc/chroot newroot command

DESCRIPTION

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

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

```
chroot newroot command >x
```

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

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

This command can be run only by the superuser.

SEE ALSO

cd(1) in the *User's Reference Manual*.
chroot(2) in the *Programmer's Reference Manual*.

BUGS

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

NAME

ckbupscd — check file system backup schedule

SYNOPSIS

/etc/ckbupscd [—m]

DESCRIPTION

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

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

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

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

where:

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

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

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

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

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

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

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

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

EXAMPLES

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

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

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

FILES

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

SEE ALSO

cron(1M).
echo(1), sh(1), sysadm(1) in the *User's Reference Manual.*

BUGS

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

NAME

clri — clear i-node

SYNOPSIS

/etc/clri special i-number ...

DESCRIPTION

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

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

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

SEE ALSO

fsck(1M), fsdb(1M), ncheck(1M).
fs(4) in the *Programmer's Reference Manual.*
rm(1) in the *User's Reference Manual.*

WARNINGS

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

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

NAME

config - configure a GENIX V System

SYNOPSIS

/etc/config [-t] [-l file] [-c file] [-m file] [-h file] dfile1 dfile2, dfile3, ...

DESCRIPTION

Config is a program that takes a description of a GENIX V system and generates three files. One file provides information regarding the interface between the hardware and device handlers. One file is a C program defining the configuration tables for the various devices on the system. The other file is a configuration definition file to be included in the C program.

The -l option specifies the name of the hardware interface file; low.s is the default name.

The -c option specifies the name of the configuration table file; conf.c is the default name.

The -m option specifies the name of the file that contains all the information regarding supported devices; /etc/master is the default name. This file is supplied with the GENIX V system and should not be modified unless the user fully understands its construction.

The -h option specifies the name of the configuration header file; config.h is the default name.

The -t option requests a short table of major device numbers for character and block type devices. This can facilitate the creation of special files.

The user must supply all dfiles for configuring the kernel; the dfiles contain device information for the user's system, one for each kind of device. All dfiles are of the same format. Each dfile is divided into two parts separated by a line with a dollar sign($) in column 1. The first part contains physical device specifications. The second part contains system-dependent information. Any line with an asterisk (*) in column 1 is a comment.

All configurations are assumed to have the following devices:

con	- console driver
memory	- memory driver
tty	- controlling terminal interface
errlog	- error logger

with standard interrupt vectors and addresses. These devices must not be specified in dfile.

First Part of dfile

Each line contains five or six fields, delimited by blanks and/or tabs in the following format:

```
devname   vector   address   ioaddr    bus   number
```

where devname is the name of the device (as it appears in the /etc/master device table), vector is the interrupt vector location (decimal), address is the buffer memory address allocated for device driver, ioaddr is the device address (I/O port address, hex), bus is the bus request level (4 through 7), and number is the number (decimal) of devices associated with the corresponding controller; number is optional, and if omitted, a default value which is the maximum value for that controller is used.

There are certain drivers that may be provided with the system, that are actually pseudo-device drivers; that is, there is no real hardware associated with the driver. Drivers of this type are identified on their respective manual entries. When these devices are specified in the description file, the interrupt vector, device address, ioaddr and bus request level must all be zero.

Second Part of dfile

The second part of the file initializes configurable variables, structures, and arrays via a syntax identical to the C language. The structures may be defined in header files and included in the dfile via the keyword &INCLUDE at the beginning of the line.

Parameters may be defined via the keyword &DEFINE at the beginning of a line.

The dfile named kernel is a special file, it initializes many of the kernel's structures and variables. Among them are:

rootdev	- The major/minor device number for the root file system.
swapdev	- The major/minor device number for the swap device.
swplo	- First swap block starts at this offset within the swap device.
nswap	- Number of blocks in the swap device.
pipedev	- The major/minor device number for the pipe device.
dumpdev	- The major/minor device number for the dump device.
init_tbl	- Table of initialization routines not specific to any driver or function.
v	- Table of sizes for certain arrays whose size grows dynamically to a maximum.
utsname	- Names for the current running system.
linesw	- The line disciple switch table.
tune	- Table of values used to fine tune the kernel's performance.

A sample from a part of the kernel file follows.

```
                     int   rootdev = makedev(0,0);
                     int   swapdev = makedev(0,1);
                     int   swplo = 0;
                     int   nswap = 8976;
                     int   pipedev = makedev(0,0);

&INCLUDE                      <sys/file.h>
&DEFINE                       NFILE                        100
struct                        file                         file[NFILE];
```

NAME

crash — examine system images

SYNOPSIS

/etc/crash [**—d** dumpfile] [—n namelist] [—w outputfile]

DESCRIPTION

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

Dumpfile is the file containing the system memory image. The default *dumpfile* is */dev/mem.* The system image can also be the pathname of a file produced by *ldsysdump*(1M).

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

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

Input during a *crash* session is of the form:

function [argument ...]

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

The default for process-related items is the current process for a running system and the process that was running at the time of the crash for a crashed system. If the contents of a table are being dumped, the default is all active table entries.

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

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

Note that if the **—p** option is used, all address and symbol arguments explicitly entered on the command line will be interpreted as physical addresses. If they are not physical addresses, results will be inconsistent.

The functions *mode, defproc,* and *redirect* correspond to the function options **—p, —s,** and **—w.** The *mode* function may be used to set the address translation mode to physical or virtual for all subsequently entered functions; *defproc* sets the value of the process slot argument for subsequent functions; and *redirect* redirects all subsequent output.

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

function [argument ...] ! shell_command

For example,

mount ! grep rw

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

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

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

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

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

table_entry = table entry I address I symbol I range I expression

start_addr = address I symbol I expression

FUNCTIONS

? [**—w** file] List available functions.

!cmd Escape to the shell to execute a command.

!! Repeat previous shell command.

adv [**—e**] [**—w** file] [[**—p**] table_entry ...]
Print the advertise table.

base [**—w** file] number ...
Print *number* in binary, octal, decimal, and hexadecimal. A number in a radix other then decimal should be preceded by a prefix that indicates its radix as follows: *0x*, hexidecimal; *0*, octal; and *0b*, binary.

buffer [**—w** file] [—format] bufferslot

or

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

bufhdr [**—f**] [**—w** file] [[**—p**] table_entry ...]
Alias: **buf.**
Print system buffer headers.

callout [**—w** file]
Alias: **c.**
Print the callout table.

dballoc [**—w** file] [class ...]
Print the dballoc table. If a class is entered, only data block allocation information for that class will be printed.

dbfree [**—w** file] [class ...]
Print free streams data block headers. If a class is entered, only data block headers for the class specified will be printed.

dblock [**—e**] [**—w** file] [**—c** class ...]

or

dblock [**—e**] [**—w** file] [[**—p**] table_entry ...]
Print allocated streams data block headers. If the class option (**—c**) is used, only data block headers for the class specified will be printed.

defproc [**—w** file] [**—c**]

or

defproc [**—w** file] [slot]
Set the value of the process slot argument. The process slot argument may be set to the current slot number (**—c**) or the slot number may be specified. If no argument is entered, the value of the previously set slot number is printed. At the start of a *crash* session, the process slot is set to the current process.

dis [**—w** file] [**—a**] start_addr [count]
Disassemble from the start address for *count* instructions. The default count is 1. The absolute option (**—a**) specifies a non-symbolic disassembly.

ds [**—w** file] virtual_address ...
Print the data symbol whose address is closest to, but not greater than, the address entered.

file [**—e**] [**—w** file] [[**—p**] table_entry ...]
Alias: **f**.
Print the file table.

findaddr [**—w** file] table slot
Print the address of *slot* in *table*. Only tables available to the *size* function are available to *findaddr*.

findslot [**—w** file] virtual_address ...
Print the table, entry slot number, and offset for the address entered. Only tables available to the *size* function are available to *findslot*.

fs [**—w** file] [[**—p**] table_entry ...]
Print the file system information table.

gdp [**—e**] [**—f**] [**—w** file] [[**—p**] table_entry ...]
Print the gift descriptor protocol table.

help [**—w** file] function ...
Print a description of the named function, including syntax and aliases.

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

kfp [**—w** file] [**—s** process] [**—r**]

or

kfp [**—w** file] [**—s** process] [value]
Print the frame pointer for the start of a kernel stack trace. The kfp value can be set

using the value argument or the reset option (—**r**), which sets the kfp through the nvram. If no argument is entered, the current value of the kfp is printed.

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

linkblk [—**e**] [—**w** file] [[—**p**] table_entry ...]
Print the linkblk table.

major [—**w** file] [entry ...]
Print the MAJOR table.

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

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

mblock [—**e**] [—**w** filename] [[—**p**] table_entry ...]
Print allocated streams message block headers.

mmu [—**w** file]
Alias: **regs.**
Print memory management unit registers. These registers are not available on a running system.

mode [—**w** file] [mode]
Set address translation of arguments to virtual (**v**) or physical (**p**) mode. If no mode argument is given, the current mode is printed. At the start of a *crash* session, the mode is virtual.

mount [—**e**] [—**w** file] [[—**p**] table_entry ...]
Alias: **m.**
Print the mount table.

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

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

pdt [—**e**] [—**w** file] [—**s** process] section segment

or

pdt [—**e**] [—**w** file] [—**s** process] [—**p**] start_addr [count]
The page descriptor table of the designated memory *section* and *segment* is printed. Alternatively, the page descriptor table starting at the start address for *count* entries is printed. If no count is entered, 1 is assumed.

pfdat [**—e**] [**—w** file] [[**—p**] table_entry ...]
Print the pfdata table.

proc [**—f**] [**—w** file] [[**—p**] table_entry ... #procid ...]

or

proc [**—f**] [**—w** file] [**—r**]
Alias: **p**.
Print the process table. Process table information may be specified in two ways. First, any mixture of table entries and process ids may be entered. Each process id must be preceded by a #. Alternatively, process table information for runnable processes may be specified with the runnable option (**—r**).

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

queue [**—e**] [**—w** file] [[**—p**] table_entry ...]
Print streams queues.

quit Alias: **q**.
Terminate the *crash* session.

r Repeat last command.

rcvd [**—e**] [**—f**] [**—w** file] [[**—p**] table_entry ...]
Print the receive descriptor table.

redirect [**—w** file] [**—c**]

or

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

region [**—e**] [**—f**] [**—w** file] [[**—p**] table_entry ...]
Print the region table.

sdt [**—e**] [**—w** file] [**—s** process] section

or

sdt [**—e**] [**—w** file] [**—s** process] [**—p**] start_addr [count]
The segment descriptor table for the named memory section is printed. Alternatively, the segment descriptor table starting at start address for *count* entries is printed. If no count is given, a count of 1 is assumed.

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

size [**—w** file] [**—x**] [structure_name ...]
Print the size of the designated structure. The (**—x**) option prints the size in hexadecimal. If no argument is given, a list of the structure names for which sizes are available is printed.

sndd [**—e**] [**—w** file] [[**—p**] table_entry ...]
Print the send descriptor table.

srmount [**—e**] [**—w** file] [[**—p**] table_entry ...]
Print the server mount table.

stack [**—w** file] [**—u**] [process]

or

stack [**—w** file] [**—k**] [process]

or

stack [**—w** file] [[**—p**] **—i** start_addr]

Alias: **s**.

Dump stack. The (**—u**) option prints the user stack. The (**—k**) option prints the kernel stack. The (**—i**) option prints the interrupt stack starting at the start address. If no arguments are entered, the kernel stack for the current process is printed. The interrupt stack and the stack for the current process are not available on a running system.

stat [**—w** file]

Print system statistics.

stream [**—e**] [**—f**] [**—w** file] [[**—p**] table_entry ...]

Print the streams table.

strstat [**—w** file]

Print streams statistics.

trace [**—w** file] [**—r**] [process]

or

trace [**—w** file] [[**—p**] **—i** start_addr]

Alias: **t**.

Print stack trace. The kfp value is used with the **—r** option. The interrupt option prints a trace of the interrupt stack beginning at the start address. The interrupt stack trace and the stack trace for the current process are not available on a running system.

ts [**—w** file] virtual_address ...

Print closest text symbol to the designated address.

tty [**—e**] [**—f**] [**—w** file] [**—t** type [[**—p**] table_entry ...]]

or

tty [**—e**] [**—f**] [**—w** file] [[**—p**] start_addr]

Valid types: **iu**.

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

user [**—f**] [**—w** file] [process]

Alias: **u**.

Print the ublock for the designated process.

var [**—w** file]

Alias: **v**.

Print the tunable system parameters.

vtop [**—w** file] [**—s** process] start_addr ...

Print the physical address translation of the virtual start address.

FILES

/dev/mem	system image of currently running system

SEE ALSO

ldsysdump(1M), sysdump(8).

NAME

cron — clock daemon

SYNOPSIS

/etc/cron

DESCRIPTION

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

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

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

FILES

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

SEE ALSO

at(1), crontab(1), sh(1) in the *User's Reference Manual.*

DIAGNOSTICS

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

NAME

dcopy — copy file systems for optimal access time

SYNOPSIS

/etc/dcopy [—sX] [—an] [—d] [—v] [—ffsize[:isize]] inputfs outputfs

DESCRIPTION

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

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

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

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

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

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

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

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

SEE ALSO

fsck(1M), mkfs(1M).
ps(1) in the *User's Reference Manual.*

NAME

dd — convert and copy a file

SYNOPSIS

dd [option=value] ...

DESCRIPTION

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

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

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

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

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

DIAGNOSTICS

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

NAME

devnm — device name

SYNOPSIS

/etc/devnm [names]

DESCRIPTION

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

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

EXAMPLE

The command:

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

produces

```
        /dev/dsk/c0d0s2 usr
```

if **/usr** is mounted on **/dev/dsk/c0d0s2.**

FILES

/dev/dsk/*
/etc/mnttab

SEE ALSO

brc(1M).

NAME

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

SYNOPSIS

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

DESCRIPTION

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

File-system may be specified either by device name (*e.g.*, **/dev/dsk/c0d0s2**) or by mount point directory name (*e.g.*, **/usr**).

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

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

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

The *df* command uses the following options:

—l only reports on local file systems.

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

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

NOTE

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

FILES

/dev/dsk/*
/etc/mnttab

SEE ALSO

mount(1M).
fs(4), mnttab(4) in the *Programmer's Reference Manual.*

NAME

diskusg — generate disk accounting data by user ID

SYNOPSIS

diskusg [options] [files]

DESCRIPTION

Diskusg generates intermediate disk accounting information from data in *files*, or the standard input if omitted. *Diskusg* output lines on the standard output, one per user, in the following format: *uid login #blocks*

where

uid -	the numerical user ID of the user.
login -	the login name of the user; and
#blocks -	the total number of disk blocks allocated to this user.

Diskusg normally reads only the i-nodes of file systems for disk accounting. In this case, *files* are the special filenames of these devices.

Diskusg recognizes the following options:

-s	the input data is already in *diskusg* output format. *Diskusg* combines all lines for a single user into a single line.
-v	verbose. Print a list on standard error of all files that are charged to no one.
-i *fnmlist*	ignore the data on those file systems whose file system name is in *fnmlist*. *Fnmlist* is a list of file system names separated by commas or enclosed within quotes. *diskusg* compares each name in this list with the file system name stored in the volume ID (see *labelit(1M)*).
-p *file*	use *file* as the name of the password file to generate login names. **/etc/passwd** is used by default.
-u *file*	write records to *file* of files that are charged to no one. Records consist of the special file name, the i-node number, and the user ID.

The output of *diskusg* is normally the input to *acctdisk* (see *acct*(1M)) which generates total accounting records that can be merged with other accounting records. *Diskusg* is normally run in *dodisk* (see *acctsh*(1M)).

EXAMPLES

The following will generate daily disk accounting information:

```
for i in /dev/rp00 /dev/rp01 /dev/rp10 /dev/rp11; do
        diskusg $i > dtmp.`basename $i` &
done
wait
diskusg -s dtmp.* | sort +0n +1 | acctdisk > disktacct
```

FILES

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

SEE ALSO

acct(1M), acctsh(1M)
acct(4) in the *Programmer's Reference Manual*

NAME

dname — Print Remote File Sharing domain and network names

SYNOPSIS

dname [—D *domain*] [—N *netspec*] [—dna]

DESCRIPTION

Dname prints or defines a host's Remote File Sharing domain name or the network used by Remote File Sharing as transport provider. When used with **d**, **n**, or **a** options, *dname* can be run by any user to print the domain name, network name or both, respectively. Only a user with root permission can use the **—D** *domain* option to set the domain name for the host or **—N** *netspec* to set the network specification used for Remote File Sharing. (The value of *netspec* is the network device name, relative to the */dev* directory.

Domain must consist of no more than 14 characters, consisting of any combination of letters (upper and lower case), digits, hyphens (—), and underscores (_)

When *dname* is used to change a domain name, the host's password is removed. The administrator will be prompted for a new password the next time Remote File Sharing is started [*rfstart*(1M)].

If *dname* is used with no options, it will default to *dname —d.*

ERRORS

You cannot use the **—N** or **—D** options while Remote File Sharing is running.

SEE ALSO

rfstart(1M).

NAME

du — summarize disk usage

SYNOPSIS

du [**—sar**] [names]

DESCRIPTION

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

The optional arguments are as follows:

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

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

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

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

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

BUGS

If the **—a** option is not used, non-directories given as arguments are not listed. If there are links between files in different directories where the directories are on separate branches of the file system hierarchy, *du* will count the excess files more than once. Files with holes in them will get an incorrect block count. (See Chapter 5, File System Administration, in the *Administrator's Guide*)

NAME

errdump — print error log

SYNOPSIS

/etc/errdump

DESCRIPTION

This command displays on the system console the error log contained in the system's nonvolatile ram. The display contains the previous saved system state, the last 5 panic messages and their time of occurrence, and an indication of the log's sanity.

DIAGNOSTICS

The phrase "not superuser" is displayed, if the command is invoked by other than the super-user. Super-user is defined as anyone logged in under the root directory from the console port.

EXAMPLE

The following is an example of the printout in response to the *errdump* command.

```
#
#
#
#
# errdump
nvram status:  sane

csr:      0x0648  (floppy)  (unassigned)  (clock)  (uart)

psw:      rsvd CSH_F_D QIE CSH_D OE NZVC TE IPL CM PM R I ISC TM FT
            0      1  0    1  0   0  0  f  0  0 1 0  5  0 3

r3:       0x00049001
r4:       0x00000081
r5:       0x00000000
r6:       0x40091348
r7:       0x0001a13f
r8:       0x4008edd8
oap:      0x400816d8
opc:      0x400083bc
osp:      0x40081700
ofp:      0x40081700
isp:      0x40080008
pcbp:     0x40041a40

fltar:    0xc0021140
fltcr:    reqacc  xlevel  ftype
          0xa     0x0     0x0
```

```
             srama            sramb
     [0]     0x02034800       0x0000011f
     [1]     0x02035100       0x00000030
     [2]     0x02035860       0x00000074
     [3]     0x02035c00       0x00000015
             Panic log

     [0]     Thu Sep 20 09:51:36 1984
             KERNEL DATA ALIGNMENT ERROR

     [1]     Thu Sep 20 09:51:37 1984
             KERNEL DATA ALIGNMENT ERROR

     [2]     Thu Sep 20 09:51:40 1984
             KERNEL DATA ALIGNMENT ERROR

     [3]     Thu Sep 20 09:52:21 1984
             KERNEL DATA ALIGNMENT ERROR

     [4]     Fri Sep 21 05:50:10 1984
             SYSTEM PARITY ERROR INTERRUPT
```

SEE ALSO

Appendix C: Error Messages in the *System Administrator's Guide.*

NAME

ff — list file names and statistics for a file system

SYNOPSIS

/etc/ff [options] special

DESCRIPTION

Ff reads the i-list and directories of the *special* file, assuming it is a file system. I-node data is saved for files which match the selection criteria. Output consists of the path name for each saved i-node, plus other file information requested using the print *options* below. Output fields are positional. The output is produced in i-node order; fields are separated by tabs. The default line produced by *ff* is:

```
path-name i-number
```

With all *options* enabled, output fields would be:

```
path-name i-number size uid
```

The argument *n* in the *option* descriptions that follow is used as a decimal integer (optionally signed), where +*n* means more than *n*, —*n* means less than *n*, and *n* means exactly *n*. A day is defined as a 24 hour period.

—I	Do not print the i-node number after each path name.
—l	Generate a supplementary list of all path names for multiply-linked files.
—p *prefix*	The specified *prefix* will be added to each generated path name. The default is . (dot).
—s	Print the file size, in bytes, after each path name.
—u	Print the owner's login name after each path name.
—a *n*	Select if the i-node has been accessed in *n* days.
—m *n*	Select if the i-node has been modified in *n* days.
—c *n*	Select if the i-node has been changed in *n* days.
—n *file*	Select if the i-node has been modified more recently than the argument *file*.
—i *i-node-list*	Generate names for only those i-nodes specified in *i-node-list*.

SEE ALSO

ncheck(1M).
find(1) in the *User's Reference Manual.*

BUGS

If the —l option is not specified, only a single path name out of all possible ones is generated for a multiply-linked i-node. If —l is specified, all possible names for every linked file on the file system are included in the output. However, no selection criteria apply to the names generated.

NAME

finc — fast incremental backup

SYNOPSIS

/etc/**finc** [selection-criteria] file-system raw-tape

DESCRIPTION

Finc selectively copies the input *file-system* to the output *raw-tape* . The cautious will want to mount the input *file-system* read-only to insure an accurate backup, although acceptable results can be obtained in read-write mode. The tape must be previously labelled by *labelit.* The selection is controlled by the *selection-criteria*, accepting only those inodes/files for whom the conditions are true.

It is recommended that production of a *finc* tape be preceded by the *ff* command, and the output of *ff* be saved as an index of the tape's contents. Files on a *finc* tape may be recovered with the *frec* command.

The argument *n* in the *selection-criteria* which follow is used as a decimal integer (optionally signed), where +*n* means more than *n*, —*n* means less than *n*, and *n* means exactly *n*. A day is defined as a 24 hours.

—a *n*	True if the file has been accessed in *n* days.
—m *n*	True if the file has been modified in *n* days.
—c *n*	True if the i-node has been changed in *n* days.
—n *file*	True for any file which has been modified more recently than the argument *file*.

EXAMPLES

To write a tape consisting of all files from file-system **/usr** modified in the last 48 hours:

```
finc —m —2 /dev/rdsk/c0d0s2 /dev/rmt/0m
```

SEE ALSO

ff(1M), frec(1M), labelit(1M).
cpio(1) in the *User's Reference Manual.*

NAME

fmtflop — physically format diskettes

SYNOPSIS

/etc/fmtflop *special_file*

DESCRIPTION

Fmtflop physically formats the media inserted in the diskette drive. The *special_file* is the path name of the diskette drive (*e.g.*, **/dev/rdiskette**).

Fmtflop formats DOUBLE SIDED media with 512 byte sectors, 8 sectors per track, and 80 tracks. Before executing *fmtflop*, the diskette must be placed in the drive and the latch closed.

SEE ALSO

dsd(7).

DIAGNOSTICS

An error message is returned if the format fails. If this occurs, remove the diskette and reinsert it to make sure it is properly seated, then try entering the command a second time. If the command fails again (especially on the same area of the disk) the diskette is probably bad and must be discarded.

NAME

frec — recover files from a backup tape

SYNOPSIS

/etc/frec [**—p** path] [**—f** reqfile] raw_tape i_number:name ...

DESCRIPTION

Frec recovers files from the specified *raw_tape* backup tape written by *volcopy*(1M) or *finc*(1M), given their *i_numbers*. The data for each recovery request will be written into the file given by *name* .

The **—p** option allows you to specify a default prefixing *path* different from your current working directory. This will be prefixed to any *names* that are not fully qualified, *i.e.*, that do not begin with / or ./. If any directories are missing in the paths of recovery *names* they will be created.

—p *path*	Specifies a prefixing *path* to be used to fully qualify any names that do not start with / or ./.
—f *reqfile*	Specifies a file which contains recovery requests. The format is i_number:newname, one per line.

EXAMPLES

To recover a file, i-number 1216 when backed-up, into a file named *junk* in your current working directory:

```
frec /dev/rmt/0m 1216:junk
```

To recover files with i_numbers 14156, 1232, and 3141 into files **/usr/src/cmd/a**, **/usr/src/cmd/b** and **/usr/joe/a.c**:

```
frec —p /usr/src/cmd /dev/rmt/0m 14156:a 1232:b
  3141:/usr/joe/a.c
```

SEE ALSO

ff(1M), finc(1M), labelit(1M).
cpio(1) in the *User's Reference Manual.*

BUGS

While paving a path (*i.e.*, creating the intermediate directories contained in a pathname), *frec* can only recover inode fields for those directories contained on the tape and requested for recovery.

NAME

fsck, dfsck — check and repair file systems

SYNOPSIS

/etc/fsck [**—y**] [**—n**] [—sX] [—SX] [**—t** file] [**—q**] [—D] [**—f**] [**—b**] [file-systems]
/etc/dfsck [options1] fsys1 ... — [options2] fsys2 ...

DESCRIPTION

Fsck

Fsck audits and interactively repairs inconsistent conditions for file systems. If the file system is found to be consistent, the number of files, blocks used, and blocks free are reported. If the file system is inconsistent the user is prompted for concurrence before each correction is attempted. It should be noted that most corrective actions will result in some loss of data. The amount and severity of data loss may be determined from the diagnostic output. The default action for each correction is to wait for the user to respond **yes** or **no**. If the user does not have write permission *fsck* defaults to a **—n** action.

The following options are accepted by *fsck*.

—y Assume a **yes** response to all questions asked by *fsck*.

—n Assume a **no** response to all questions asked by *fsck*; do not open the file system for writing.

—s*X* Ignore the actual free list and (unconditionally) reconstruct a new one by rewriting the super-block of the file system. The file system should be unmounted while this is done; if this is not possible, care should be taken that the system is quiescent and that it is rebooted immediately afterwards. This precaution is necessary so that the old, bad, in-core copy of the superblock will not continue to be used, or written on the file system.

The **—s***X* option allows for creating an optimal free-list organization.

If *X* is not given, the values used when the file system was created are used. The format of *X* is *cylinder size:gap size.*

—S*X* Conditionally reconstruct the free list. This option is like **—s***X* above except that the free list is rebuilt only if there were no discrepancies discovered in the file system. Using **—S** will force a **no** response to all questions asked by *fsck*. This option is useful for forcing free list reorganization on uncontaminated file systems.

—t If *fsck* cannot obtain enough memory to keep its tables, it uses a scratch file. If the **—t** option is specified, the file named in the next argument is used as the scratch file, if needed. Without the **—t** flag, *fsck* will prompt the user for the name of the scratch file. The file chosen should not be on the file system being checked, and if it is not a special file or did not already exist, it is removed when *fsck* completes.

—q Quiet *fsck*. Do not print size-check messages. Unreferenced **fifos** will silently be removed. If *fsck* requires it, counts in the superblock will be automatically fixed and the free list salvaged.

—D Directories are checked for bad blocks. Useful after system crashes.

—f Fast check. Check block and sizes and check the free list. The free list will be reconstructed if it is necessary.

—b Reboot. If the file system being checked is the root file system and modifications have been made, then either remount the root file system or reboot the system. A remount is done only if there was minor damage.

If no *file-systems* are specified, *fsck* will read a list of default file systems from the file **/etc/checklist**.

Inconsistencies checked are as follows:

1. Blocks claimed by more than one i-node or the free list.
2. Blocks claimed by an i-node or the free list outside the range of the file system.
3. Incorrect link counts.
4. Size checks:
   Incorrect number of blocks.
   Directory size not 16-byte aligned.
5. Bad i-node format.
6. Blocks not accounted for anywhere.
7. Directory checks:
   File pointing to unallocated i-node.
   I-node number out of range.
8. Super Block checks:
   More than 65536 i-nodes.
   More blocks for i-nodes than there are in the file system.
9. Bad free block list format.
10. Total free block and/or free i-node count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with the user's concurrence, reconnected by placing them in the **lost+found** directory, if the files are nonempty. The user will be notified if the file or directory is empty or not. Empty files or directories are removed, as long as the **—n** option is not specified. *Fsck* will force the reconnection of nonempty directories. The name assigned is the i-node number. The only restriction is that the directory **lost+found** must preexist in the root of the file system being checked and must have empty slots in which entries can be made. This is accomplished by making **lost+found**, copying a number of files to the directory, and then removing them (before fsck is executed).

Checking the raw device is almost always faster and should be used with everything but the *root* file system.

Dfsck

This version of the *fsck* command is appropriate for *Series 32000* computers only if equipped with dual hard disk drives. *Dfsck* should not be used to check the *root* file system.

Dfsck allows two file system checks on two different drives simultaneously. *Options1* and *options2* are used to pass options to *fsck* for the two sets of file systems. A — is the separator between the file system groups.

The *dfsck* program permits a user to interact with two *fsck* programs at once. To aid in this, *dfsck* will print the file system name for each message to the user. When answering a question from *dfsck*, the user must prefix the response with a **1** or a **2** (indicating that the answer refers to the first or second file system group).

FILES

/etc/checklist	contains default list of file systems to check.

SEE ALSO

checkfsys(1M),mkfs(1M), ncheck(1M), crash(1M).
uadmin(2), checklist(4), fs(4) in the *Programmer's Reference Manual.*

BUGS

I-node numbers for **.** and **..** in each directory are not checked for validity.

NAME

fsdb — file system debugger

SYNOPSIS

/etc/fsdb special [—]

DESCRIPTION

Fsdb can be used to patch up a damaged file system after a crash. It has conversions to translate block and i-numbers into their corresponding disk addresses. Also included are mnemonic offsets to access different parts of an i-node. These greatly simplify the process of correcting control block entries or descending the file system tree.

Fsdb contains several error-checking routines to verify i-node and block addresses. These can be disabled if necessary by invoking *fsdb* with the optional — argument or by the use of the **O** symbol. (*Fsdb* reads the i-size and f-size entries from the superblock of the file system as the basis for these checks.)

Numbers are considered decimal by default. Octal numbers must be prefixed with a zero. During any assignment operation, numbers are checked for a possible truncation error due to a size mismatch between source and destination.

Fsdb reads a block at a time and will therefore work with raw as well as block I/O. A buffer management routine is used to retain commonly used blocks of data in order to reduce the number of read system calls. All assignment operations result in an immediate write-through of the corresponding block.

The symbols recognized by *fsdb* are:

#	absolute address
i	convert from i-number to i-node address
b	convert to block address
d	directory slot offset
+,—	address arithmetic
q	quit
>,<	save, restore an address
=	numerical assignment
=+	incremental assignment
=—	decremental assignment
="	character string assignment
O	error checking flip flop
p	general print facilities
f	file print facility
B	byte mode
W	word mode
D	double word mode
!	escape to shell

The print facilities generate a formatted output in various styles. The current address is normalized to an appropriate boundary before printing begins. It advances with the printing and is left at the address of the last item printed. The output can be terminated at any time by typing the delete character. If a number follows the **p** symbol, that many entries are printed. A check is made to detect block boundary overflows since logically sequential blocks are generally not physically sequential. If a count of zero is used, all entries to the end of the current block are printed. The print options available are:

i	print as i-nodes
d	print as directories
o	print as octal words

e	print as decimal words
c	print as characters
b	print as octal bytes

The **f** symbol is used to print data blocks associated with the current i-node. If followed by a number, that block of the file is printed. (Blocks are numbered from zero.) The desired print option letter follows the block number, if present, or the **f** symbol. This print facility works for small as well as large files. It checks for special devices and that the block pointers used to find the data are not zero.

Dots, tabs, and spaces may be used as function delimiters but are not necessary. A line with just a new-line character will increment the current address by the size of the data type last printed. That is, the address is set to the next byte, word, double word, directory entry or i-node, allowing the user to step through a region of a file system. Information is printed in a format appropriate to the data type. Bytes, words and double words are displayed with the octal address followed by the value in octal and decimal. A **.B** or **.D** is appended to the address for byte and double word values, respectively. Directories are printed as a directory slot offset followed by the decimal i-number and the character representation of the entry name. I-nodes are printed with labeled fields describing each element.

The following mnemonics are used for i-node examination and refer to the current working i-node:

md	mode
ln	link count
uid	user ID number
gid	group ID number
sz	file size
a#	data block numbers (0 — 12)
at	access time
mt	modification time
maj	major device number
min	minor device number

EXAMPLES

386i	prints i-number 386 in an i-node format. This now becomes the current working i-node.
ln=4	changes the link count for the working i-node to 4.
ln=+1	increments the link count by 1.
fc	prints, in ASCII, block zero of the file associated with the working i-node.
2i.fd	prints the first 32 directory entries for the root i-node of this file system.
d5i.fc	changes the current i-node to that associated with the 5th directory entry (numbered from zero) found from the above command. The first logical block of the file is then printed in ASCII.
512B.p0o	prints the superblock of this file system in octal.
2i.a0b.d7=3	changes the i-number for the seventh directory slot in the root directory to 3. This example also shows how several operations can be combined on one command line.
d7.nm="name"	changes the name field in the directory slot to the given string. Quotes are optional when used with **nm** if the first character is alphabetic.
a2b.p0d	prints the third block of the current i-node as directory entries.

SEE ALSO

fsck(1M).
dir(4), fs(4) in the *Programmer's Reference Manual.*

NAME

fsstat — report file system status

SYNOPSIS

/etc/fsstat special_file

DESCRIPTION

Fsstat reports on the status of the file system on *special_file*. During startup, this command is used to determine if the file system needs checking before it is mounted. *Fsstat* succeeds if the file system is unmounted and appears okay. For the root file system, it succeeds if the file system is active and not marked bad.

SEE ALSO

fs(4) in the *Programmer's Reference Manual.*

DIAGNOSTICS

The command has the following exit codes:

0 – the file system is not mounted and appears okay,
(except for root where 0 means mounted and okay).
1 – the file system is not mounted and needs to be checked.
2 – the file system is mounted.
3 – the command failed.

NAME

fstyp — determine file system identifier

SYNOPSIS

fstyp special

DESCRIPTION

Fstyp allows the user to determine the file system identifier of mounted or unmounted file systems using heuristic programs. The file system type is required by *mount*(2) and sometimes by *mount*(1M) to mount file systems of different types.

The directory **/etc/fstyp.d** contains a program for each file system type to be checked; each of these programs applies some appropriate heuristic to determine whether the supplied *special* file is of the type for which it checks. If it is, the program prints on standard output the usual file-system identifier for that type and exits with a return code of 0; otherwise it prints error messages on standard error and exits with a non-zero return code. *Fstyp* runs the programs in **/etc/fstyp.d** in alphabetical order, passing *special* as an argument; if any program succeeds, its file-system type identifier is printed and *fstyp* exits immediately. If no program succeeds, *fstyp* prints "Unknown_fstyp" to indicate failure.

WARNING

The use of heuristics implies that the result of *fstyp* is not guaranteed to be accurate.

SEE ALSO

mount(1M).
mount(2), sysfs(2) in the *Programmer's Reference Manual.*

NAME

fumount — forced unmount of an advertised resource

SYNOPSIS

fumount [-w sec] resource

DESCRIPTION

Fumount unadvertises *resource* and disconnects remote access to the resource. The *-w sec* causes a delay of *sec* seconds prior to the execution of the disconnect.

When the forced unmount occurs, an administrative shell script is started on each remote computer that has the resource mounted (**/usr/bin/rfuadmin**). If a grace period of seconds is specified, **rfuadmin** is started with the *fuwarn* option. When the actual forced unmount is ready to occur, **rfuadmin** is started with the *fumount* option. See the **rfuadmin**(1M) man page for information on the action taken in response to the forced unmount.

This command is restricted to the super-user.

ERRORS

If *resource* (1) does not physically reside on the local machine, (2) is an invalid resource name, (3) is not currently advertised and is not remotely mounted, or (4) the command is not run with super-user privileges, an error message will be sent to standard error.

SEE ALSO

adv(1M), mount(1M), rfuadmin(1M), rfudaemon(1M), rmount(1M), unadv(1M).

NAME

fusage — disk access profiler

SYNOPSIS

fusage [[*mount_point*] | [advertised_resource] | [block_special_device] [...]]

DESCRIPTION

When used with no options, *fusage* reports block I/O transfers, in kilobytes, to and from all locally mounted file systems and advertised Remote File Sharing resources on a per client basis. The count data are cumulative since the time of the mount. When used with an option, *fusage* reports on the named file system, advertised resource, or block special device.

The report includes one section for each file system and advertised resource and has one entry for each machine that has the directory remotely mounted, ordered by decreasing usage. Sections are ordered by device name; advertised resources that are not complete file systems will immediately follow the sections for the file systems they are in.

SEE ALSO

adv(1M), mount(1M), df(1M), crash(1M).

NAME

fuser — identify processes using a file or file structure

SYNOPSIS

/etc/fuser [**—ku**] files | resources [—] [[**—ku**] files | resources]

DESCRIPTION

Fuser outputs the process IDs of the processes that are using the *files* or remote *resources* specified as arguments. Each process ID is followed by a letter code, interpreted as follows: if the process is using the file as 1) its current directory, the code is **c**, 2) the parent of its current directory (only when the file is being used by the system), the code is **p**, or 3) its root directory, the code is **r**. For block special devices with mounted file systems, all processes using any file on that device are listed. For remote resource names, all processes using any file associated with that remote resource (Remote File Sharing) are reported. (*Fuser* cannot use the mount point of the remote resource; it must use the resource name.) For all other types of files (text files, executables, directories, devices, etc.) only the processes using that file are reported.

The following options may be used with *fuser*:

—u the user login name, in parentheses, also follows the process ID.

—k the SIGKILL signal is sent to each process. Since this option spawns kills for each process, the kill messages may not show up immediately [see *kill*(2)].

If more than one group of files are specified, the options may be respecified for each additional group of files. A lone dash cancels the options currently in force; then, the new set of options applies to the next group of files.

The process IDs are printed as a single line on the standard output, separated by spaces and terminated with a single new line. All other output is written on standard error.

You cannot list processes using a particular file from a remote resource mounted on your machine. You can only use the resource name as an argument.

Any user with permission to read **/dev/kmem** and **/dev/mem** can use *fuser*. Only the super-user can terminate another user's process.

FILES

/unix	for system namelist
/dev/kmem	for system image
/dev/mem	also for system image

SEE ALSO

mount(1M).
ps(1) in the *User's Reference Manual*.
kill(2), signal(2) in the *Programmer's Reference Manual*.

NAME

fwtmp, wtmpfix — manipulate connect accounting records

SYNOPSIS

/usr/lib/acct/fwtmp [—ic]
/usr/lib/acct/wtmpfix [files]

DESCRIPTION

fwtmp

Fwtmp reads from the standard input and writes to the standard output, converting binary records of the type found in **wtmp** to formatted ASCII records. The ASCII version is useful to enable editing, via *ed*(1), bad records or general purpose maintenance of the file.

The argument **—ic** is used to denote that input is in ASCII form, and output is to be written in binary form.

wtmpfix

Wtmpfix examines the standard input or named files in **wtmp** format, corrects the time/date stamps to make the entries consistent, and writes to the standard output. A — can be used in place of *files* to indicate the standard input. If time/date corrections are not performed, *acctcon*(1) will fault when it encounters certain date-change records.

Each time the date is set, a pair of date change records are written to **/etc/wtmp**. The first record is the old date denoted by the string **old time** placed in the line field and the flag **OLD_TIME** placed in the type field of the **<utmp.h>** structure. The second record specifies the new date and is denoted by the string **new time** placed in the line field and the flag **NEW_TIME** placed in the type field. *Wtmpfix* uses these records to synchronize all time stamps in the file.

In addition to correcting time/date stamps, *wtmpfix* will check the validity of the name field to ensure that it consists solely of alphanumeric characters or spaces. If it encounters a name that is considered invalid, it will change the login name to **INVALID** and write a diagnostic to the standard error. In this way, *wtmpfix* reduces the chance that *acctcon*(1) will fail when processing connect accounting records.

FILES

/etc/wtmp
/usr/include/utmp.h

SEE ALSO

acct(1M), acctcms(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), runacct(1M)
acctcom(1), ed(1) in the *User's Reference Manual*
acct(2), acct(4), utmp(4) in the *Programmer's Reference Manual*

NAME

getty — set terminal type, modes, speed, and line discipline

SYNOPSIS

/etc/getty [**—h**] [**—t** timeout] line [speed [type [linedisc]]]
/etc/getty **—c** file

DESCRIPTION

Getty is a program that is invoked by *init*(1M). It is the second process in the series, (*init-getty-login-shell*) that ultimately connects a user with the GENIX V.3 system. It can only be executed by the super-user; that is, a process with the user-ID of **root**. Initially *getty* prints the login message field for the entry it is using from **/etc/gettydefs**. *Getty* reads the user's login name and invokes the *login*(1) command with the user's name as argument. While reading the name, *getty* attempts to adapt the system to the speed and type of terminal being used. It does this by using the options and arguments specified.

Line is the name of a tty line in **/dev** to which *getty* is to attach itself. *Getty* uses this string as the name of a file in the **/dev** directory to open for reading and writing. Unless *getty* is invoked with the **—h** flag, *getty* will force a hangup on the line by setting the speed to zero before setting the speed to the default or specified speed. The **—t** flag plus *timeout* (in seconds), specifies that *getty* should exit if the open on the line succeeds and no one types anything in the specified number of seconds.

Speed, the optional second argument, is a label to a speed and tty definition in the file **/etc/gettydefs**. This definition tells *getty* at what speed to initially run, what the login message should look like, what the initial tty settings are, and what speed to try next should the user indicate that the speed is inappropriate (by typing a <*break*> character). The default *speed* is 300 baud.

Type, the optional third argument, is a character string describing to *getty* what type of terminal is connected to the line in question. *Getty* recognizes the following types:

none	default
ds40-1	Dataspeed40/1
tektronix,tek	Tektronix
vt61	DEC vt61
vt100	DEC vt100
hp45	Hewlett-Packard 45
c100	Concept 100

The default terminal is **none**; *i.e.*, any crt or normal terminal unknown to the system. Also, for terminal type to have any meaning, the virtual terminal handlers must be compiled into the operating system. They are available, but not compiled in the default condition.

Linedisc, the optional fourth argument, is a character string describing which line discipline to use in communicating with the terminal. Again the hooks for line disciplines are available in the operating system but there is only one presently available, the default line discipline, **LDISC0**.

When given no optional arguments, *getty* sets the *speed* of the interface to 300 baud, specifies that raw mode is to be used (awaken on every character), that echo is to be suppressed, either parity allowed, new-line characters will be converted to carriage return-line feed, and tab expansion performed on the standard output. It types the login message before reading the user's name a character at a time. If a null character (or framing error) is received, it is assumed to be the result of the user pushing the "break" key. This will cause *getty* to attempt the next *speed* in the series. The series that *getty* tries is determined by what it finds in **/etc/gettydefs**.

After the user's name has been typed in, it is terminated by a new-line or carriage-return character. The latter results in the system being set to treat carriage returns appropriately (see *ioctl*(2)).

The user's name is scanned to see if it contains any lower-case alphabetic characters; if not, and if the name is non-empty, the system is told to map any future upper-case characters into the corresponding lower-case characters.

Finally, *login* is **exec**'d with the user's name as an argument. Additional arguments may be typed after the login name. These are passed to *login*, which will place them in the environment (see *login*(1)).

A check option is provided. When *getty* is invoked with the **—c** option and *file*, it scans the file as if it were scanning **/etc/gettydefs** and prints out the results to the standard output. If there are any unrecognized modes or improperly constructed entries, it reports these. If the entries are correct, it prints out the values of the various flags. See *ioctl*(2) to interpret the values. Note that some values are added to the flags automatically.

FILES

/etc/gettydefs
/etc/issue

SEE ALSO

ct(1C), init(1M), tty(7).
login(1) in the *User's Reference Manual.*
ioctl(2), gettydefs(4), inittab(4) in the *Programmer's Reference Manual.*

BUGS

While *getty* understands simple single character quoting conventions, it is not possible to quote certain special control characters used by *getty*. Thus, you cannot login via *getty* and type a **#**, **@**, **/**, **!**, **␣**, backspace, **^U**, **^D**, or **&** as part of your login name or arguments. *Getty* uses them to determine when the end of the line has been reached, which protocol is being used, and what the erase character is. They will always be interpreted as having their special meaning.

NAME

hdeadd — add/delete hdelog (Hard Disk Error Log) reports

SYNOPSIS

/etc/hdeadd —a [*aoptions*]

/etc/hdeadd —d [*doptions*]

/etc/hdeadd —e [[**—D**] major minor]

/etc/hdeadd —f filename

/etc/hdeadd —r [**—D**] major minor filename

/etc/hdeadd —s [**—D**] major minor filename

DESCRIPTION

This command is part of the bad block handling utility. It may be used only by the super-user for manually adding or deleting disk error reports recorded by **hdelogger.** These include disk errors reported while in firmware mode and disk errors that cause the system to PANIC.

Hdeadd may be used to print the list of equipped disks or to determine if a specific disk device is on the list. In addition, this command has some options that are for use in testing the feature.

The following options may be used with *hdeadd*:

—a	*hdeadd* allows a Hard Disk Error (HDE) report to be added manually to the HDE Log of a disk.
—d	*hdeadd* allows a specific report or a range of reports to be deleted from the HDE Log of a disk.
—e	prints out the list of major/minor device numbers of the equipped hard disks. If the *major* and *minor* device numbers are also provided, it determines if that specification is an equipped hard disk. The result is both printed on the standard output and is used to determine the exit status. A NORMAL (or TRUE) exit means it is an equipped disk.
—f	the file specified by *filename* is assumed to contain a canned set of HDE Log manipulations. Each line of text contains one specification in the command argument form, starting with a **—a** or a **—d** option.
—s	saves a copy of the HDE Log of the specified (by *major/minor* device number) disk in the file specified by *filename*.
—r	restores the HDE Log of the specified disk from the file specified by *filename*.

The valid *aoptions* are only *hard disk error* specifications.

The valid *doptions* are either a *hard disk error* specification or an *error range* specification.

A *hard disk error* specification includes the following values:

—D *maj min*	Specifies the major device number (*maj*) and minor device number (*min*) of the disk.
—b *blockno*	**Normal form:** Specifies the physical disk block number in integer counter form (i.e., treating the disk as a simple stream of blocks). Physical disk block numbering starts with zero meaning sector 0 of track 0 of cylinder 0. This is the normal form that is reported by the operating system.
—B *cyl trk sec*	**Alternate form:** Specifies the physical disk block number in terms of its physical cylinder number (*cyl*), track number within cylinder (*trk*), and sector number within track (*sec*). This alternate form is

available to cover the possibility of a non-operating system detector reporting block numbers in this hardware form.

—t *mmddhhmm*[*yy*]	**Optional:** Specifies the time of day when the error actually occurred. If omitted when adding reports, the current time is used. If omitted when deleting reports, any reports for the given block are deleted.

An *error range* specification includes the following values:

—D *maj min*	Specifies the major device number (*maj*) and minor device number (*min*) of the disk.
—F *mmddhhmm*[*yy*]	**Optional:** Specifies the "from" time for the time interval being purged. If omitted, zero (the beginning of time) is used.
—T *mmddhhmm*[*yy*]	**Optional:** Specifies the "to" time for the time interval being purged. If omitted, the end of time is used. The range comparisons include the end values of the range in the purge.

FILES

/dev/hdelog

SEE ALSO

hdefix(1M), hdelogger(1M), hdelog(7), hdeupdate(1M).
Bad Block Handling in the *Administrator's Guide.*

DIAGNOSTICS

The HDE commands exit with one of three values:

0	means NORMAL, or TRUE
1	means bad command usage or execution errors
2	means BAD BLOCKS or FALSE (but command executed successfully)

NAME

hdefix — report or change bad block mapping

SYNOPSIS

/etc/hdefix —p [[**—D**] major minor]

/etc/hdefix —a [major minor [*blocknospec* ...]]

/etc/hdefix —F [**—D**] major minor [*blocknospec* ...]]

/etc/hdefix —r [**—D**] major minor filename

/etc/hdefix —s [**—D**] major minor filename

DESCRIPTION

This command is part of the bad block handling utility. It causes bad blocks to be mapped to surrogate images in the media-specific data portion of the disk.

The superuser may use the *hdefix* command in single-user state to find out what blocks are currently mapped to surrogate images on the equipped hard disks and to change what blocks are mapped.

When the mapping to surrogate images is changed, block initialization is performed. The original block is assumed to be unreadable and zeros are written to the new surrogate image. There is a high probability that this will result in some data loss.

If the block is associated with a file system, the file system may be damaged as a result of the mapping change. To handle this situation, the file system is marked dirty, which means *fsck*(1M) must be run before the file system can be used, and a system reboot is forced after all other bad block processing is complete. If the block is a data block of a file, that file will be corrupted, even after this recovery has finished.

The following options may be used with *hdefix*:

—p *hdefix* prints a report of media properties that includes the currently mapped bad blocks. If a disk is specified (by giving its *major* and *minor* device numbers), only the report for that disk is printed. If no disk is specified, a report is given for each equipped disk.

—D used to specify the major device number (*maj*) and minor device number (*min*) of the disk.

—a To map new bad blocks, the **—a** option is used. If no arguments follow the **—a** option, each equipped disk is processed, using the HDE Log on each disk to determine which blocks to map. If a disk is specified, only that disk is processed. If one or more block numbers are specified, those blocks are mapped, instead of using the HDE Log to get blocks to map. This is the only way to map an unreadable block containing the HDE Log.

—F forces blocks to be removed from the map without any attempt to initialize them. This is intended only for testing the bad block handling feature. If no block number is specified, the last block in the map is removed and the block number is printed on the standard output.

—s a copy of the bad block map table and the surrogates pointed to by the map are saved in the file specified by the *filename* argument.

—r the bad block map table and the surrogates pointed to by the map are restored from the file specified by the *filename* argument. The save and restore options are intended only for testing. This option should only be used on a quiescent system.

A *blocknospec* has the following forms:

—b *blockno*	Specifies the physical disk block number in integer counter form (i.e., treating the disk as a simple stream of blocks). Physical disk block numbering starts with zero, meaning sector 0 of track 0 of cylinder 0.
—B *cyl trk sec*	Specifies the physical disk block number in terms of its physical cylinder number (*cyl*), track number within cylinder (*trk*), and sector number within track (*sec*). Only one of the two alternate forms of block number should be specified for a given block.

FILES

/dev/hdelog

SEE ALSO

hdeadd(1M), hdelogger(1M), hdelog(7), hdeupdate(1M).
Bad Block Handling in the *Administrator's Guide.*

DIAGNOSTICS

The HDE commands exit with one of three values:

0	means NORMAL, or TRUE
1	means bad command usage or execution errors
2	means BAD BLOCKS or FALSE (but command executed successfully)

NAME

hdelogger — Hard Disk Error status report command and Log Daemon

SYNOPSIS

/etc/hdelogger [**—s**] [**—f**] [**—D** maj min]

DESCRIPTION

This command is part of the bad block handling utility. It is executed automatically by the **init** in run levels 2, 3 and 4.

The *hdelogger* command serves two purposes. When run by the **init** process (process 1 — see *init*(1M)), this command performs the functions of the Hard Disk Error (HDE) Log Daemon. These functions include providing summaries of outstanding errors during system startup and shutdown transitions, along with adding new errors to HDE Logs and giving the revised status summaries as errors are reported by hard disk drivers. When run as the daemon, no options are used.

When run as a normal command (process 1 is not its parent), this command provides on the spot reports of outstanding errors as recorded in the HDE Logs of equipped hard disks. You must be the super-user to run the command this way. The following options control report generation:

—s	Specifies that summary reports are to be generated. The summary report provides sufficient information for normal bad block handling operations. This is the default.
—f	Specifies that full reports are to be generated. This is intended mainly for testing the bad block handling feature, but is available in case additional detail is needed for troubleshooting complicated problems.
—D *maj min*	Restricts the report generation to a specific hard disk. If this option is omitted, reports will be generated for all equipped hard disks.

FILES

/dev/hdelog

SEE ALSO

hdeadd(1M), hdefix(1M), hdelog(7), hdeupdate(1M).
Bad Block Handling, in the *Administrator's Guide.*

DIAGNOSTICS

The HDE commands exit with one of three values:

0	means NORMAL, or TRUE
1	means bad command usage or execution errors
2	means BAD BLOCKS or FALSE (but command executed successfully)

NAME

hdeupdate — display/modify physical disk configuration

SYNOPSIS

/etc/hdeupdate —p —D major minor [**—u**]
/etc/hdeupdate —m —D major minor [**—u**]
/etc/hdeupdate —c —D major minor

DESCRIPTION

This command is part of the bad block handling utility. It may be used only by the super-user for manually allocating and de-allocating areas on disk to be used by HDE utilities for logging hard disk errors and bad block forwarding.

Hdeupdate may also be used to print and modify the physical disk configuration as stored in physical block 0 of cylinder 0 of each physical disk. It also allows super-user to relocate the *Stand Alone Shell* (SASH) almost anywhere on physical cylinder 0.

The following options may be used with *hdeupdate*:

—p *hdeupdate* will print out all of physical disk configuration as stored on physical block 0 of specified disk. The inclusion of **—u** option will allow modification of displayed information.

—m *hdeupdate* will print out the block defect map as stored on specified disk. The inclusion of **—u** option will allow modification of displayed information.

—c *hdeupdate* will duplicate function of a stand alone program *cpboot* to relocate, or reload a new version of, *Stand Alone Shell* (SASH).

—D *maj min* Specifies the major device number (*maj*) and minor device number (*min*) of the disk. Unlike other HDE utilities, this option is mandatory.

FILES

/dev/hdelog

SEE ALSO

hdeadd(1M), hdefix(1M), hdelogger(1M), hdelog(7).
Bad Block Handling in the *Administrator's Guide.*

DIAGNOSTICS

The HDE commands exit with one of three values:

0	means NORMAL, or TRUE
1	means bad command usage or execution errors
2	means BAD BLOCKS or FALSE (but command executed successfully)

NAME

helpadm — make changes to the Help Facility database

SYNOPSIS

/etc/helpadm

DESCRIPTION

The GENIX V.3 system Help Facility Administration command, *helpadm,* allows GENIX V.3 system administrators and command developers to define the content of the Help Facility database for specific commands and to monitor use of the Help Facility. The *helpadm* command can only be executed by login root, login bin, or a login that is a member of group bin.

The *helpadm* command prints a menu of 3 types of Help Facility data which can be modified, and 2 choices relating to monitoring use of the Help Facility. The five choices are:

- modify *startup* data
- add, modify, or delete a *glossary* term
- add, modify, or delete command data (description, options, examples, and keywords)
- prevent monitoring use of the Help Facility (login root and login bin only)
- permit monitoring use of the Help Facility (login root and login bin only)

The user may make one of the above choices by entering its corresponding letter (given in the menu), or may exit to the shell by typing **q** (for "quit").

If one of the first three choices is chosen, then the user is prompted for additional information; specifically, which *startup* screen, *glossary* term definition, or command description is to be modified. The user may also be prompted for information to identify whether the changes to the database are additions, modifications, or deletions. If the user is modifying existing data or adding new data, then they are prompted to make the appropriate modifications/additions. If the user is deleting a *glossary* term or a command from the database, then they must respond affirmatively to the next query in order for the deletion to be done. In any case, before the user's changes are final, they must respond affirmatively when asked whether they are sure they want their requested database changes to be done.

By default, *helpadm* will put the user into *ed*(1) to make additions/modifications to database information. If the user wishes to be put into a different editor, then they should set the environment variable **EDITOR** in their environment to the desired editor, and then export **EDITOR .**

If the user chooses to monitor/prevent monitoring use of the Help Facility, the choice made is acted on with no further interaction by the user.

SEE ALSO

ed(1), glossary(1), help(1), locate(1), starter(1), usage(1).

WARNINGS

When the GENIX V.3 system is delivered to a customer, **/etc/profile** exports the environment variable **LOGNAME** . If **/etc/profile** has been changed so that **LOGNAME** is not exported, then the options to monitor/prevent monitoring use of the Help Facility may not work properly.

FILES

HELPLOG	/usr/lib/help/HELPLOG
helpclean	/usr/lib/help/helpclean

NAME

id — print user and group IDs and names

SYNOPSIS

id

DESCRIPTION

Id outputs the user and group IDs and the corresponding names of the invoking process. If the effective and real IDs are different, both are printed.

SEE ALSO

logname(1) in the *User's Reference Manual.*
getuid(2) in the *Programmer's Reference Manual.*

NAME

idload — Remote File Sharing user and group mapping

SYNOPSIS

idload [-n] [-g *g_rules*] [-u *u_rules*] [*directory*]

DESCRIPTION

Idload is used on Remote File Sharing server machines to build translation tables for user and group IDs. It takes your **/etc/passwd** and **/etc/group** files and produces translation tables for user and group IDs from remote machines, according to the rules set down in the *u_rules* and *g_rules* files. If you are mapping by user and group name, you will need copies of remote **/etc/passwd** and **/etc/group** files. If no rules files are specified, remote user and group IDs are mapped to MAXUID+1 (this is an ID number that is one higher than the highest number you could assign on your system.)

By default, the remote password and group files are assumed to reside in **/usr/nserve/auth.info/***domain/host/***[passwd|group]**. The *directory* argument indicates that some directory structure other than **/usr/nserve/auth.info** contains the *domain/host* **passwd** and **group** files. (*Host* is the name of the host the files are from and *domain* is the domain that host is a member of.)

This command is run automatically when the first remote mount is done of a remote resource [see *mount*(1M)].

-n This is used to do a trial run of the ID mapping. No translation table will be produced, however, a display of the mapping is output to the terminal (*stdout*).

-u *u_rules* The *u_rules* file contains the rules for user ID translation. The default rules file is **/usr/nserve/auth.info/uid.rules.**

-g *g_rules* The *g_rules* file contains the rules for group ID translation. The default rules file is **/usr/nserve/auth.info/gid.rules.**

This command is restricted to the super-user.

Rules

The rules files have two types of sections, both optional: **global** and **host**. There can be only one global section, though there can be one host section for each host you want to map.

The **global** section describes the default conditions for translation for any machines that are not explicitly referenced in a **host** section. If the global section is missing, the default action is to map all remote user and group IDs from undefined hosts to MAXUID+1. The syntax of the first line of the **global** section is:

global

A **host** section is used for each client machine or group of machines that you want to map differently from the global definitions. The syntax of the first line of each **host** section is:

host *name [...]*

where *name* is replaced by the full name(s) of a host (*domain.host*).

The format of a rules file is described below. (All lines are optional, but must appear in the order shown.)

global
default *local* | **transparent**
exclude *[remote_id-remote_id]* | *[remote_id]*
map *[remote_id:local]*

host domain.hostname [domain.hostname...]
default *local* | **transparent**

exclude. [*remote_id-remote_id*] | [*remote_id*] | [*remote_name*]
map [*remote:local*] | *remote* | **all**

Each of these instruction types is described below.

The line

> **default** *local* | **transparent**

defines the mode of mapping for remote users that are not specifically mapped in instructions in other lines. **transparent** means that all remote user and group IDs will have the same numeric value locally unless they appear in the **exclude** instruction. *Local* can be replaced by a local user name or ID to map all users into a particular local name or ID number. If the default line is omitted, all users that are not specifically mapped are mapped into a "special guest" login ID.

The line

> **exclude** [*remote_id-remote_id*] | [*remote_id*] | [*remote_name*]

defines remote IDs that will be excluded from the **default** mapping. The **exclude** instruction must precede any **map** instructions in a block. You can use a range of ID numbers, a single ID number, or a single name. (*Remote_name* cannot be used in a *global* block.)

The line

> **map** [*remote:local*] | *remote* | **all**

defines the local IDs and names that remote IDs and names will be mapped into. *remote* is either a remote ID number or remote name; *local* is either a local ID number or local name. Placing a colon between a *remote* and a *local* will give the value on the left the permissions of the value on the right. A single *remote* name or ID will assign the user or group permissions of the same local name or ID. **All** is a predefined alias for the set of all user and group IDs found in the local **/etc/passwd** and **/etc/group** files. (You cannot map by remote name in **global** blocks.)

NOTE: **Idload** will always output warning messages for **map all**, since password files always contain multiple administrative user names with the same ID number. The first mapping attempt on the ID number will succeed, all subsequent attempts will fail.

Remote File Sharing doesn't need to be running to use **idload**.

EXIT STATUS

On successful completion, **idload** will produce one or more translation tables and return a successful exit status. If **idload** fails, the command will return an unsuccessful exit status without producing a translation table.

ERRORS

If (1) the neither rules files can be found or opened, (2) there are syntax errors in the rules file, (3) there are semantic errors in the rules file, (4) host information could not be found, or (5) the command is not run with super-user privileges, an error message will be sent to standard error. Partial failures will cause a warning message to appear, though the process will continue.

FILES

/etc/passwd
/etc/group
/usr/nserve/auth.info/*domain*/*host*/[**user**|**group**]
/usr/nserve/auth.info/uid.rules
/usr/nserve/auth.info/gid.rules

SEE ALSO

mount(1M).

NAME

infocmp — compare or print out terminfo descriptions

SYNOPSIS

infocmp [**—d**] [**—c**] [**—n**] [**—I**] [**—L**] [**—C**] [**—r**] [**—u**] [**—s** d|i|l|c] [**—v**] [**—V**] [**—1**] [**—w** width] [**—A** directory] [**—B** directory] [termname ...]

DESCRIPTION

Infocmp can be used to compare a binary *terminfo*(4) entry with other terminfo entries, rewrite a *terminfo*(4) description to take advantage of the **use=** terminfo field, or print out a *terminfo*(4) description from the binary file (*term*(4)) in a variety of formats. In all cases, the boolean fields will be printed first, followed by the numeric fields, followed by the string fields.

Default Options

If no options are specified and zero or one *termnames* are specified, the **—I** option will be assumed. If more than one *termname* is specified, the **—d** option will be assumed.

Comparison Options [—d] [—c] [—n]

Infocmp compares the *terminfo*(4) description of the first terminal *termname* with each of the descriptions given by the entries for the other terminal's *termnames*. If a capability is defined for only one of the terminals, the value returned will depend on the type of the capability: F for boolean variables, **—1** for integer variables, and **NULL** for string variables.

—d produce a list of each capability that is different. In this manner, if one has two entries for the same terminal or similar terminals, using *infocmp* will show what is different between the two entries. This is sometimes necessary when more than one person produces an entry for the same terminal and one wants to see what is different between the two.

—c produce a list of each capability that is common between the two entries. Capabilities that are not set are ignored. This option can be used as a quick check to see if the **—u** option is worth using.

—n produce a list of each capability that is in neither entry. If no *termnames* are given, the environment variable **TERM** will be used for both of the *termnames*. This can be used as a quick check to see if anything was left out of the description.

Source Listing Options [—I] [—L] [—C] [—r]

The —I, —L, and —C options will produce a source listing for each terminal named.

—I use the *terminfo*(4) names

—L use the long C variable name listed in <**term.h**>

—C use the *termcap* names

—r when using —C, put out all capabilities in *termcap* form

If no *termnames* are given, the environment variable **TERM** will be used for the terminal name.

The source produced by the —C option may be used directly as a *termcap* entry, but not all of the parameterized strings may be changed to the *termcap* format. *Infocmp* will attempt to convert most of the parameterized information, but that which it doesn't will be plainly marked in the output and commented out. These should be edited by hand.

All padding information for strings will be collected together and placed at the beginning of the string where *termcap* expects it. Mandatory padding (padding information with a trailing '/') will become optional.

All *termcap* variables no longer supported by *terminfo*(4), but which are derivable from other *terminfo*(4) variables, will be output. Not all *terminfo*(4) capabilities will be translated; only those variables which were part of *termcap* will normally be output. Specifying the **—r** option will take off this restriction, allowing all capabilities to be output in *termcap* form.

Note that because padding is collected to the beginning of the capability, not all capabilities are output, mandatory padding is not supported, and *termcap* strings were not as flexible, it is not always possible to convert a *terminfo*(4) string capability into an equivalent *termcap* format. Not all of these strings will be able to be converted. A subsequent conversion of the *termcap* file back into *terminfo*(4) format will not necessarily reproduce the original *terminfo*(4) source.

Some common *terminfo* parameter sequences, their *termcap* equivalents, and some terminal types which commonly have such sequences, are:

Terminfo	Termcap	Representative Terminals
%p1%c	%.	adm
%p1%d	%d	hp, ANSI standard, vt100
%p1%'x'%+%c	%+x	concept
%i	%i	ANSI standard, vt100
%p1%?%'x'%>%t%p1%'y'%+%;	%>xy	concept
%p2 is printed before %p1	%r	hp

Use= Option [—u]

—u produce a *terminfo*(4) source description of the first terminal *termname* which is relative to the sum of the descriptions given by the entries for the other terminals *termnames*. It does this by analyzing the differences between the first *termname* and the other *termnames* and producing a description with **use=** fields for the other terminals. In this manner, it is possible to retrofit generic terminfo entries into a terminal's description. Or, if two similar terminals exist, but were coded at different times or by different people so that each description is a full description, using *infocmp* will show what can be done to change one description to be relative to the other.

A capability will get printed with an at-sign (@) if it no longer exists in the first *termname*, but one of the other *termname* entries contains a value for it. A capability's value gets printed if the value in the first *termname* is not found in any of the other *termname* entries, or if the first of the other *termname* entries that has this capability gives a different value for the capability than that in the first *termname*.

The order of the other *termname* entries is significant. Since the terminfo compiler **tic**(1M) does a left-to-right scan of the capabilities, specifying two **use=** entries that contain differing entries for the same capabilities will produce different results depending on the order that the entries are given in. *Infocmp* will flag any such inconsistencies between the other *termname* entries as they are found.

Alternatively, specifying a capability *after* a **use=** entry that contains that capability will cause the second specification to be ignored. Using *infocmp* to recreate a description can be a useful check to make sure that everything was specified correctly in the original source description.

Another error that does not cause incorrect compiled files, but will slow down the compilation time, is specifying extra **use=** fields that are superfluous. *Infocmp* will flag any other *termname* **use=** fields that were not needed.

Other Options [—s d|i|l|c] [—v] [—V] [—1] [—w width]

—s sort the fields within each type according to the argument below:

d	leave fields in the order that they are stored in the *terminfo* database.
i	sort by *terminfo* name.
l	sort by the long C variable name.
c	sort by the *termcap* name.

If no **—s** option is given, the fields printed out will be sorted alphabetically by the *terminfo* name within each type, except in the case of the **—C** or the **—L** options, which cause the sorting to be done by the *termcap* name or the long C variable name, respectively.

—v print out tracing information on standard error as the program runs.

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

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

—w change the output to width characters.

Changing Databases [—A directory] [—B directory]

The location of the compiled *terminfo*(4) database is taken from the environment variable **TERMINFO.** If the variable is not defined, or the terminal is not found in that location, the system *terminfo*(4) database, usually in */usr/lib/terminfo*, will be used. The options **—A** and **—B** may be used to override this location. The **—A** option will set **TERMINFO** for the first *termname* and the **—B** option will set **TERMINFO** for the other *termnames*. With this, it is possible to compare descriptions for a terminal with the same name located in two different databases. This is useful for comparing descriptions for the same terminal created by different people. Otherwise the terminals would have to be named differently in the *terminfo*(4) database for a comparison to be made.

FILES

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

DIAGNOSTICS

malloc is out of space!
: There was not enough memory available to process all the terminal descriptions requested. Run *infocmp* several times, each time including a subset of the desired *termnames*.

use= order dependency found:
: A value specified in one relative terminal specification was different from that in another relative terminal specification.

'use=*term*' did not add anything to the description.
: A relative terminal name did not contribute anything to the final description.

must have at least two terminal names for a comparison to be done.
: The **—u**, **—d** and **—c** options require at least two terminal names.

SEE ALSO

tic(1M), curses(3X), term(4), terminfo(4) in the *Programmer's Reference Manual*.
captoinfo(1M) in the *Administrator's Reference Manual*.
Chapter 10 of the *Programmer's Guide*.

NOTE

The *termcap* database (from earlier releases of GENIX V.3 System) may not be supplied in future releases.

NAME

init, telinit — process control initialization

SYNOPSIS

/etc/init [**0123456SsQq**]

/etc/telinit [**0123456sSQqabc**]

DESCRIPTION

Init

Init is a general process spawner. Its primary role is to create processes from information stored in the file **/etc/inittab** (see *inittab*(4)). This file usually has *init* spawn *getty*'s on each line that a user may log in on. It also controls autonomous processes required by any particular system.

Init considers the system to be in a *run-level* at any given time. A *run-level* can be viewed as a software configuration of the system where each configuration allows only a selected group of processes to exist. The processes spawned by *init* for each of these *run-levels* is defined in the *inittab* file. *Init* can be in one of eight *run-levels*, **0—6** and **S** or **s**. The *run-level* is changed by having a privileged user run **/etc/init**. This user-spawned *init* sends appropriate signals to the original *init* spawned by the operating system when the system was rebooted, telling it which *run-level* to change to.

Init is invoked inside the GENIX V.3 system as the last step in the boot procedure. First *init* looks in **/etc/inittab** for the *initdefault* entry (see *inittab*(4)). If there is one, *init* uses the *run-level* specified in that entry as the initial *run-level* to enter. If this entry is not in */etc/inittab*, *init* requests that the user enter a *run-level* from the virtual system console, **/dev/console.** If an **S** or an **s** is entered, *init* goes into the SINGLE USER state. This is the only *run-level* that doesn't require the existence of a properly formatted */etc/inittab* file. If it doesn't exist, then by default the only legal *run-level* that *init* can enter is the SINGLE USER state. In the SINGLE USER state the virtual console terminal **/dev/console** is opened for reading and writing and the command **/bin/su** is invoked immediately. To exit from the SINGLE USER state, use either *init* or *telinit*, to signal *init* to change the *run-level* of the system. Note that if the shell is terminated (via an end-of-file), *init* will only re-initialize to the SINGLE USER state.

When attempting to boot the system, failure of *init* to prompt for a new *run-level* may be due to the fact that the device **/dev/console** is linked to a device other than the physical system console (**/dev/contty**). If this occurs, *init* can be forced to relink **/dev/console** by typing a delete on the system console which is colocated with the processor.

When *init* prompts for the new *run-level*, the operator may enter only one of the digits **0** through **6** or the letters **S** or **s**. If **S** or **s** is entered, *init* operates as previously described in the SINGLE USER state with the additional result that **/dev/console** is linked to the user's terminal line, thus making it the virtual system console. A message is generated on the physical console, **/dev/contty**, saying where the virtual terminal has been relocated.

When *init* comes up initially and whenever it switches out of SINGLE USER state to normal run states, it sets the *ioctl*(2) states of the virtual console, **/dev/console**, to those modes saved in the file **/etc/ioctl.syscon**. This file is written by *init* whenever the SINGLE USER state is entered.

If a **0** through **6** is entered *init* enters the corresponding *run-level*. Any other input will be rejected and the user will be re-prompted. Note that, on the *Series 32000* Computer, the *run-levels* **0**, **1**, **5**, and **6** are reserved states for shutting the system down; the *run-levels* **2**, **3**, and **4** are available as normal operating states.

If this is the first time *init* has entered a *run-level* other than SINGLE USER, *init* first scans *inittab* for special entries of the type *boot* and *bootwait*. These entries are performed, providing the *run-level* entered matches that of the entry before any normal processing of *inittab* takes place. In this way any special initialization of the operating system, such as mounting file systems, can take place before users are allowed onto the system. The *inittab* file is scanned to find all entries that are to be processed for that *run-level*.

Run-level 2 is defined to contain all of the terminal processes and daemons that are spawned in the multi-user environment. Hence, it is commonly referred to as the MULTI-USER state. *Run-level* 3 is defined to start up remote file sharing processes and daemons as well as mount and advertise remote resources. So, *run-level* 3 extends multi-user mode and is know as the Remote File Sharing state. *Run-level* 4 is available to be defined as an alternative multi-user environment configuration, however, it is not necessary for system operation and is usually unused.

In a MULTI-USER environment, the *inittab* file is set up so that *init* will create a process for each terminal on the system that the administrator sets up to respawn.

For terminal processes, ultimately the shell will terminate because of an end-of-file either typed explicitly or generated as the result of hanging up. When *init* receives a signal telling it that a process it spawned has died, it records the fact and the reason it died in **/etc/utmp** and **/etc/wtmp** if it exists (see *who*(1)). A history of the processes spawned is kept in **/etc/wtmp.**

To spawn each process in the *inittab* file, *init* reads each entry and for each entry that should be respawned, it forks a child process. After it has spawned all of the processes specified by the *inittab* file, *init* waits for one of its descendant processes to die, a powerfail signal, or until *init* is signaled by *init* or *telinit* to change the system's *run-level*. When one of these conditions occurs, *init* re-examines the *inittab* file. New entries can be added to the *inittab* file at any time; however, *init* still waits for one of the above three conditions to occur. To get around this, **init Q** or **init q** command wakes *init* to re-examine the *inittab* file immediately.

If *init* receives a *powerfail* signal (*SIGPWR*) it scans *inittab* for special entries of the type *powerfail* and *powerwait*. These entries are invoked (if the *run-levels* permit) before any further processing takes place. In this way *init* can perform various cleanup and recording functions during the powerdown of the operating system. Note that in the SINGLE-USER state only *powerfail* and *powerwait* entries are executed.

When *init* is requested to change *run-levels* (via *telinit*), *init* sends the warning signal (**SIGTERM**) to all processes that are undefined in the target *run-level*. *Init* waits 5 seconds before forcibly terminating these processes via the kill signal (**SIGKILL**).

Telinit

Telinit, which is linked to **/etc/init** , is used to direct the actions of *init*. It takes a one-character argument and signals *init* via the *kill* system call to perform the appropriate action. The following arguments serve as directives to *init*.

0—6	tells *init* to place the system in one of the *run-levels* **0—6.**
a,b,c	tells *init* to process only those **/etc/inittab** file entries having the **a, b** or **c** *run-level* set. These are pseudo-states, which may be defined to run certain commands, but which do not cause the current *run-level* to change.
Q,q	tells *init* to re-examine the **/etc/inittab** file.
s,S	tells *init* to enter the single user environment. When this level change is effected, the virtual system teletype, **/dev/console,** is changed to the terminal from which the command was executed.

FILES

/etc/inittab
/etc/utmp
/etc/wtmp

/etc/ioctl.syscon
/dev/console
/dev/contty

SEE ALSO

getty(1M), termio(7).
login(1), sh(1), who(1) in the *User's Reference Manual.*
kill(2), inittab(4), utmp(4) in the *Programmer's Reference Manual.*

DIAGNOSTICS

If *init* finds that it is respawning an entry from **/etc/inittab** more than 10 times in 2 minutes, it will assume that there is an error in the command string in the entry, and generate an error message on the system console. It will then refuse to respawn this entry until either 5 minutes has elapsed or it receives a signal from a user-spawned *init* (*telinit*). This prevents *init* from eating up system resources when someone makes a typographical error in the *inittab* file or a program is removed that is referenced in the *inittab.*

WARNINGS

Telinit can be run only by someone who is superuser or a member of group **sys.**

BUGS

Attempting to relink **/dev/console** with **/dev/contty** by typing a delete on the system console does not work.

NAME

install — install commands

SYNOPSIS

/etc/install [**—c** dira] [**—f** dirb] [**—i**] [**—n** dirc] [**—m** mode] [**—u** user] [**—g** group] [**—o**] [**—s**] file [dirx ...]

DESCRIPTION

The *install* command is most commonly used in "makefiles" [See *make*(1)] to install a *file* (updated target file) in a specific place within a file system. Each *file* is installed by copying it into the appropriate directory, thereby retaining the mode and owner of the original command. The program prints messages telling the user exactly what files it is replacing or creating and where they are going.

If no options or directories (*dirx* ...) are given, *install* will search a set of default directories (**/bin**, **/usr/bin**, **/etc**, **/lib**, and **/usr/lib**, in that order) for a file with the same name as *file*. When the first occurrence is found, *install* issues a message saying that it is overwriting that file with *file*, and proceeds to do so. If the file is not found, the program states this and exits without further action.

If one or more directories (*dirx* ...) are specified after *file*, those directories will be searched before the directories specified in the default list.

The meanings of the options are:

—c *dira*	Installs a new command (*file*) in the directory specified by *dira*, only if it is not found. If it is found, *install* issues a message saying that the file already exists, and exits without overwriting it. May be used alone or with the **—s** option.
—f *dirb*	Forces *file* to be installed in given directory, whether or not one already exists. If the file being installed does not already exist, the mode and owner of the new file will be set to **755** and **bin**, respectively. If the file already exists, the mode and owner will be that of the already existing file. May be used alone or with the **—o** or **—s** options.
—i	Ignores default directory list, searching only through the given directories (*dirx* ...). May be used alone or with any other options except **—c** and **—f**.
—n *dirc*	If *file* is not found in any of the searched directories, it is put in the directory specified in *dirc*. The mode and owner of the new file will be set to **755** and **bin**, respectively. May be used alone or with any other options except **—c** and **—f**.
—m *mode*	The mode of the new file is set to *mode*. Only available to the superuser.
—u *user*	The owner of the new file is set to *user*. Only available to the superuser.
—g *group*	The group ID of the new file is set to *group*. Only available to the superuser.
—o	If *file* is found, this option saves the "found" file by copying it to **OLD***file* in the directory in which it was found. This option is useful when installing a frequently used file such as */bin/sh* or */etc/getty*, where the existing file cannot be removed. May be used alone or with any other options except **—c**.
—s	Suppresses printing of messages other than error messages. May be used alone or with any other options.

SEE ALSO
make(1).

NAME

killall — kill all active processes

SYNOPSIS

/etc/killall [signal]

DESCRIPTION

Killall is used by **/etc/shutdown** to kill all active processes not directly related to the shutdown procedure.

Killall terminates all processes with open files so that the mounted file systems will be unbusied and can be unmounted.

Killall sends *signal* (see *kill*[1]) to all processes not belonging to the above group of exclusions. If no *signal* is specified, a default of **9** is used.

FILES

/etc/shutdown

SEE ALSO

fuser(1M), shutdown(1M).
kill(1), ps(1) in the *User's Reference Manual.*
signal(2) in the *Programmer's Reference Manual.*

WARNINGS

The *killall* command can be run only by the superuser.

NAME

labelit — provide labels for file systems

SYNOPSIS

/etc/labelit special [fsname volume [**—n**]]

DESCRIPTION

Labelit can be used to provide labels for unmounted disk file systems or file systems being copied to tape. The **—n** option provides for initial labeling only (this destroys previous contents).

With the optional arguments omitted, *labelit* prints current label values.

The *special* name should be the physical disk section (*e.g.*, **/dev/dsk/c0d0s2**), or the cartridge tape (*e.g.*, **/dev/SA/ctape1**). The device may not be on a remote machine.

The *fsname* argument represents the mounted name (*e.g.*, **root**, **u1**, etc.) of the file system.

Volume may be used to equate an internal name to a volume name applied externally to the disk pack, diskette or tape.

For file systems on disk, *fsname* and *volume* are recorded in the superblock.

SEE ALSO

makefsys(1M).
sh(1) in the *User's Reference Manual.*
fs(4) in the *Programmer's Reference Manual.*

NAME

ldsysdump — load system dump from a tape or floppy diskettes

SYNOPSIS

/etc/ldsysdump [-t] [-f device] destination_file

DESCRIPTION

The *ldsysdump* command loads the memory image file from either a tape or the floppy diskettes used to take a system dump and recombines them into a single file on the hard disk suitable for use by *crash*(1). The *destination_file* is the name of the hard disk file into which the data from the tape or diskettes will be loaded.

When invoked for a tape, *ldsysdump* prompts the user to load the dump tape. The user has the option of quitting before starting the tape load.

When invoked for floppy diskettes, *ldsysdump* begins an interactive procedure that prompts the user to insert the diskettes to be loaded. The user has the option of quitting the session at any time. This allows only the portion of the system image needed to be dumped.

OPTIONS

-t — tells *ldsysdump* to load system image from a tape
-f — tells *ldsysdump* to use *device* as an alternate path name where to read the dump from

EXAMPLES

This example loads the 4 floppies produced via *sysdump*(8) or a panic crash on a machine equipped with 2 MB of memory.

```
$ ldsysdump /usr/tmp/cdump

Will extract kernel core dump from '/dev/diskette' to '/usr/tmp/cdump'

Insert kernel dump floppy #1  -  Enter 'q' to quit or 'return' to continue:

EXTRACT 4096 (512 byte) blocks of kernel dump from 4 floppies

32 blocks/dot: ................................

Insert kernel dump floppy #2  -  Enter 'q' to quit or 'return' to continue:
32 blocks/dot: ................................

Insert kernel dump floppy #3  -  Enter 'q' to quit or 'return' to continue:
32 blocks/dot: ................................

Insert kernel dump floppy #4  -  Enter 'q' to quit or 'return' to continue:
32 blocks/dot: ........

4 kernel dump floppies coalesced, 4096 (512 byte) blocks
$
```

FILES

/dev/diskette	device used for floppy access
/dev/mt/0m	device used for tape access

SEE ALSO

crash(1M), sysdump(8).
ulimit(1) in the *User's Reference Manual.*

DIAGNOSTICS

If a floppy diskette is inserted out of sequence or an I/O error occurs during a floppy read, a message is printed. The user is allowed to insert a new floppy and continue the session.

WARNINGS

Since the *Series 32000* computer can be equipped with up to 8 MB of memory, the *destination_file* can become quite large. The file size limit must be set large enough to hold a file of this size.

NAME

link, unlink — link and unlink files and directories

SYNOPSIS

/etc/link file1 file2
/etc/unlink file

DESCRIPTION

The *link* command is used to create a file name that points to another file. Linked files and directories can be removed by the *unlink* command; however, it is strongly recommended that the *rm*(1) and *rmdir*(1) commands be used instead of the *unlink* command.

The only difference between *ln*(1) and *link/unlink* is that the latter do exactly what they are told to do, abandoning all error checking. This is because they directly invoke the *link*(2) and *unlink*(2) system calls.

SEE ALSO

rm(1) in the *User's Reference Manual.*
link(2), unlink(2) in the *Programmer's Reference Manual.*

WARNINGS

These commands can be run only by the superuser.

NAME

lpadmin — configure the LP spooling system

SYNOPSIS

/usr/lib/lpadmin **—p** printer [options]
/usr/lib/lpadmin **—x** dest
/usr/lib/lpadmin **—d**[dest]

DESCRIPTION

Lpadmin configures line printer (LP) spooling systems to describe printers, classes and devices. It is used to add and remove destinations, change membership in classes, change devices for printers, change printer interface programs and to change the system default destination. *Lpadmin* may not be used when the LP scheduler, *lpsched*(1M), is running, except where noted below.

Exactly one of the **—p**, **—d** or **—x** options must be present for every legal invocation of *lpadmin*.

—p*printer*	names a *printer* to which all of the *options* below refer. If *printer* does not exist then it will be created.
—x*dest*	removes destination *dest* from the LP system. If *dest* is a printer and is the only member of a class, then the class will be deleted, too. No other *options* are allowed with **—x**.
—d[*dest*]	makes *dest*, an existing destination, the new system default destination. If *dest* is not supplied, then there is no system default destination. This option may be used when *lpsched*(1M) is running. No other *options* are allowed with **—d**.

The following *options* are only useful with **—p** and may appear in any order. For ease of discussion, the printer will be referred to as *P* below.

—c*class*	inserts printer *P* into the specified *class*. *Class* will be created if it does not already exist.
—e*printer*	copies an existing *printer's* interface program to be the new interface program for *P*.
—h	indicates that the device associated with *P* is hardwired. This *option* is assumed when adding a new printer unless the **—l** *option* is supplied.
—i*interface*	establishes a new interface program for *P*. *Interface* is the path name of the new program.
—l	indicates that the device associated with *P* is a login terminal. The LP scheduler, *lpsched*, disables all login terminals automatically each time it is started. Before re-enabling *P*, its current *device* should be established using *lpadmin*.
—m*model*	selects a model interface program for *P*. *Model* is one of the model interface names supplied with the LP Spooling Utilities (see *Models* below).
—r*class*	removes printer *P* from the specified *class*. If *P* is the last member of the *class*, then the *class* will be removed.
—v*device*	associates a new *device* with printer *P*. *Device* is the pathname of a file that is writable by *lp*. Note that the same *device* can be associated with more than one *printer*. If only the **—p** and **—v** *options* are supplied, then *lpadmin* may be used while the scheduler is running.

Restrictions.

When creating a new printer, the **—v** option and one of the **—e**, **—i** or **—m** options must be supplied. Only one of the **—e**, **—i** or **—m** options may be supplied. The **—h** and **—l** keyletters are mutually exclusive. Printer and class names may be no longer than 14 characters and must consist entirely of the characters **A-Z**, **a-z**, **0-9** and _(underscore).

Models.

Model printer interface programs are supplied with the LP Spooling Utilities. They are shell procedures which interface between *lpsched* and devices. All models reside in the directory **/usr/spool/lp/model** and may be used as is with *lpadmin* **—m**. Copies of model interface programs may also be modified and then associated with printers using *lpadmin* **—i**. The following describes the *models* which may be given on the *lp* command line using the **—o** keyletter:

LQP-40
: Letter quality printer using XON/XOFF protocol at 9600 baud.

DQP-10
: Dot matrix draft quality printer using XON/XOFF protocol at 9600 baud.

EXAMPLES

1. For a DQP-10 printer named cl8, it will use the DQP-10 model interface after the command:

```
/usr/lib/lpadmin —pcl8 —mdqp10
```

2. A LQP-40 printer called pr1 can be added to the lp configuration with the command:

```
/usr/lib/lpadmin —ppr1 —v/dev/contty —mlqp40
```

FILES

/usr/spool/lp/*

SEE ALSO

accept(1M), lpsched(1M).
enable(1), lp(1), lpstat(1) in the *User's Reference Manual.*

NAME

lpsched, lpshut, lpmove — start/stop the LP scheduler and move requests

SYNOPSIS

/usr/lib/lpsched
/usr/lib/lpshut
/usr/lib/lpmove requests dest
/usr/lib/lpmove dest1 dest2

DESCRIPTION

Lpsched schedules requests taken by *lp*(1) for printing on line printers (LP's).

Lpshut shuts down the line printer scheduler. All printers that are printing at the time *lpshut* is invoked will stop printing. Requests that were printing at the time a printer was shut down will be reprinted in their entirety after *lpsched* is started again.

Lpmove moves requests that were queued by *lp*(1) between LP destinations. This command may be used only when *lpsched* is not running.

The first form of the command moves the named *requests* to the LP destination, *dest*. *Requests* are request ids as returned by *lp*(1). The second form moves all requests for destination *dest1* to destination *dest2*. As a side effect, *lp (1)* will reject requests for *dest1*.

Note that *lpmove* never checks the acceptance status (see *accept*(1M)) for the new destination when moving requests.

FILES

/usr/spool/lp/*

SEE ALSO

accept(1M), lpadmin(1M).
enable(1), lp(1), lpstat(1) in the *User's Reference Manual.*

NAME

makefsys — create a file system on a diskette

SYNOPSIS

makefsys

DESCRIPTION

This command allows the user to create a file system on a diskette. It also writes an internal label in the file system super-block.

The user is asked some questions before the file system is created. Once created, the diskette is self-identifying.

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

sysadm makefsys

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

SEE ALSO

checkfsys(1M), labelit(1M), mkfs(1M), mountfsys(1M).
sysadm(1) in the *User's Reference Manual.*

NAME

mkfs — construct a file system

SYNOPSIS

/etc/mkfs special blocks[:i-nodes] [gap blocks/cyl]
/etc/mkfs special proto [gap blocks/cyl]

DESCRIPTION

Mkfs constructs a file system by writing on the *special* file using the values found in the remaining arguments of the command line. The command waits 10 seconds before starting to construct the file system. During this 10-second pause the command can be aborted by entering a delete (DEL).

If the second argument is a string of digits, the size of the file system is the value of *blocks* interpreted as a decimal number. This is the number of *physical* (512 byte) disk blocks the file system will occupy. If the number of i-nodes is not given, the default is the number of *logical* (1024 byte) blocks divided by 4. *Mkfs* builds a file system with a single empty directory on it. The boot program block (block zero) is left uninitialized.

If the second argument is the name of a file that can be opened, *mkfs* assumes it to be a prototype file *proto*, and will take its directions from that file. The prototype file contains tokens separated by spaces or new-lines. A sample prototype specification follows (line numbers have been added to aid in the explanation):

```
1.   /stand/diskboot
2.   4872 110
3.   d——777 3 1
4.   usr   d——777 3 1
5.         sh    ———755 3 1 /bin/sh
6.         ken   d——755 6 1
7.               $
8.         b0    b——644 3 1 0 0
9.         c0    c——644 3 1 0 0
10.        $
11.  $
```

Line 1 in the example is the name of a file to be copied onto block zero as the bootstrap program.

Line 2 specifies the number of *physical* (512 byte) blocks the file system is to occupy and the number of i-nodes in the file system.

Lines 3-9 tell *mkfs* about files and directories to be included in this file system.

Line 3 specifies the root directory.

lines 4-6 and 8-9 specifies other directories and files.

The $ on line 7 tells *mkfs* to end the branch of the file system it is on, and continue from the next higher directory. The $ on lines 10 and 11 end the process, since no additional specifications follow.

File specifications give the mode, the user ID, the group ID, and the initial contents of the file. Valid syntax for the contents field depends on the first character of the mode.

The mode for a file is specified by a 6-character string. The first character specifies the type of the file. The character range is **—bcd** to specify regular, block special, character special and directory files respectively. The second character of the mode is either **u** or — to specify set-user-ID mode or not. The third is **g** or — for the set-group-ID mode. The rest of the mode is a 3 digit octal number giving the owner, group, and other read, write, execute permissions (see *chmod*(1)).

Two decimal number tokens come after the mode; they specify the user and group IDs of the owner of the file.

If the file is a regular file, the next token of the specification may be a path name whence the contents and size are copied. If the file is a block or character special file, two decimal numbers follow which give the major and minor device numbers. If the file is a directory, *mkfs* makes the entries **.** and **..** and then reads a list of names and (recursively) file specifications for the entries in the directory. As noted above, the scan is terminated with the token **$**.

The final argument in both forms of the command specifies the rotational *gap* and the number of *blocks/cyl*. The following values are recommended for the devices available on the SYS32™/30:

Device	Gap Size	Blks/Cyl	
43M Hard Disk	17	136	(Quantum)
140M Hard Disk	16	240	(Maxtor XT-114)
Floppy Disk	1	8	

If the *gap* and *blocks/cyl* are not specified or are considered illegal values a default value of gap size 7 and 400 blocks/cyl is used.

FILES

/etc/vtoc/*

SEE ALSO

chmod(1) in the *User's Reference Manual*.
dir(4), fs(4) in the *Programmer's Reference Manual*.

BUGS

With a prototype file, it is not possible to copy in a file larger than 64 Kbytes, nor is there a way to specify links. The maximum number of i-nodes configurable is 65500.

NAME

mknod — build special file

SYNOPSIS

/etc/mknod name **b** | **c** major minor
/etc/mknod name **p**

DESCRIPTION

Mknod makes a directory entry and corresponding i-node for a special file.

The first argument is the *name* of the entry. The GENIX V.3 System convention is to keep such files in the /dev directory.

In the first case, the second argument is **b** if the special file is block-type (disks, tape) or **c** if it is character-type (other devices). The last two arguments are numbers specifying the *major* device type and the *minor* device (*e.g.*, unit, drive, or line number). They may be either decimal or octal. The assignment of major device numbers is specific to each system. The information is contained in the system source file **conf.c**. You must be the super-user to use this form of the command.

The second case is the form of the *mknod* that is used to create FIFO's (a.k.a named pipes).

WARNING

If **mknod** is used to create a device in a remote directory (Remote File Sharing), the major and minor device numbers are interpreted by the server.

SEE ALSO

mknod(2) in the *Programmer's Reference Manual.*

NAME

mount, umount — mount and unmount file systems and remote resources

SYNOPSIS

/etc/mount [[-r] [-f *fstyp*] *special directory*]
/etc/mount [[-r] [-d] *resource directory*]
/etc/umount *special*
/etc/umount [-d] *resource*

DESCRIPTION

File systems other than **root** (/) are considered *removable* in the sense that they can be either available to users or unavailable. *Mount* announces to the system that *special*, a block special device or *resource*, a remote resource, is available to users from the mount point *directory*. *Directory* must exist already; it becomes the name of the root of the newly mounted *special*.

Mount, when entered with arguments, adds an entry to the table of mounted devices, **/etc/mnttab**. *Umount* removes the entry. If invoked with no arguments, *mount* prints the entire mount table. If invoked with an incomplete argument list, *Mount* searches **/etc/fstab** for the missing arguments.

The following options are available:

-r	indicates that *special* or *resource* is to be mounted read-only. If *special* or *resource* is write-protected, this flag must be used.
-d	indicates that *resource* is a remote resource that is to be mounted on *directory* or unmounted. To mount a remote resource, Remote File Sharing must be up and running and the resource must be advertised by a remote computer [see **rfstart**(1M) and **adv**(1M)]. If **-d** is not used, *special* must be a local block special device.
-f *fstyp*	indicates that *fstyp* is the file system type to be mounted. If this argument is omitted, it defaults to the **root** *fstyp*.
special	indicates the block special device that is to be mounted on *directory*.
resource	indicates the remote resource name that is to be mounted on a directory.
directory	indicates the directory mount point for *special* or *resource*. (The directory must already exist.)

Umount announces to the system that the file system previously mounted *special* or *resource* is to be made unavailable. If invoked with an incomplete argument list, *umount* searches **/etc/fstab** for the missing arguments.

Mount can be used by any user to list mounted file systems and resources. Only a superuser can mount and unmount file systems.

FILES

/etc/mnttab	mount table
/etc/fstab	file system table

SEE ALSO

adv(1M), fuser(1M), rfstart(1M), setmnt(1M), unadv(1M).
mount(2), umount(2), fstab(4), mnttab(4) in the *Programmer's Reference Manual*.
"Remote File Sharing" chapter, *Administrator's Guide* for guidelines when mounting remote resources.

DIAGNOSTICS

If the *mount*(2) system call fails, *mount* prints an appropriate diagnostic. *Mount* issues a warning if the file system to be mounted is currently mounted under another name. A remote resource mount will fail if the resource is not available or if Remote File Sharing is not running.

Umount fails if *special* or *resource* is not mounted or if it is busy. *Special* or *resource* is busy if it contains an open file or some user's working directory. In such a case, you can use *fuser*(1M) to list and kill processes that are using *special* or *resource.*

WARNINGS

Physically removing a mounted file system diskette from the diskette drive before issuing the *umount* command damages the file system.

NAME

mountall, umountall — mount, unmount multiple file systems

SYNOPSIS

/etc/mountall [—] [file-system-table] ...
/etc/umountall [—k]

DESCRIPTION

These commands may be executed only by the superuser.

Mountall is used to mount file systems according to a *file-system-table.* (**/etc/fstab** is the default file system table.) The special file name "—" reads from the standard input.

Before each file system is mounted, it is checked using *fsstat*(1M) to see if it appears mountable. If the file system does not appear mountable, it is checked, using *fsck*(1M), before the mount is attempted.

Umountall causes all mounted file systems except **root** to be unmounted. The **—k** option sends a SIGKILL signal, via *fuser*(1M), to processes that have files open.

FILES

File-system-table format:

column 1	block special file name of file system
column 2	mount-point directory
column 3	"—r" if to be mounted read-only; "—d" if remote
column 4	(optional) file system type string
column 5+	ignored

White-space separates columns. Lines beginning with "#" are comments. Empty lines are ignored.

A typical file-system-table might read:

```
/dev/dsk/c0d0s2    /usr -r S51K
```

SEE ALSO

fsck(1M), fsstat(1M), fuser(1M), mount(1M).
sysadm(1) in the *User's Reference Manual.*
signal(2), fstab(4) in the *Programmer's Reference Manual.*

DIAGNOSTICS

No messages are printed if the file systems are mountable and clean.

Error and warning messages come from *fsck*(1M), *fsstat*(1M), and *mount*(1M).

NAME

mountfsys, umountfsys — mount, unmount a diskette file system

SYNOPSIS

mountfsys [**—y**] [**—r**]
umountfsys [**—y**]

DESCRIPTION

The *mountfsys* command mounts a file system that is on a removable disk so that users can read and write on it. The options provide the following:

—r the file system is mounted read-only.

—y suppresses any questions asked during mounting or unmounting.

The *umountfsys* command unmounts the file system.

By default, the name of the file system is displayed and the user is asked if it should be mounted. The optional **—y** argument suppresses questions and mounts or unmounts the file system immediately.

The identical functions are available under the *sysadm* menu:

sysadm mountfsys
sysadm umountfsys

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

SEE ALSO

checkfsys(1M), mount(1M), makefsys(1M).
sysadm(1) in the *User's Reference Manual.*

WARNING

ONCE THE DISK IS MOUNTED IT MUST NOT BE REMOVED FROM THE DISK DRIVE UNTIL IT HAS BEEN UNMOUNTED!

Removing the disk while it is still mounted can cause severe damage to the data on the disk.

BUGS

A file system that has no label cannot be mounted with the *mountfsys* command.

NAME

mvdir — move a directory

SYNOPSIS

/etc/mvdir dirname name

DESCRIPTION

Mvdir moves directories within a file system. *Dirname* must be a directory. If *name* does not exist, it will be created as a directory. If *name* does exist, *dirname* will be created as *name/dirname*. *Dirname* and *name* may not be on the same path; that is, one may not be subordinate to the other. For example:

```
mvdir x/y x/z
```

is legal, but

```
mvdir x/y x/y/z
```

is not.

SEE ALSO

mkdir(1), mv(1) in the *User's Reference Manual.*

WARNINGS

Only the superuser can use *mvdir*.

NAME

ncheck — generate path names from i-numbers

SYNOPSIS

/etc/ncheck [**—i** i-numbers] [**—a**] [**—s**] [file-system]

DESCRIPTION

Ncheck with no arguments generates a path-name vs. i-number list of all files on a set of default file systems (see */etc/checklist*). Names of directory files are followed by **/.**.

The options are as follows:

—i limits the report to only those files whose i-numbers follow.

—a allows printing of the names **.** and **..**, which are ordinarily suppressed.

—s limits the report to special files and files with set-user-ID mode. This option may be used to detect violations of security policy.

File system must be specified by the file system's special file.

The report should be sorted so that it is more useful.

SEE ALSO

fsck(1M).
sort(1) in the *User's Reference Manual.*

DIAGNOSTICS

If the file system structure is not consistent, **??** denotes the "parent" of a parentless file and a path-name beginning with **...** denotes a loop.

NAME

newgrp — log in to a new group

SYNOPSIS

newgrp [—] [group]

DESCRIPTION

Newgrp changes a user's group identification. The user remains logged in and the current directory is unchanged, but calculations of access permissions to files are performed with respect to the new real and effective group IDs. The user is always given a new shell, replacing the current shell, by *newgrp*, regardless of whether it terminated successfully or due to an error condition (*i.e.*, unknown group).

Exported variables retain their values after invoking *newgrp*; however, all unexported variables are either reset to their default value or set to null. System variables (such as PS1, PS2, PATH, MAIL, and HOME), unless exported by the system or explicitly exported by the user, are reset to default values. For example, a user has a primary prompt string (PS1) other than **$** (default) and has not exported PS1. After an invocation of *newgrp*, successful or not, their PS1 will now be set to the default prompt string **$**. Note that the shell command *export* (see *sh*(1)) is the method to export variables so that they retain their assigned value when invoking new shells.

With no arguments, *newgrp* changes the group identification back to the group specified in the user's password file entry. This is a way to exit the effect of an earlier *newgrp* command.

If the first argument to *newgrp* is a —, the environment is changed to what would be expected if the user actually logged in again as a member of the new group.

A password is demanded if the group has a password and the user does not, or if the group has a password and the user is not listed in **/etc/group** as being a member of that group.

FILES

/etc/group	system's group file
/etc/passwd	system's password file

SEE ALSO

login(1), sh(1) in the *User's Reference Manual.*
group(4), passwd(4), environ(5) in the *Programmer's Reference Manual.*

BUGS

There is no convenient way to enter a password into **/etc/group.** Use of group passwords is not encouraged, because, by their very nature, they encourage poor security practices. Group passwords may disappear in the future.

NAME

nlsadmin — network listener service administration

SYNOPSIS

nlsadmin —x
nlsadmin [options] net_spec

DESCRIPTION

Nlsadmin administers the network listener process(es) on a machine. Each network has a separate instance of the network listener process associated with it; each instance (and thus, each network) is configured separately. The listener process "listens" to the network for service requests, accepts requests when they arrive, and spawns servers in response to those service requests. The network listener process will work with any network (more precisely, with any transport provider) that conforms to the transport provider specification.

The listener supports two classes of service: a general listener service, serving processes on remote machines, and a terminal login service, for terminals connected directly to a network. The terminal login service provides networked access to this machine in a form suitable for terminals connected directly to the network. However, this direct terminal service requires special associated software, and is only available with some networks (for example, the AT&T STARLAN network).

Nlsadmin can establish a listener process for a given network, configure the specific attributes of that listener, and start and kill the listener process for that network. *Nlsadmin* can also report on the listener processes on a machine, either individually (per network) or collectively.

The following list shows how to use *nlsadmin.* In this list, *net_spec* is a particular listener process. Specifically, *net_spec* is the relative path name of the entry under **/dev** for a given network (that is, a transport provider). Changing the list of services provided by the listener produces immediate changes, while changing an address on which the listener listens has no effect until the listener is restarted. The following combination of options can be used.

nlsadmin will give a brief usage message.

nlsadmin —x will report the status of all of the listener processes installed on this machine.

nlsadmin *net_spec* will print the status of the listener process for *net_spec.*

nlsadmin —q *net_spec*
will query the status of the listener process for the specified network, and will reflect the result of that query in its exit code. If a listener process is active, *nlsadmin* will exit with a status of 0; if no process is active, the exit code will be 1; while the exit code will be greater than 1 in case of error.

nlsadmin —v *net_spec*
will print a verbose report on the servers associated with *net_spec,* giving the service code, status, command, and comment for each.

nlsadmin —z *code net_spec*
will print a report on the server associated with *net_spec* that has service code *code,* giving the same information as in the **—v** option.

nlsadmin —q —z *code net_spec*
will query the status of the service with service code *code* on network *net_spec,* and will exit with a status of 0 if that service is enabled, 1 if that service is disabled, and greater than 1 in case of error.

nlsadmin —l *addr net_spec*

will change or set the address on which the listener listens (the general listener service). This is the address generally used by remote processes to access the servers available through this listener (see the **—a** option, below). *Addr* is the transport address on which to listen and is interpreted using a syntax that allows for a variety of address formats. By default *addr* is interpreted as the symbolic ASCII representation of the transport address. An *addr* preceded by a **\x** will let you enter an address in hexadecimal notation. Note that *addr* must appear as a single word to the shell and must be quoted if it contains any blanks.

If *addr* is just a dash ("—"), *nlsadmin* will report the address currently configured, instead of changing it.

A change of address will not take effect until the next time the listener for that network is started.

nlsadmin —t *addr net_spec*

will change or set the address on which the listener listens for requests for terminal service, but is otherwise similar to the **—l** option above. A terminal service address should not be defined unless the appropriate remote login software is available; if such software is available, it must be configured as service code 1 (see the **—a** option, below).

nlsadmin —i *net_spec*

will initialize or change a listener process for the network specified by *net_spec*, that is, it will create and initialize the files required by the listener. Note that the listener should only be initialized once for a given network, and that doing so does not actually invoke the listener for that network. The listener must be initialized before assigning addressing or services.

nlsadmin [—m] —a *service_code [***—p** *modules]* **—c** *cmd* **—y** *comment net_spec*

will add a new service to the list of services available through the indicated listener. *Service_code* is the code for the service, *cmd* is the command to be invoked in response to that service code, comprised of the full path name of the server and its arguments, and *comment* is a brief (free-form) description of the service for use in various reports. Note that *cmd* must appear as a single word to the shell and must be quoted if it contains arguments for the server. Similarly, the *comment* must also appear as a single word to the shell. When a service is added, it is initially enabled (see the **—e** and **—d** options, below).

If the **—m** option is specified, the entry will be marked as an administrative entry. Service codes 1 through 100 are reserved for administrative entries, which are those that require special handling internally. In particular, code 1 is assigned to the remote login service, which is the service automatically invoked for connections to the terminal login address.

If the **—p** option is specified, then *modules* will be interpreted as a list of STREAMS modules for the listener to push before starting the service being added. The modules are pushed in the order they are specified. *modules* should be a comma-separated list of modules, with no white space included. This option is only available with the GENIX V.3 Release.

A service must explicitly be added to the listener for each network on which that service is to be available. This operation will normally be performed only when the service is installed on a machine, or when populating the list of services for a new network.

nlsadmin —r *service_code net_spec*
: will remove the entry for the *service_code* from that listener's list of services. This will normally be performed only in conjunction with the de-installation of a service from a machine.

nlsadmin —e *service_code net_spec*
nlsadmin —d *service_code net_spec*
: will enable or disable (respectively) the service indicated by *service_code* for the specified network. The service must have previously been added to the listener for that network (see the **—a** option, above). Disabling a service will cause subsequent service requests for that service to be denied, but the processes from any prior service requests that are still running will continue unaffected.

nlsadmin —s *net_spec*
nlsadmin —k *net_spec*
: will start and kill (respectively) the listener process for the indicated network. These operations will normally be performed as part of the system startup and shutdown procedures. Before a listener can be started for a particular network, it must first have been initialized, and an address must be defined for the general listener service (see the **—i** and **—l** options, above). When a listener is killed, processes that are still running as a result of prior service requests will continue unaffected.

The listener runs under its own ID of *listen*, with group ID *adm*. This ID must be entered in the system password file **/etc/passwd**; the **HOME** directory listed for that ID will be concatenated with *net_spec* to determine the location of the listener configuration information for each network.

Nlsadmin may be invoked by any user to generate reports, but all operations that affect a listener's status or configuration are restricted to the super-user.

FILES

/usr/net/nls/*net_spec*

SEE ALSO

Network Programmer's Guide

NAME

nsquery — Remote File Sharing name server query

SYNOPSIS

nsquery [-h] [name]

DESCRIPTION

Nsquery provides information about resources available to the host from both the local domain and from other domains. All resources are reported, regardless of whether the host is authorized to access them. When used with no options, *nsquery* identifies all resources in the domain that have been advertised as sharable. A report on selected resources can be obtained by specifying *name*, where *name* is:

nodename	The report will include only those resources available from *nodename*.
domain.	The report will include only those resources available from *domain*.
domain.nodename	The report will include only those resources available from *domain.nodename*.

When the name does not include the delimiter ".", it will be interpreted as a *nodename* within the local domain. If the name ends with a delimiter ".", it will be interpreted as a domain name.

The information contained in the report on each resource includes its advertised name (domain.resource), the read/write permissions, the server (nodename.domain) that advertised the resource, and a brief textual description.

When *—h* is used, the header is not printed.

A remote domain must be listed in your **rfmaster** file in order to query that domain.

EXIT STATUS

If no entries are found when *nsquery* is executed, the report header is printed.

ERRORS

If your host cannot contact the domain name server, an error message will be sent to standard error.

SEE ALSO

adv(1M), unadv(1M).
rfmaster(4) in the *Programmer's Reference Manual.*

NAME

powerdown — stop all processes and turn off the power

SYNOPSIS

powerdown [**—y** | **—Y**]

DESCRIPTION

This command brings the system to a state where nothing is running and then turns off the power.

By default, the user is asked questions that control how much warning the other users are given. The options:

—y prevents the questions from being asked and just gives the warning messages. There is a 60 second pause between the warning messages. Note that pressing the standby button on the side of the cabinet will accomplish the same thing.

—Y is the same as **—y** except it has no pause between messages. It is the fastest way to bring the system down.

The identical function is also available under the *sysadm* command:

sysadm powerdown

Password control can be instituted on this command. See *sysadm*(1), **admpasswd** subcommand.

EXAMPLES

```
some-long-running-command; powerdown —y
```

The first command is run to completion and then the machine turns off. This is useful for, say, formatting a document to the printer at the end of a day.

FILES

/etc/shutdown - invoked by powerdown

SEE ALSO

shutdown(1M).
sysadm(1) in the *User's Reference Manual.*

NAME

profiler: prfld, prfstat, prfdc, prfsnap, prfpr — GENIX V.3 system profiler

SYNOPSIS

/etc/prfld [system_namelist]
/etc/prfstat on
/etc/prfstat off
/etc/prfdc file [period [off_hour]]
/etc/prfsnap file
/etc/prfpr file [cutoff [system_namelist]]

DESCRIPTION

Prfld, *prfstat*, *prfdc*, *prfsnap*, and *prfpr* form a system of programs to facilitate an activity study of the GENIX V.3 operating system.

Prfld is used to initialize the recording mechanism in the system. It generates a table containing the starting address of each system subroutine as extracted from *system_namelist*.

Prfstat is used to enable or disable the sampling mechanism. Profiler overhead is less than 1% as calculated for 500 text addresses. *Prfstat* will also reveal the number of text addresses being measured.

Prfdc and *prfsnap* perform the data collection function of the profiler by copying the current value of all the text address counters to a file where the data can be analyzed. *Prfdc* will store the counters into *file* every *period* minutes and will turn off at *off_hour* (valid values for *off_hour* are **0—24**). *Prfsnap* collects data at the time of invocation only, appending the counter values to *file*.

Prfpr formats the data collected by *prfdc* or *prfsnap*. Each text address is converted to the nearest text symbol (as found in *system_namelist*) and is printed if the percent activity for that range is greater than *cutoff*.

FILES

/dev/prf	interface to profile data and text addresses
/unix	default for system namelist file

NAME

pwck, grpck — password/group file checkers

SYNOPSIS

/etc/pwck [file]
/etc/grpck [file]

DESCRIPTION

Pwck scans the password file and notes any inconsistencies. The checks include validation of the number of fields, login name, user ID, group ID, and whether the login directory and the program-to-use-as-Shell exist. The default password file is **/etc/passwd.**

Grpck verifies all entries in the group file. This verification includes a check of the number of fields, group name, group ID, and whether all login names appear in the password file. The default group file is **/etc/group.**

FILES

/etc/group
/etc/passwd

SEE ALSO

group(4), passwd(4) in the *Programmer's Reference Manual.*

DIAGNOSTICS

Group entries in **/etc/group** with no login names are flagged.

NAME

rc0 — run commands performed to stop the operating system

SYNOPSIS

/etc/rc0

DESCRIPTION

This file is executed at each system state change that needs to have the system in an inactive state. It is responsible for those actions that bring the system to a quiescent state, traditionally called "shutdown."

There are three system states that require this procedure. They are state 0 (the system halt state), state 5 (the firmware state), and state 6 (the reboot state). Whenever a change to one of these states occurs, the */etc/rc0* procedure is run. The entry in **/etc/inittab** might read:

```
s0:056:wait:/etc/rc0 >/dev/console 2>&1 </dev/console
```

Some of the actions performed by */etc/rc0* are carried out by files in the directory **/etc/shutdown.d.** and files beginning with **K** in **/etc/rc0.d.** These files are executed in ascii order (see FILES below for more information), terminating some system service. The combination of commands in */etc/rc0* and files in **/etc/shutdown.d** and **/etc/rc0.d** determines how the system is shut down.

The recommended sequence for */etc/rc0* is:

Stop System Services and Daemons.

> Various system services (such as networking or LP Spooler) are gracefully terminated.
>
> When new services are added that should be terminated when the system is shut down, the appropriate files are installed in **/etc/shutdown.d** and **/etc/rc0.d.**

Terminate Processes

> SIGTERM signals are sent to all running processes by *killall*(1M). Processes stop themselves cleanly if sent SIGTERM.

Kill Processes

> SIGKILL signals are sent to all remaining processes; no process can resist SIGKILL.
>
> At this point the only processes left are those associated with */etc/rc0* and processes 0 and 1, which are special to the operating system.

Unmount All File Systems

> Only the root file system (/) remains mounted.

Depending on which system state the systems end up in (0, 5, or 6), the entries in **/etc/inittab** will direct what happens next. If the **/etc/inittab** has not defined any other actions to be performed as in the case of system state 0, then the operating system will have nothing to do. It should not be possible to get the system's attention. The only thing that can be done is to turn off the power or possibly get the attention of a firmware monitor. The command can be used only by the super-user.

FILES

The execution by **/bin/sh** of any files in **/etc/shutdown.d** occurs in ascii sort-sequence order. See rc2(1M) for more information.

SEE ALSO

killall(1M), rc2(1M), shutdown(1M).

NAME

rc2 — run commands performed for multi-user environment

SYNOPSIS

/etc/rc2

DESCRIPTION

This file is executed via an entry in **/etc/inittab** and is responsible for those initializations that bring the system to a ready-to-use state, traditionally state 2, called the "multi-user" state.

The actions performed by **/etc/rc2** are found in files in the directory **/etc/rc.d** and files beginning with S in **/etc/rc2.d.** These files are executed by **/bin/sh** in ascii sort—sequence order (see FILES for more information). When functions are added that need to be initialized when the system goes multi-user, an appropriate file should be added in **/etc/rc2.d.**

The functions done by **/etc/rc2** command and associated **/etc/rc2.d** files include:

Setting and exporting the TIMEZONE variable.

Setting-up and mounting the user (**/usr**) file system.

Cleaning up (remaking) the **/tmp** and **/usr/tmp** directories.

Loading the network interface and ports cards with program data and starting the associated processes.

Starting the *cron* daemon by executing **/etc/cron.**

Cleaning up (deleting) uucp locks status, and temporary files in the **/usr/spool/uucp** directory.

Other functions can be added, as required, to support the addition of hardware and software features.

EXAMPLES

The following are prototypical files found in **/etc/rc2.d.** These files are prefixed by an **S** and a number indicating the execution order of the files.

MOUNTFILESYS

```
#   Set up and mount file systems

cd /
/etc/mountall /etc/fstab
```

RMTMPFILES

```
# clean up /tmp
rm —rf /tmp
mkdir /tmp
chmod 777 /tmp
chgrp sys /tmp
chown sys /tmp
```

uucp

```
#   clean-up uucp locks, status, and temporary files

rm —rf /usr/spool/locks/*
```

The file **/etc/TIMEZONE** is included early in */etc/rc2*, thus establishing the default time zone for all commands that follow.

FILES

Here are some hints about files in **/etc/rc.d**:

The order in which files are executed is important. Since they are executed in ascii sort—sequence order, using the first character of the file name as a sequence indicator will help keep the proper order. Thus, files starting with the following characters would be:

[0-9].	very early
[A-Z].	early
[a-n].	later
[o-z].	last

3.mountfs

Files in **/etc/rc.d** that begin with a dot (.) will not be executed. This feature can be used to hide files that are not to be executed for the time being without removing them. The command can be used only by the super-user.

Files in **/etc/rc2.d** must begin with an **S** or a **K** followed by a number and the rest of the file name. Upon entering run level 2, files beginning with **S** are executed with the **start** option; files beginning with **K**, are executed with the **stop** option. Files beginning with other characters are ignored.

SEE ALSO

shutdown(1M).

NAME

rfadmin — Remote File Sharing domain administration

SYNOPSIS

rfadmin

rfadmin -a hostname

rfadmin -r hostname

rfadmin -p

DESCRIPTION

Rfadmin is used to add and remove hosts and their associated authentication information from a *domain/*__passwd__ file on a Remote File Sharing primary domain name server. It is also used to transfer domain name server responsibilities from one machine to another. Used with no options, *rfadmin* returns the *hostname* of the current domain name server for the local domain.

Rfadmin can only be used to modify domain files on the primary domain name server (**—a** and **—r** options). If domain name server reponsibilities are temporarily passed to a secondary domain name server, that computer can use the **—p** option to pass domain name server responsibility back to the primary. Any host can use *rfadmin* with no options to print information about the domain. The user must have **root** permissions to use the command.

-a *hostname*	Used to add a host to a domain that is served by this domain name server. *hostname* must be of the form *domain.nodename.* It creates an entry for *hostname* in the *domain/*passwd file, which has the same format as */etc/passwd,* and prompts for an initial authentication password; the password prompting process conforms with that of *passwd*(1).
-r *hostname*	Used to remove a host from its domain by removing it from the *domain/*passwd file.
-p	Used to pass the domain name server responsibilities back to a primary or to a secondary name server.

ERRORS

When used with the *-a* option, if *hostname* is not unique in the domain.

When used with the *-r* option, if (1) *hostname* does not exist in the domain, (2) *hostname* is defined as a domain name server, or (3) there are resources advertised by *hostname,* an error message will be sent to standard error.

When used with the *-p* option to change the domain name server, if there are no backup name servers defined for *domain,* an error message will be sent to standard error.

FILES

/usr/nserve/auth.info/*domain*/passwd
(For each *domain,* this file: is created on the primary, should be copied to all secondaries, and should be copied to all hosts that want to do password verification of hosts in the *domain.*)

SEE ALSO

passwd(1), rfstart(1M), rfstop(1M), umount(1M).

NAME

rfpasswd — change Remote File Sharing host password

SYNOPSIS

rfpasswd

DESCRIPTION

Rfpasswd updates the Remote File Sharing authentication password for a host; processing of the new password follows the same criteria as *passwd*(1). The updated password is registered at the domain name server (/usr/nserve/auth.info/*domain*/passwd) and replaces the password stored at the local host (**/usr/nserve/loc.passwd** file).

This command is restricted to the super-user.

NOTE: If you change your host password, make sure that hosts that validate your password are notified of this change. To receive the new password, hosts must obtain a copy of the *domain*/**passwd** file from the domain's primary name server. If this is not done, attempts to mount remote resources may fail!

ERRORS

If (1) the old password entered from this command does not match the existing password for this machine, (2) the two new passwords entered from this command do not match, (3) the new password does not satisfy the security criteria in *passwd*(1), (4) the domain name server does not know about this machine, or (5) the command is not run with super-user privileges, an error message will be sent to standard error. Also, Remote File Sharing must be running on your host and your domain's primary name server. A new password cannot be logged if a secondary is acting as the domain name server.

FILES

/usr/nserve/auth.info/*domain*/passwd
/usr/nserve/loc.passwd

SEE ALSO

passwd(1), rfstart(1M), rfadmin(1M).

NAME

rfstart — start Remote File Sharing

SYNOPSIS

Rfstart [-v] [**-p** primary_addr]

DESCRIPTION

Rfstart starts Remote File Sharing and defines an authentication level for incoming requests. (This command can only be used after the domain name server is set up and your computer's domain name and network specification has been defined using *dname*(1M).)

-v Specifies that verification of all clients is required in response to initial incoming mount requests; any host not in the file **/usr/nserve/auth.info/***domain*/passwd for the **domain** they belong to, will not be allowed to mount resources from your host. If **-v** is not specified, hosts named in *domain*/passwd will be verified, other hosts will be allowed to connect without verification.

-p *primary_addr*
Indicates the primary domain name server for your domain. *Primary_addr* must be the network address of the primary name server for your domain. If the **-p** option is not specified, the address of the domain name server is taken from the **rfmaster** file. (See **rfmaster**(1M) for a description of the valid address syntax.)

If the host password has not been set, *rfstart* will prompt for a password; the password prompting process must match the password entered for your machine at the primary domain name server (see *rfadmin*(1M)). If you remove the **loc.passwd** file or change domains, you will also have to reenter the password.

Also, when *rfstart* is run on a domain name server, entries in the **rfmaster**(4) file are syntactically validated.

This command is restricted to the super-user.

ERRORS

If syntax errors are found in validating the **rfmaster**(4) file, a warning describing each error will be sent to standard error.

If (1) the shared resource environment is already running, (2) there is no communications network, (3) the domain name server cannot be found, (4) the domain name server does not recognize the machine, or (5) the command is run without super-user privileges, an error message will be sent to standard error.

Remote file sharing will not start if the host password in /usr/nserve/loc.passwd is corrupted. If you suspect this has happened, remove the file and run *rfstart* again to reenter your password.

NOTE: *Rfstart* will NOT fail if your host password does not match the password on the domain name server. You will simply receive a warning message. However, if you try to mount a resource from the primary or any other host that validates your password, the mount will fail if your password does not match the one that host has listed for your machine.

FILES

/usr/nserve/rfmaster
/usr/nserve/loc.passwd

SEE ALSO

adv(1M), dname(1M), mount(1M), rfadmin(1M), rfstop(1M), unadv(1M).
rfmaster(4) in the *Programmer's Reference Manual.*

NAME

rfstop — stop the Remote File Sharing environment

SYNOPSIS

rfstop

DESCRIPTION

Rfstop disconnects a host from the Remote File Sharing environment until another *rfstart*(1M) is executed.

When executed on the domain name server, the domain name server responsibility is moved to a secondary name server as designated in the **rfmaster** file.

This command is restricted to the super-user.

ERRORS

If (1) there are resources currently advertised by this host, (2) resources from this machine are still remotely mounted by other hosts, (3) there are still remotely mounted resources in the local file system tree, (4) *rfstart*(1M) had not previously been executed, or (5) the command is not run with super-user privileges, an error message will be sent to standard error.

SEE ALSO

adv(1M), mount(1M), rfadmin(1M), rfstart(1M), unadv(1M).
rfmaster(4) in the *Programmer's Reference Manual.*

NAME

rfuadmin — Remote File Sharing notification shell script

SYNOPSIS

rfuadmin message remote_resource [seconds]

DESCRIPTION

The *rfuadmin* administrative shell script responds to unexpected Remote File Sharing events, such as broken network connections and forced unmounts, picked up by the *rfudaemon* process. This command is not intended to be run directly from the shell.

The response to messages received by *rfudaemon* can be tailored to suit the particular system by editing the *rfuadmin* script. The following paragraphs describe the arguments passed to *rfuadmin* and the responses.

disconnect *remote_resource*

A link to a remote resource has been cut. *Rfudaemon* executes *rfuadmin*, passing it the message **disconnect** and the name of the disconnected resource. *Rfuadmin* sends this message to all terminals using *wall*(1):

Remote_resource has been disconnected from the system.

Then it executes *fuser*(1M) to kill all processes using the resource, unmounts the resource [*umount*(1M)] to clean up the kernel, and starts *rmount* to try to remount the resource.

fumount *remote_resource*

A remote server machine has forced an unmount of a resource a local machine has mounted. The processing is similar to processing for a disconnect.

fuwarn *remote_resource seconds*

This message notifies *rfuadmin* that a resource is about to be unmounted. *Rfudaemon* sends this script the *fuwarn* message, the resource name, and the number of seconds in which the forced unmount will occur. *rfuadmin* sends this message to all terminals:

Remote_resource is being removed from the system in # seconds.

SEE ALSO

fumount(1M), rmount(1M), rfudaemon(1M), rfstart(1M).
wall(1) in the *User's Reference Manual.*

BUGS

The console must be on when Remote File Sharing is running. If it's not, *rfuadmin* will hang when it tries to write to the console (*wall*) and recovery from disconected resources will not complete.

NAME

rfudaemon — Remote File Sharing daemon process

SYNOPSIS

rfudaemon

DESCRIPTION

The *rfudaemon* command is started automatically by *rfstart*(1M) and runs as a daemon process as long as Remote File Sharing is active. Its function is to listen for unexpected events, such as broken network connections and forced unmounts, and execute appropriate administrative procedures.

When such an event occurs, *rfudaemon* executes the administrative shell script *rfuadmin*, with arguments that identify the event. This command is not intended to be run from the shell. Here are the events:

DISCONNECT
: A link to a remote resource has been cut. *Rfudaemon* executes *rfuadmin*, with two arguments: *disconnect* and the name of the disconnected resource.

FUMOUNT
: A remote server machine has forced an unmount of a resource a local machine has mounted. *Rfudaemon* executes *rfuadmin*, with two arguments: *fumount* and the name of the disconnected resource.

GETUMSG
: A remote user-level program has sent a message to the local *rfudaemon*. Currently the only message sent is *fuwarn*, which notifies *rfuadmin* that a resource is about to be unmounted. It sends *rfuadmin* the *fuwarn*, the resource name, and the number of seconds in which the forced unmount will occur.

LASTUMSG
: The local machine wants to stop the *rfudaemon* [*rfstop*(1M)]. This causes *rfudaemon* to exit.

SEE ALSO

rfstart(1M), rfuadmin(1M).

NAME

rmntstat — display mounted resource information

SYNOPSIS

rmntstat [-h] [*resource*]

DESCRIPTION

When used with no options, *rmntstat* displays a list of all local Remote File Sharing resources that are remotely mounted, the local path name, and the corresponding clients. *rmntstat* returns the remote mount data regardless of whether a resource is currently advertised; this ensures that resources that have been unadvertised but are still remotely mounted are included in the report. When a *resource* is specified, *rmntstat* displays the remote mount information only for that resource. The **-h** option causes header information to be omitted from the display.

EXIT STATUS

If no local resources are remotely mounted, *rmntstat* will return a successful exit status.

ERRORS

If *resource* (1) does not physically reside on the local machine or (2) is an invalid resource name, an error message will be sent to standard error.

SEE ALSO

mount(1M), fumount(1M), unadv(1M).

NAME

rmount — retry remote resource mounts

SYNOPSIS

/etc/rmount -d[r] special directory

DESCRIPTION

Rmount is an administrative shell script that tries to mount remote resource *special* on *directory*. If the remote mount is unsuccessful, *rmount* will wait 60 seconds and try to mount the resource again. This will repeat forever. The RETRIES=0 value in the shell script can be changed to limit the number of times the shell script will try to mount a remote resource. The wait time (TIME=60) can also be changed.

See *mount*(1M) for a description of the options.

FILES

/etc/mnttab	mount table

SEE ALSO

fumount(1M), fuser(1M), mount(1M), rfstart(1M), rfuadmin(1M), setmnt(1M).
mnttab(4) in the *Programmer's Reference Manual.*

NAME

rmountall, rumountall — mount, unmount Remote File Sharing resources

SYNOPSIS

/etc/rmountall [—] " file-system-table " [...]
/etc/rumountall [—k]

DESCRIPTION

Rmountall is a Remote File Sharing command used to mount remote resources according to a *file-system-table*. (**/etc/fstab** is the recommended *file-system-table.*) The special file name "-" reads from the standard input.

Rumountall causes all mounted remote resources to be unmounted. The **—k** option sends a SIGKILL signal, via *fuser*(1M), to processes that have files open.

These commands may be executed only by the super-user.

The file-system-table format is as follows:

column 1	block special file name of file system
column 2	mount-point directory
column 3	—r if to be mounted read-only; —d if remote resource
column 4	file system type (not use with Remote File Sharing)
column 5+	ignored

White-space separates columns. Lines beginning with "#" are comments. Empty lines are ignored.

SEE ALSO

fuser(1M), mount(1M), rfstart(1M),
sysadm(1) in the *User's Reference Manual.*
signal(2) in the *Programmer's Reference Manual.*

DIAGNOSTICS

No messages are printed if the remote resources are mounted successfully.

Error and warning messages come from *mount*(1M).

NAME

runacct — run daily accounting

SYNOPSIS

/usr/lib/acct/runacct [mmdd [state]]

DESCRIPTION

Runacct is the main daily accounting shell procedure. It is normally initiated via *cron*(1M). *Runacct* processes connect, fee, disk, and process accounting files. It also prepares summary files for *prdaily* or billing purposes. *Runacct* is distributed only to source code licensees.

Runacct takes care not to damage active accounting files or summary files in the event of errors. It records its progress by writing descriptive diagnostic messages into **active**. When an error is detected, a message is written to **/dev/console**, mail (see *mail*(1)) is sent to **root** and **adm**, and *runacct* terminates. *Runacct* uses a series of lock files to protect against re-invocation. The files **lock** and **lock1** are used to prevent simultaneous invocation, and **lastdate** is used to prevent more than one invocation per day.

Runacct breaks its processing into separate, restartable *states* using **statefile** to remember the last *state* completed. It accomplishes this by writing the *state* name into **statefile**. *Runacct* then looks in **statefile** to see what it has done and to determine what to process next. *States* are executed in the following order:

SETUP	Move active accounting files into working files.
WTMPFIX	Verify integrity of **wtmp** file, correcting date changes if necessary.
CONNECT1	Produce connect session records in **ctmp.h** format.
CONNECT2	Convert **ctmp.h** records into **tacct.h** format.
PROCESS	Convert process accounting records into **tacct.h** format.
MERGE	Merge the connect and process accounting records.
FEES	Convert output of *chargefee* into **tacct.h** format and merge with connect and process accounting records.
DISK	Merge disk accounting records with connect, process, and fee accounting records.
MERGETACCT	Merge the daily total accounting records in **daytacct** with the summary total accounting records in **/usr/adm/acct/sum/tacct**.
CMS	Produce command summaries.
USEREXIT	Any installation-dependent accounting programs can be included here.
CLEANUP	Cleanup temporary files and exit.

To restart *runacct* after a failure, first check the **active** file for diagnostics, then fix up any corrupted data files such as **pacct** or **wtmp**. The **lock** files and **lastdate** file must be removed before *runacct* can be restarted. The argument *mmdd* is necessary if *runacct* is being restarted, and specifies the month and day for which *runacct* will rerun the accounting. Entry point for processing is based on the contents of **statefile**; to override this, include the desired *state* on the command line to designate where processing should begin.

EXAMPLES

To start *runacct*.

```
nohup runacct 2> /usr/adm/acct/nite/fd2log &
```

To restart *runacct*.

```
nohup runacct 0601 2>> /usr/adm/acct/nite/fd2log &
```

To restart *runacct* at a specific *state.*

```
nohup runacct 0601 MERGE 2>> /usr/adm/acct/nite/fd2log &
```

FILES

/etc/wtmp
/usr/adm/pacct*
/usr/src/cmd/acct/tacct.h
/usr/src/cmd/acct/ctmp.h
/usr/adm/acct/nite/active
/usr/adm/acct/nite/daytacct
/usr/adm/acct/nite/lock
/usr/adm/acct/nite/lock1
/usr/adm/acct/nite/lastdate
/usr/adm/acct/nite/statefile
/usr/adm/acct/nite/ptacct*.*mmdd*

SEE ALSO

acct(1M), acctcms(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), cron(1M), fwtmp(1M)
acctcom(1), mail(1) in the *User's Reference Manual*
acct(2), acct(4), utmp(4) in the *Programmer's Reference Manual*

BUGS

Normally it is not a good idea to restart *runacct* in the **SETUP** *state*. Run **SETUP** manually and restart via:

runacct *mmdd* **WTMPFIX**

If *runacct* failed in the **PROCESS** *state*, remove the last **ptacct** file because it will not be complete.

NAME

s5dump — incremental file system dump

SYNOPSIS

/etc/s5dump filesystem
/etc/s5dump [-keys [*argument* ...]] filesystem

DESCRIPTION

S5dump copies to floppy diskettes all files in the *filesystem*. The *key* specifies options about the dump. *Key* consists of characters from the set **fsbh94.**

Without using any *key, s5dump* will take the next argument as the *filesystem* which the user wants to dump.

The default output is */dev/rdiskette,* unless *key* **f** is used to specify a different outputing device. By default the size of the output floppy diskette is 1208 blocks (double density, double sided 5-1/4" diskette). Use *key* **b** to specify an alternative block size.

In the case of a cartridge output use *key* **f** together with s and **4** or **9** which together will determine the size of the output unit. For diskette output do not use *keys* **s, 4** and **9.**

f	Place the dump on the next *argument* instead of the default floppy diskettes. Use also **s** together with **4** or **9** for cartridge tape output.
s	The size of the dump tape is specified in feet. The number of feet is taken from the next *argument.* When *key* **s** is used without the *key* **4** or **9** the program will ask for tape drive type. Answer *yes* or *no* to the questions.
b	The size of the dump diskette or tape is specified in blocks. The number of blocks is taken from the next *argument.* Do not use **b** together with s or **4** or **9.**
4	This number specifies a 4-track cartridge tape drive. **4** is ignored if used with **b.**
9	This number specifies a 9-track cartridge tape drive. **9** is ignored if used with **b.**
h	Print *key* options and their usuage.

DIAGNOISITICS

All questions *s5dump* poses **must** be answered by typing yes or no, appropriately. *S5dump* requires operator intervention on these conditions:

cartridge tape drive type if *key* s is used, but **4** or **9** are missing,

end of diskette (tape),

end of dump,

tape write error,

tape open error,

disk read error (if there are more than a threshold of 32).

FILES

/dev/rdiskette

SEE ALSO

s5restor(1M), dump(4),

DIAGNOSTICS

Many, and verbose.

NAME

s5restor — incremental file system restore

SYNOPSIS

s5restor *disk*
s5restor [-key [*argument ...]] disk*

DESCRIPTION

S5restor is a stand-alone program which reads floppies (cartridge tapes) dumped with the *s5dump*(1M) command.

If no keys are given, the *argument* is used as the name of disk where the filesystem will be restored. *Key* consists of characters from the set **fsb94h.**

The default input device is */dev/rdiskette* (double sided, double density 5-1/4" diskettes) The default input unit size is 1280 blocks.

For an optional block size use *key* **b.**

To use an alternative tape drive use *key* **f.** If it is necessary to obtain a different input unit size, use **f** together with **s,** and **4** or **9.**

f The next *argument* is used as the name of the archive instead of the default /dev/rdiskette.

s The size of the input cartridge tape in feet. The number of feet is taken from the next *argument.* Do not use this with *key* **b.**

b The size of the input unit in blocks. The number of blocks is taken from the next *argument.* Do not use this with *key* **s.**

9 This number specifies that the input device is from a 9-track tape drive. This is used to determine the size of the input unit. *Key* **b** will ignore this number. Do not use this key when using the default diskette unit.

4 This number specifies that the input device is from a 4-track tape drive. This is used to determine the size of the input unit. *Key* **b** will ignore this number. Do not use this key when using the default diskette unit.

h Print options available and usage.

DIAGNOSTICS

If the dump extends over more than one diskette (tape), it will ask you to change to the next diskette (tape). Reply with a new line when the next volume has been mounted.

When s is used without **4** or **9,** it will ask you for the tape drive type.

Complains about bad key characters.

Complains if it gets a read error.

Doing a *mkfs* and a *s5restor* on a disk will change the size of the file system.

SEE ALSO

s5dump(1M), mkfs(1M)

NAME

sadp — disk access profiler

SYNOPSIS

sadp [**—th**] [**—d** device[—drive]] s [n]

DESCRIPTION

Sadp reports disk access location and seek distance, in tabular or histogram form. It samples disk activity once every second during an interval of *s* seconds. This is done repeatedly if *n* is specified. Cylinder usage and disk distance are recorded in units of 8 cylinders.

Valid values of *device* are **hdsk** for integral disk and **fdsk** for integral floppy. *Drive* specifies the disk drives and it may be:

> a drive number in the range supported by *device*,
> two numbers separated by a minus (indicating an inclusive range),

or

> a list of drive numbers separated by commas.

Up to 8 disk drives may be reported. The **—d** option may be omitted, if only one *device* is present.

The **—t** flag causes the data to be reported in tabular form. The **—h** flag produces a histogram on the printer of the data. Default is **—t**.

EXAMPLE

The command:

```
sadp —d hdsk—0 900 4
```

will generate 4 tabular reports, each describing cylinder usage and seek distance of **hdsk** disk drive 0 during a 15-minute interval.

FILES

/dev/kmem

SEE ALSO

mem(7).

NAME

sanityck — set/check file system sanity flag

SYNOPSIS

sanityck c dev
sanityck s dev

DESCRIPTION

Sanityck c reads the super block of the file system *dev*. If the sanity flag is 0, *sanityck* returns 1 to the shell. If the sanity flag is 1, it sets the sanity flag to 0 and returns 0. *Sanity s* sets the sanity flag of the file system *dev* to 1.

DIAGNOSTICS

Sanityck returns 2 if there are problems opening, reading, writing, or seeking within the specified file system.

WARNING

The use of this command could be hazardous to the system.

NAME

sar: sa1, sa2, sadc — system activity report package

SYNOPSIS

/usr/lib/sa/sadc [t n] [ofile]

/usr/lib/sa/sa1 [t n]

/usr/lib/sa/sa2 [**—ubdycwaqvmprSDA**] [**—s** time] [**—e** time] [**—i** sec]

DESCRIPTION

System activity data can be accessed at the special request of a user (see *sar*(1)) and automatically on a routine basis as described here. The operating system contains a number of counters that are incremented as various system actions occur. These include counters for CPU utilization, buffer usage, disk and tape I/O activity, TTY device activity, switching and system-call activity, file-access, queue activity, inter-process communications, paging and Remote File Sharing.

Sadc and shell procedures, *sa1* and *sa2*, are used to sample, save, and process this data.

Sadc, the data collector, samples system data *n* times every *t* seconds and writes in binary format to *ofile* or to standard output. If *t* and *n* are omitted, a special record is written. This facility is used at system boot time, when booting to a multiuser state, to mark the time at which the counters restart from zero. For example, the **/etc/init.d/perf** file writes the restart mark to the daily data by the command entry:

```
su sys —c "/usr/lib/sa/sadc /usr/adm/sa/sa`date +%d`"
```

The shell script *sa1*, a variant of *sadc*, is used to collect and store data in binary file **/usr/adm/sa/sa***dd* where *dd* is the current day. The arguments *t* and *n* cause records to be written *n* times at an interval of *t* seconds, or once if omitted. The entries in **/usr/spool/cron/crontabs/sys** (see *cron*(1M)):

```
0 * * * 0-6 /usr/lib/sa/sa1
20,40 8—17 * * 1—5 /usr/lib/sa/sa1
```

will produce records every 20 minutes during working hours and hourly otherwise.

The shell script *sa2*, a variant of *sar*(1), writes a daily report in file **/usr/adm/sa/sar***dd*. The options are explained in *sar*(1). The **/usr/spool/cron/crontabs/sys** entry:

```
5 18 * * 1—5 /usr/lib/sa/sa2 —s 8:00 —e 18:01 —i 1200 —A
```

will report important activities hourly during the working day.

The structure of the binary daily data file is:

```
struct sa {
        struct sysinfo si; /* see /usr/include/sys/sysinfo.h */
        struct minfo mi; /* defined in sys/sysinfo.h */
        struck dinfo di; /* RFS info defined in sys/sysinfo.h */
        int minserve, maxserve;  /* RFS server low and high water marks */
        int  szinode;       /* current size of inode table */
        int  szfile;        /* current size of file table */
        int  szproc;        /* current size of proc table */
        int   szlckf;       /* current size of file record header table */
        int   szlckr;       /* current size of file record lock table */
        int  mszinode;      /* size of inode table */
        int  mszfile;       /* size of file table */
        int  mszproc;       /* size of proc table */
        int   mszlckf;      /* maximum size of file record header table */
        int   mszlckr;      /* maximum size of file record lock table */
        long  inodeovf;     /* cumulative overflows of inode table */
        long  fileovf;      /* cumulative overflows of file table */
        long  procovf;      /* cumulative overflows of proc table */
        time_t  ts;         /* time stamp, seconds */
        long  devio[NDEVS][4];      /* device unit information */
#define IO_OPS          0          /* cumulative I/O requests */
#define IO_BCNT         1          /* cumulative blocks transferred */
#define IO_ACT          2          /* cumulative drive busy time in ticks */
#define IO_RESP         3          /* cumulative I/O resp time in ticks */
};
```

FILES

/usr/adm/sa/sa*dd*	daily data file
/usr/adm/sa/sar*dd*	daily report file
/tmp/sa.adrfl	address file

SEE ALSO

cron(1M).
sag(1G), sar(1), timex(1) in the *User's Reference Manual.*

NAME

setmnt — establish mount table

SYNOPSIS

/etc/setmnt

DESCRIPTION

Setmnt creates the **/etc/mnttab** table which is needed for both the *mount*(1M) and *umount* commands. *Setmnt* reads standard input and creates a *mnttab* entry for each line. Input lines have the format:

```
filesys node
```

where *filesys* is the name of the file system's *special file* (*e.g.*, **/dev/dsk/c0d0s?**) and *node* is the root name of that file system. Thus *filesys* and *node* become the first two strings in the mount table entry.

FILES

/etc/mnttab

SEE ALSO

mount(1M).

BUGS

Problems may occur if *filesys* or *node* are longer than 32 characters.
Setmnt silently enforces an upper limit on the maximum number of *mnttab* entries.

NAME

shutdown — shut down system, change system state

SYNOPSIS

/etc/shutdown [**—y**] [—ggrace_period [—iinit_state]

DESCRIPTION

This command is executed by the super-user to change the state of the machine. By default, it brings the system to a state where only the console has access to the GENIX V.3 system. This state is traditionally called "single-user."

The command sends a warning message and a final message before it starts actual shutdown activities. By default, the command asks for confirmation before it starts shutting down daemons and killing processes. The options are used as follows:

—y pre-answers the confirmation question so the command can be run without user intervention. A default of 60 seconds is allowed between the warning message and the final message. Another 60 seconds is allowed between the final message and the confirmation.

—g*grace_period*
allows the super-user to change the number of seconds from the 60-second default.

—i*init_state*
specifies the state that *init*(1M) is to be put in following the warnings, if any. By default, system state **"s"** is used (the same as states **"1"** and **"S"**).

Other recommended system state definitions are:

state 0
Shut the machine down so it is safe to remove the power. Have the machine remove power if it can. The */etc/rc0* procedure is called to do this work.

state 1, s, S
Bring the machine to the state traditionally called single-user. The */etc/rc0* procedure is called to do this work. (Though **s** and **1** are both used to go to single user state, **s** only kills processes spawned by **init** and does not unmount file systems. State **1** unmounts everything except root and kills all user processes, except those that relate to the console.)

state 5
Stop the GENIX V.3 system and go to the firmware monitor.

state 6
Stop the GENIX V.3 system and reboot to the state defined by the *initdefault* entry in **/etc/inittab**.

SEE ALSO

init(1M), rc0(1M), rc2(1M).
inittab(4) in the *Programmer's Reference Manual.*

NAME

strace — print STREAMS trace messages

SYNOPSIS

strace [mid sid level] ...

DESCRIPTION

Strace without arguments writes all STREAMS event trace messages from all drivers and modules to its standard output. These messages are obtained from the STREAMS log driver [*log*(7)]. If arguments are provided they must be in triplets of the form *mid, sid, level,* where *mid* is a STREAMS module ID number, *sid* is a sub-ID number, and *level* is a tracing priority level. Each triplet indicates that tracing messages are to be received from the given module/driver, sub-ID (usually indicating minor device), and priority level equal to or less than the given level. The token *all* may be used for any member to indicate no restriction for that attribute.

The format of each trace message output is:

<seq> <time> <ticks> <level> <flags> <mid> <sid> <text>

<seq>	trace sequence number
<time>	time of message in hh:mm:ss
<ticks>	time of message in machine ticks since boot
<level>	tracing priority level
<flags>	E : message is also in the error log F : indicates a fatal error N : mail was sent to the system administrator
<mid>	module ID number of source
<sid>	sub-ID number of source
<text>	formatted text of the trace message

Once initiated, *strace* will continue to execute until terminated by the user.

EXAMPLES

Output all trace messages from the module or driver whose module ID is 41:

```
strace 41 all all
```

Output those trace messages from driver/module ID 41 with sub-IDs 0, 1, or 2:

```
strace 41 0 1  41 1 1  41 2 0
```

Messages from sub-IDs 0 and 1 must have a tracing level less than or equal to 1. Those from sub-ID 2 must have a tracing level of 0.

CAVEATS

Due to performance considerations, only one *strace* process is permitted to open the STREAMS log driver at a time. The log driver has a list of the triplets specified in the command invocation, and compares each potential trace message against this list to decide if it should be formatted and sent up to the *strace* process. Hence, long lists of triplets will have a greater impact on overall STREAMS performance. Running *strace* will have the most impact on the timing of the modules and drivers generating the trace messages that are sent to the *strace* process. If trace messages are generated faster than the *strace* process can handle them, then some of the messages will be lost. This last case can be determined by examining the sequence numbers on the trace messages output.

SEE ALSO

log(7).
STREAMS Programmer's Guide.

NAME

strclean — STREAMS error logger cleanup program

SYNOPSIS

strclean [**-d** logdir] [**-a** age]

DESCRIPTION

Strclean is used to clean up the STREAMS error logger directory on a regular basis (for example, by using *cron*(1M)). By default, all files with names matching **error.*** in **/usr/adm/streams** that have not been modified in the last 3 days are removed. A directory other than **/usr/adm/streams** can be specified using the **-d** option. The maximum age in days for a log file can be changed using the **-a** option.

EXAMPLE

strclean -d /usr/adm/streams -a 3

has the same result as running *strclean* with no arguments.

NOTES

Strclean is typically run from *cron*(1M) on a daily or weekly basis.

FILES

/usr/adm/streams/error.*

SEE ALSO

cron(1M), strerr(1M).
STREAMS Programmer's Guide.

NAME

strerr — STREAMS error logger daemon

SYNOPSIS

strerr

DESCRIPTION

Strerr receives error log messages from the STREAMS log driver [*log*(7)] and appends them to a log file. The error log files produced reside in the directory **/usr/adm/streams,** and are named **error.***mm-dd*, where *mm* is the month and *dd* is the day of the messages contained in each log file.

The format of an error log message is:

<seq> <time> <ticks> <flags> <mid> <sid> <text>

<seq>	error sequence number
<time>	time of message in hh:mm:ss
<ticks>	time of message in machine ticks since boot priority level
<flags>	T : the message was also sent to a tracing process F : indicates a fatal error N : send mail to the system administrator
<mid>	module ID number of source
<sid>	sub-ID number of source
<text>	formatted text of the error message

Messages that appear in the error log are intended to report exceptional conditions that require the attention of the system administrator. Those messages which indicate the total failure of a STREAMS driver or module should have the F flag set. Those messages requiring the immediate attention of the administrator will have the N flag set, which causes the error logger to send the message to the system administrator via *mail*(1). The priority level usually has no meaning in the error log but will have meaning if the message is also sent to a tracer process.

Once initiated, *strerr* will continue to execute until terminated by the user. Commonly, *strerr* would be executed asynchronously.

CAVEATS

Only one *strerr* process at a time is permitted to open the STREAMS log driver.

If a module or driver is generating a large number of error messages, running the error logger will cause a degradation in STREAMS performance. If a large burst of messages are generated in a short time, the log driver may not be able to deliver some of the messages. This situation is indicated by gaps in the sequence numbering of the messages in the log files.

FILES

/usr/adm/streams/error.*mm-dd*

SEE ALSO

log(7).
STREAMS Programmer's Guide.

NAME

su — become super-user or another user

SYNOPSIS

su [—] [name [arg ...]]

DESCRIPTION

Su allows one to become another user without logging off. The default user *name* is **root** (*i.e.*, super-user).

To use *su*, the appropriate password must be supplied (unless one is already **root**). If the password is correct, *su* will execute a new shell with the real and effective user ID set to that of the specified user. The new shell will be the optional program named in the shell field of the specified user's password file entry (see *passwd*(4)), or **/bin/sh** if none is specified (see *sh*(1)). To restore normal user ID privileges, type an **EOF** (*cntrl-d*) to the new shell.

Any additional arguments given on the command line are passed to the program invoked as the shell. When using programs like *sh*(1), an *arg* of the form **—c** *string* executes *string* via the shell and an arg of **—r** will give the user a restricted shell.

The following statements are true only if the optional program named in the shell field of the specified user's password file entry is like *sh*(1). If the first argument to *su* is a **—**, the environment will be changed to what would be expected if the user actually logged in as the specified user. This is done by invoking the program used as the shell with an *arg0* value whose first character is **—**, thus causing first the system's profile (**/etc/profile**) and then the specified user's profile (**.profile** in the new HOME directory) to be executed. Otherwise, the environment is passed along with the possible exception of **$PATH**, which is set to **/bin:/etc:/usr/bin** for **root**. Note that if the optional program used as the shell is **/bin/sh**, the user's **.profile** can check *arg0* for **—sh** or **—su** to determine if it was invoked by *login*(1) or *su*(1), respectively. If the user's program is other than **/bin/sh**, then **.profile** is invoked with an *arg0* of *-program* by both *login*(1) and *su*(1).

All attempts to become another user using *su* are logged in the log file **/usr/adm/sulog**.

EXAMPLES

To become user **bin** while retaining your previously exported environment, execute:

```
su bin
```

To become user **bin** but change the environment to what would be expected if **bin** had originally logged in, execute:

```
su - bin
```

To execute *command* with the temporary environment and permissions of user **bin**, type:

```
su - bin -c "command args"
```

FILES

/etc/passwd	system's password file
/etc/profile	system's profile
$HOME/.profile	user's profile
/usr/adm/sulog	log file

SEE ALSO

env(1), login(1), sh(1) in the *User's Reference Manual*.
passwd(4), profile(4), environ(5) in the *Programmer's Reference Manual*.

NAME

swap — swap administrative interface

SYNOPSIS

/etc/swap —a swapdev swaplow swaplen
/etc/swap —d swapdev swaplow
/etc/swap —l

DESCRIPTION

Swap provides a method of adding, deleting, and monitoring the system swap areas used by the memory manager. The following options are recognized:

—a Add the specified swap area. *swapdev* is the name of the block special device, *e.g.*, **/dev/dsk/c0d0s1.** *Swaplow* is the offset in 512-byte blocks into the device where the swap area should begin. *Swaplen* is the length of the swap area in 512-byte blocks. This option can only be used by the superuser. Swap areas are normally added by the system start up routine **/etc/rc** when going into multi-user mode.

—d Delete the specified swap area. *Swapdev* is the name of block special device, *e.g.*, **/dev/dsk/1s0.** *swaplow* is the offset in 512-byte blocks into the device where the swap area should begin. Using this option marks the swap area as "INDEL" (in process of being deleted). The system will not allocate any new blocks from the area, and will try to free swap blocks from it. The area will remain in use until all blocks from it are freed. This option can only be used by the superuser.

—l List the status of all the swap areas. The output has four columns:

DEV	The *swapdev* special file for the swap area if one can be found in the **/dev/dsk** or **/dev** directories, and its major/minor device number in decimal.
LOW	The *swaplow* value for the area in 512-byte blocks.
LEN	The *swaplen* value for the area in 512-byte blocks.
FREE	The number of free 512-byte blocks in the area. If the swap area is being deleted, this column will be marked INDEL.

WARNINGS

No check is done to see if a swap area being added overlaps with an existing swap area or file system.

NAME

sync — update the super block

SYNOPSIS

sync

DESCRIPTION

Sync executes the *sync* system primitive. If the system is to be stopped, *sync* must be called to insure file system integrity. It will flush all previously unwritten system buffers out to disk, thus assuring that all file modifications up to that point will be saved. See *sync*(2) for details.

NOTE

If you have done a write to a file on a remote machine in a Remote File Sharing environment, you cannot use *sync* to force buffers to be written out to disk on the remote machine. *Sync* will only write local buffers to local disks.

SEE ALSO

sync(2) in the *Programmer's Reference Manual.*

NAME

sysadm — menu interface to do system administration

SYNOPSIS

sysadm [*sub-command*]

DESCRIPTION

This command, when invoked without an argument, presents a menu of system administration sub-commands, from which the user selects. If the optional argument is presented, the named sub-command is run or the named sub-menu is presented.

The *sysadm* command may be given a password. See **admpasswd** in the SUBCOMMANDS section.

SUB-COMMANDS

The following menus of sub-commands are available. (The number of bullets (●) in front of each item indicates the level of the menu or subcommand.)

● diagnostics

system diagnostics menu

These subcommands look for and sometimes repair problems in the system. Those subcommands that issue reports allow you to determine if there are detectable problems. Commands that attempt repair are for repair people only. You must know what you are doing!

● ● diskrepair

advice on repair of built-in disk errors

This subcommand advises you on how to go about repairing errors that occur on built-in disks.

WARNING: Because this is a repair function, it should only be performed by qualified service personnel.
NOTE: Reports of disk errors most probably result in the loss of files and/or damage to data. It will be necessary to restore the repaired disk from backup copies.

● ● diskreport

report on built-in disk errors

This subcommand shows you if the system has collected any information indicating that there have been errors while reading the built-in disks. You can request either summary or full reports. The summary report provides sufficient information about disk errors to determine if repair should be attempted. If the message no errors logged is part of the report, then there is probably no damage. If a number of errors is reported, there is damage and you should call for service. The full report gives additional detail for the expert repair person trouble shooting complicated problems.

NOTE: Reports of disk errors most probably result in the loss of files and/or damage to data. It will be necessary to restore the repaired disk from backup copies.

● diskmgmt

disk management menu

The subcommands in this menu provide functions for using removable disks. The subcommands include the ability to format disks, copy disks, and to use disks as mountable file systems. It also contains a menu of subcommands for handling non-removable media.

● ● checkfsys
check a removable disk file system for errors

Checkfsys checks a file system on a removable disk for errors. If there are errors, this procedure attempts to repair them.

● ● cpdisk
make exact copies of a removable disk

This procedure copies the contents of a removable disk into the machine and then allows the user to make exact copies of it. These copies are identical to the original in every way. The copies are made by first reading the original removable disk entirely into the machine and then writing it out onto duplicate disks. The procedure will fail if there is not enough space in the system to hold the original disk.

● ● erase
erase data from removable disk

This procedure erases a removable disk by overwriting it with null bytes. The main purpose is to remove data that the user does not want seen. Once performed, this operation is irreversible.

● ● format
format new removable disks

Format prepares new removable disks for use. Once formatted, programs and data can be written on the disks.

● ● harddisk
hard disk management menu

The subcommands in this menu provide functions for using hard disks. For each hard disk, the disk can be partitioned with default partitioning or the current disk partitioning can be displayed.

● ● ● display
display hard disk partitioning

Display will allow the user to display the hard disk partitioning. This will inform the user of current disk partitioning information.

● ● ● partitioning
partition a hard disk

Partitioning configures hard disks. This will allow you to partition a hard disk according to the default partitioning.

● ● ● rmdisk
remove a hard disk

Removes a hard disk from the system configuration. It may then be physically disconnected (once the machine has been turned off) or freshly partitioned (after the machine has been restarted).

● ● makefsys
create a new file system on a removable disk

Makefsys creates a new file system on a removable disk which can then store data which the user does not wish to keep on the hard disk. When "mounted", the file system has all the properties of a file kept on the hard disk, except that it is smaller.

● ● mountfsys

mount a removable disk file system

Mountfsys mounts a file system, found on a removable disk, making it available to the user. The file system is unmounted with the "umountfsys" command. THE DISK MUST NOT BE REMOVED WHILE THE FILE SYSTEM IS STILL MOUNTED.
IF THE FILE SYSTEM HAS BEEN MOUNTED WITH THE **mountfsys** COMMAND, IT MUST BE UNMOUNTED WITH **umountfsys.**

● ● umountfsys

unmount a removable disk file system

Umountfsys unmounts a file system, allowing the user to remove the disk. THE DISK MUST NOT BE REMOVED UNTIL THE FILE SYSTEM IS UNMOUNTED.
umountfsys MAY ONLY BE USED TO UNMOUNT FILE SYSTEMS MOUNTED WITH THE **mountfsys** COMMAND.

● filemgmt

file management menu

The subcommands in this menu allow the user to protect files on the hard disk file systems by copying them onto diskettes and later restoring them to the hard disk by copying them back. Subcommands are also provided to determine which files might be best kept on diskette based on age or size.

● ● backup

backup files from integral hard disk to removable disk or tape

Backup saves copies of files from the integral hard disk file systems to removable disk or tape. There are two kinds of backups:

COMPLETE — copies all files (useful in case of serious file system damage)

INCREMENTAL — copies files changed since the last backup

The normal usage is to do a complete backup of each file system and then periodically do incremental backups. Two cycles are recommended (one set of complete backups and several incrementals to each cycle). Files backed up with "backup" are restored using "restore."

● ● bupsched

backup reminder scheduling menu

Backup scheduling is used to schedule backup reminder messages and backup reminder checks. Backup reminder messages are sent to the console to remind the administrator to backup particular file systems when the machine is shutdown or a reminder check has been run during the specified time period.

Backup reminder checks specify particular times at which the system will check to see if any backup reminder messages have been scheduled.

● ● ● schedcheck

schedule backup reminder checks

Backup reminder checks are run at specific times to check to see if any reminders are scheduled. The user specifies the times at which the check is to be run. Checks are run for the reminder messages scheduled by *schedmsg*.

● ● ● schedmsg
schedule backup reminder message

Backup reminder messages are sent to the console if the machine is shutdown or a reminder check has been scheduled. The user specifies the times at which it is appropriate to send a message and the file systems to be included in the message.

● ● diskuse
display how much of the hard disk is being used

Diskuse lets the user know what percentage of the hard disk is currently occupied by files. The list is organized by file system names.

● ● fileage
list files older than a particular date

Fileage prints the names of all files older than the date specified by the user. If no date is entered, all files older than 90 days will be listed. If no directory is specified to look in, the **/usr/admin** directory will be used.

● ● filesize
list the largest files in a particular directory

Filesize prints the names of the largest files in a specific directory. If no directory is specified, the **/usr/admin** directory will be used. If the user does not specify how many large files to list, 10 files will be listed.

● ● restore
restore files from "backup" and "store" media to integral hard disk

Restore copies files from disks and tapes made by "backup" and "store" back onto the hard disk. You can restore individual files, directories of files, or the entire contents of a disk or tape. The user can restore from both "incremental" and "complete" media. The user can also list the names of files stored on the disk or tape.

● ● store
store files and directories of files onto disk or tape

Store copies files from the integral hard disk to disk or tape and allows the user to optionally verify that they worked and to optionally remove them when done. Typically, these would be files that the user wants to archive or restrict access to. The user can store single files and directories of files. Use the "restore" command to put stored files back on the integral hard disk and to list the files stored.

● machinemgmt
machine management menu

Machine management functions are tools used to operate the machine, *e.g.*, turn it off, reboot, or go to the firmware monitor.

● ● autold
set automatic boot device, default manual boot program

This procedure specifies the default manual program to boot from firmware and/or the device to be used when automatically rebooting.

● ● firmware

stop all running programs then enter firmware mode

This procedure will stop all running programs, close any open files, write out information to the disk (such as directory information), then enter the firmware mode. (Machine diagnostics and other special functions that are not available on the GENIX V.3 system.)

● ● floppykey

create a "floppy key" removable disk

The "floppy key" removable disk allows the user to enter firmware mode if the firmware password has been changed and then forgotten. Thus the "floppy key" is just that, the "key" to the system and should be protected as such.

● ● powerdown

stop all running programs, then turn off the machine

Powerdown will stop all running programs, close any open files, write out information to disk (such as directory information), then turn the machine power off.

● ● reboot

stop all running programs then reboot the machine

Reboot will stop all running programs, close any open files, write out information to disk (such as directory information), then reboot the machine. This can be used to get out of some types of system trouble, such as when a process cannot be killed.

● ● whoson

print list of users currently logged onto the system

Whoson prints the login ID, terminal device number, and sign-on time of all users who are currently using the computer.

● packagemgmt

package management

These submenus and subcommands manage various software and hardware packages that you install on your machine. Not all optional packages add subcommands here.

● softwaremgmt

software management menu

These subcommands permit the user to install new software, remove software, and run software directly from the removable disk it is delivered on. The "remove" and "run" capabilities are dependent on the particular software packages. See the instructions delivered with each package.

● ● installpkg
install new software package onto integral hard disk

Install copies files from removable disk onto the integral hard disk and performs additional work if necessary so that the software can be run. From then on, the user will have access to those commands.

● ● listpkg
list packages already installed

This subcommand show you a list of currently installed optional software packages.

● ● removepkg
remove previously installed package from integral hard disk

This subcommand displays a list of currently installed optional software packages. Actions necessary to remove the software packages specified by the user will then be performed. The removable disk used to "installpkg" the software is needed to remove it.

● ● runpkg
run software package without installing it

This package allows the user to run software from a removable disk without installing it permanently on the system. This is useful if the user does not use the software often or does not have enough room on the system. WARNING: Not all software packages have the ability to run their contents this way. See the instructions that come with the software package.

● syssetup
system setup menu

System setup routines allow the user to tell the computer what its environment looks like: what the date, time, and time zone is, what administration and system capabilities are to be under password control, what the machine's name is, etc. The first-time setup sequence is also here.

● ● admpasswd
assign or change administrative passwords

Admpasswd lets you set or make changes to passwords for administrative commands and logins such as setup and sysadm.

● ● datetime
set the date, time, time zone, and daylight savings time

Datetime tells the computer the date, time, time zone, and whether you observe Daylight Savings Time (DST). It is normally run once when the machine is first set up. If you observe DST, the computer will automatically start to observe it in the spring and return to Standard Time in the fall. The machine has to be turned off and turned back on again to guarantee that ALL times will be reported correctly. Most are correct the next time the user logs in.

● ● nodename
    set the node name of this machine

    This allows you to change the node name of this machine. The node name is used by various communications networks to identify this machine.

● ● setup
    set up your machine the very first time

    Setup allows the user to define the first login, to set the passwords on the user-definable administration logins and to set the time zone for your location.

● ● syspasswd
    assign system passwords

    Syspasswd lets the user set system passwords normally reserved for the very knowledgeable user. For this reason, this procedure may assign those passwords, but may not change or clear them. Once set, they may only be changed by the specific login or the "root" login.

● ttymgmt
    terminal management

    This procedure allows the user to manage the computer's terminal functions.

● ● lineset
    show tty line settings and hunt sequences

    The tty line settings are often hunt sequences where, if the first line setting does not work, the line "hunts" to the next line setting until one that does work comes by. This subcommand shows the various sequences with only specific line settings in them. It also shows each line setting in detail.

● ● mklineset
    create new tyy line settings and hunt sequences

    This subcommand helps you to create tty line setting entries. You might want to add line settings that are not in the current set or create hunt sequences with only specific line settings in them. The created hunt sequences are circular; stepping past the last setting puts you on the first.

● ● modtty
    show and optionally modify characteristics of tty lines

    This subcommand reports and allows you to change the characteristics of tty lines (also called "ports").

● usermgmt
    user management menu

    These subcommands allow you to add, modify and delete the list of users that have access to your machine. You can also place them in separate groups so that they can share access to files within the group but protect themselves from other groups.

● ● addgroup
    add a group to the system

    Addgroup adds a new group name or ID to the computer. Group names and IDs are used to identify groups of users who desire common access to a set of files and directories.

● ● adduser
add a user to the system

Adduser installs a new login ID on the machine. You are asked a series of questions about the user and then the new entry is made. You can enter more than one user at a time. Once this procedure is finished, the new login ID is available.

● ● delgroup
delete a group from the system

Delgroup allows you to remove groups from the computer. The deleted group is no longer identified by name. However, files may still be identified with the group ID number.

● ● deluser
delete a user from the system

Deluser allows you to remove users from the computer. The deleted user's files are removed from the hard disk and their logins are removed from the **/etc/passwd** file.

● ● lsgroup
list groups in the system

Lsgroup will list all the groups that have been entered into the computer. This list is updated automatically by "addgroup" and "delgroup"

● ● lsuser
list users in the system

Lsuser will list all the users that have been entered into the computer. This list is updated automatically by "adduser" and "deluser."

● ● modadduser
modify defaults used by adduser

Modadduser allows the user to change some of the defaults used when adduser creates a new login. Changing the defaults does not effect any existing logins, only logins made from this point on.

● ● modgroup
make changes to a group on the system

Modgroup allows the user to change the name of a group that the user enters when "addgroup" is run to set up new groups.

● ● moduser
menu of commands to modify a user's login

This menu contains commands that modify the various aspects of a user's login.

● ● ● chgloginid
change a user's login ID

This procedure allows the user to change a user's login ID. Administrative and system logins cannot be changed.

● ● ● chgpasswd
change a user's passwd

This proceudure allows removal or change of a suer's password. Administrative and system login passwords channot be changed. To change administrative and system login passwords, see the system setup menu: sysadm syssetup.

● ● ● chgshell
change a user's login shell

This procedure allows the user to change the command run when a user logs in. The login shell of the administrative and system logins cannot be changed by this procedure.

EXAMPLES

sysadm adduser

FILES

The files that support *sysadm* are found in **/usr/admin.**

The menu starts in directory **/usr/admin/menu.**

NAME

sysdef — output system definition

SYNOPSIS

/etc/sysdef [system_namelist [master.d]]

DESCRIPTION

Sysdef outputs the current system definition in tabular form. It lists all hardware devices, their local bus addresses, and unit count, as well as pseudo devices, system devices, loadable modules and the values of all tunable parameters. It generates the output by analyzing the named operating system file (*system_namelist*) and extracting the configuration information from the name list itself.

FILES

/unix	default operating system file (where the system namelist is)
/etc/master.d/*	default directory containing master files

SEE ALSO

master(4), nlist(3C) in the *Programmer's Reference Manual.*

DIAGNOSTICS

internal name list overflow

if the master table contains more than an internally specified number of entries for use by *nlist*(3C).

NAME

tic — terminfo compiler

SYNOPSIS

tic [—v[n]] [—c] file

DESCRIPTION

Tic translates a *terminfo*(4) file from the source format into the compiled format. The results are placed in the directory */usr/lib/terminfo*. The compiled format is necessary for use with the library routines described in *curses*(3X).

—vn	(verbose) output to standard error trace information showing *tic*'s progress. The optional integer *n* is a number from 1 to 10, inclusive, indicating the desired level of detail of information. If *n* is omitted, the default level is 1. If *n* is specified and greater than 1, the level of detail is increased.
—c	only check *file* for errors. Errors in **use=** links are not detected.
file	contains one or more *terminfo*(4) terminal descriptions in source format (see *terminfo*(4)). Each description in the file describes the capabilities of a particular terminal. When a **use=***entry-name* field is discovered in a terminal entry currently being compiled, *tic* reads in the binary from */usr/lib/terminfo* to complete the entry. (Entries created from *file* will be used first. If the environment variable **TERMINFO** is set, that directory is searched instead of */usr/lib/terminfo*.) *tic* duplicates the capabilities in *entry-name* for the current entry, with the exception of those capabilities that explicitly are defined in the current entry.

If the environment variable **TERMINFO** is set, the compiled results are placed there instead of */usr/lib/terminfo*.

FILES

/usr/lib/terminfo/?/* compiled terminal description data base

SEE ALSO

curses(3X), term(4), terminfo(4) in the *Programmer's Reference Manual*.
Chapter 10 in the *Programmer's Guide*.

WARNINGS

Total compiled entries cannot exceed 4096 bytes. The name field cannot exceed 128 bytes.

Terminal names exceeding 14 characters will be truncated to 14 characters and a warning message will be printed.

When the **—c** option is used, duplicate terminal names will not be diagnosed; however, when **—c** is not used, they will be.

BUGS

To allow existing executables from the previous release of the GENIX V.3 System to continue to run with the compiled terminfo entries created by the new terminfo compiler, cancelled capabilities will not be marked as cancelled within the terminfo binary unless the entry name has a '+' within it. (Such terminal names are only used for inclusion within other entries via a **use=** entry. Such names would not be used for real terminal names.)

For example:

```
4415+nl, kf1@, kf2@, ....

4415+base, kf1=\EOc, kf2=\EOd, ....

4415-nl|4415 terminal without keys,
    use=4415+nl, use=4415+base,
```

The above example works as expected; the definitions for the keys do not show up in the *4415—nl* entry. However, if the entry *4415+nl* did not have a plus sign within its name, the cancellations would not be marked within the compiled file and the definitions for the function keys would not be cancelled within *4415—nl.*

DIAGNOSTICS

Most diagnostic messages produced by *tic* during the compilation of the source file are preceded with the approximate line number and the name of the terminal currently being worked on.

mkdir ... returned bad status
: The named directory could not.be created.

File does not start with terminal names in column one
: The first thing seen in the file, after comments, must be the list of terminal names.

Token after a *seek*(2) not NAMES
: Somehow the file being compiled changed during the compilation.

Not enough memory for use_list element
or
Out of memory
: Not enough free memory was available (*malloc*(3) failed).

Can't open ...
: The named file could not be created.

Error in writing ...
: The named file could not be written to.

Can't link ... to ...
: A link failed.

Error in re-reading compiled file ...
The compiled file could not be read back in.

Premature EOF
: The current entry ended prematurely.

Backspaced off beginning of line
: This error indicates something wrong happened within *tic*.

Unknown Capability - "..."
: The named invalid capability was found within the file.

Wrong type used for capability "..."
: For example, a string capability was given a numeric value.

Unknown token type
: Tokens must be followed by '@' to cancel, ',' for booleans, '#' for numbers, or '=' for strings.

"...": bad term name
or
Line ...: Illegal terminal name - "..."

Terminal names must start with a letter or digit
: The given name was invalid. Names must not contain white space or slashes, and must begin with a letter or digit.

"...": terminal name too long.
: An extremely long terminal name was found.

"...": terminal name too short.
: A one-letter name was found.

"..." filename too long, truncating to "..."
: The given name was truncated to 14 characters due to GENIX V.3 file name length limitations.

"..." defined in more than one entry. Entry being used is "...".
: An entry was found more than once.

Terminal name "..." synonym for itself
: A name was listed twice in the list of synonyms.

At least one synonym should begin with a letter.
: At least one of the names of the terminal should begin with a letter.

Illegal character - "..."
: The given invalid character was found in the input file.

Newline in middle of terminal name
: The trailing comma was probably left off of the list of names.

Missing comma
: A comma was missing.

Missing numeric value
: The number was missing after a numeric capability.

NULL string value
: The proper way to say that a string capability does not exist is to cancel it.

Very long string found. Missing comma?
: self-explanatory

Unknown option. Usage is:
: An invalid option was entered.

Too many file names. Usage is:
: self-explanatory

"..." non-existant or permission denied
: The given directory could not be written into.

"..." is not a directory
: self-explanatory

"...": Permission denied
: access denied.

"...": Not a directory
: *tic* wanted to use the given name as a directory, but it already exists as a file

SYSTEM ERROR!! Fork failed!!!
: A *fork*(2) failed.

Error in following up use-links. Either there is a loop in the links or they reference non-existant terminals. The following is a list of the entries involved:
: A *terminfo*(4) entry with a **use=***name* capability either referenced a non-existant terminal called *name* or *name* somehow referred back to the given entry.

NAME

uadmin — administrative control

SYNOPSIS

/etc/uadmin cmd fcn

DESCRIPTION

The *uadmin* command provides control for basic administrative functions. This command is tightly coupled to the System Administration procedures and is not intended for general use. It may be invoked only by the super-user.

The arguments *cmd* (command) and *fcn* (function) are converted to integers and passed to the *uadmin* system call.

SEE ALSO

uadmin(2) in the *Programmer's Reference Manual.*

NAME

umountall [—k]

DESCRIPTION

The *umountall* command is executed by the super-user, "root," to unmount all currently mounted file systems except the root file system.

The **—k** option causes *fuser*(1M) to send a SIGKILL signal to all processes that have files open in each file system before it is unmounted. Without **—k**, an unmount may fail because the file system is busy.

EXAMPLES

```
/etc/umountall

/etc/umountall —k
```

SEE ALSO

fuser(1M).
signal(2) in the *Programmer's Reference Manual.*

NAME

unadv — unadvertise a Remote File Sharing resource

SYNOPSIS

unadv *resource*

DESCRIPTION

Unadv unadvertises a Remote File Sharing *resource*, which is the advertised symbolic name of a local directory, by removing it from the advertised information on the domain name server. *unadv* prevents subsequent remote mounts of that resource. It does not affect continued access through existing remote or local mounts.

An administrator at a server can unadvertise only those resources that physically reside on the local machine. A domain administrator can unadvertise any resource in the domain from the primary name server by specifying *resource* name as *domain.resource*. (A domain administrator should only unadvertise another hosts resources to clean up the domain advertise table when that host goes down. Unadvertising another host's resource changes the domain advertise table, but not the host advertise table.)

This command is restricted to the super-user.

ERRORS

If *resource* is not found in the advertised information, an error message will be sent to standard error.

SEE ALSO

adv(1M), fumount(1M), nsquery(1M).

NAME

uucheck — check the uucp directories and permissions file

SYNOPSIS

/usr/lib/uucp/uucheck [**—v**] [**—x** debug_level]

DESCRIPTION

Uucheck checks for the presence of the *uucp* system required files and directories. Within the *uucp* makefile, it is executed before the installation takes place. It also checks for some obvious errors in the Permissions file (**/usr/lib/uucp/Permissions**). When executed with the **—v** option, it gives a detailed explanation of how the uucp programs will interpret the Permissions file. The **—x** option is used for debugging. *Debug-option* is a single digit in the range 1-9; the higher the value, the greater the detail.

Note that *uucheck* can only be used by the super-user or *uucp*.

FILES

/usr/lib/uucp/Systems
/usr/lib/uucp/Permissions
/usr/lib/uucp/Devices
/usr/lib/uucp/Maxuuscheds
/usr/lib/uucp/Maxuuxqts
/usr/spool/uucp/*
/usr/spool/locks/LCK*
/usr/spool/uucppublic/*

SEE ALSO

uucico(1M), uusched(1M).
uucp(1C), uustat(1C), uux(1C) in the *User's Reference Manual.*

BUGS

The program does not check file/directory modes or some errors in the Permissions file such as duplicate login or machine name.

NAME

uucico — file transport program for the uucp system

SYNOPSIS

/usr/lib/uucp/uucico [**—r** role_number] [**—x** debug_level]
[**—i** interface] [**—d** spool_directory] **—s** system_name

DESCRIPTION

Uucico is the file transport program for *uucp* work file transfers. Role numbers for the **—r** are the digit 1 for master mode or 0 for slave mode (default). The **—r** option should be specified as the digit 1 for master mode when *uucico* is started by a program or *cron. Uux* and *uucp* both queue jobs that will be transferred by *uucico.* It is normally started by the scheduler, *uusched,* but can be started manually; this is done for debugging. For example, the shell *Uutry* starts *uucico* with debugging turned on. A single digit must be used for the **—x** option with higher numbers for more debugging.

The **—i** option defines the interface used with *uucico.* This interface only affects slave mode. Known interfaces are UNIX (default), TLI (basic Transport Layer Interface), and TLIS (Transport Layer Interface with Streams modules, read/write).

FILES

/usr/lib/uucp/Systems
/usr/lib/uucp/Permissions
/usr/lib/uucp/Devices
/usr/lib/uucp/Devconfig
/usr/lib/uucp/Sysfiles
/usr/lib/uucp/Maxuuxqts
/usr/lib/uucp/Maxuuscheds
/usr/spool/uucp/*
/usr/spool/locks/LCK*
/usr/spool/uucppublic/*

SEE ALSO

cron(1M), uusched(1M), uutry(1M).
uucp(1C), uustat(1C), uux(1C) in the *User's Reference Manual.*

NAME

uucleanup — uucp spool directory clean-up

SYNOPSIS

/usr/lib/uucp/uucleanup [**—C***time*] [**—W***time*] [**—X***time*] [**—m***string*]
[**—o***time*] [**—s***system*]

DESCRIPTION

Uucleanup will scan the spool directories for old files and take appropriate action to remove them in a useful way:

Inform the requestor of send/receive requests for systems that can not be reached.

Return mail, which cannot be delivered, to the sender.

Delete or execute rnews for rnews type files (depending on where the news originated——locally or remotely).

Remove all other files.

In addition, there is provision to warn users of requests that have been waiting for a given number of days (default 1). Note that *uucleanup* will process as if all option *times* were specified to the default values unless *time* is specifically set.

The following options are available.

—C*time* Any **C.** files greater or equal to *time* days old will be removed with appropriate information to the requestor. (default 7 days)

—D*time* Any **D.** files greater or equal to *time* days old will be removed. An attempt will be made to deliver mail messages and execute rnews when appropriate. (default 7 days)

—W*time* Any **C.** files equal to *time* days old will cause a mail message to be sent to the requestor warning about the delay in contacting the remote. The message includes the *JOBID*, and in the case of mail, the mail message. The administrator may include a message line telling whom to call to check the problem (**—m** option). (default 1 day)

—X*time* Any **X.** files greater or equal to *time* days old will be removed. The **D.** files are probably not present (if they were, the **X.** could get executed). But if there are **D.** files, they will be taken care of by D. processing. (default 2 days)

—m*string*
This line will be included in the warning message generated by the **—W** option.

—o*time* Other files whose age is more than *time* days will be deleted. (default 2 days) The default line is "See your local administrator to locate the problem."

—s*system* Execute for *system* spool directory only.

—x*debug_level*
The **—x** debug level is a single digit between 0 and 9; higher numbers give more detailed debugging information. (If *uucleanup* was compiled with -DSMALL, no debugging output will be available.)

This program is typically started by the shell *uudemon.cleanup*, which should be started by *cron*(1M).

FILES

/usr/lib/uucp	directory with commands used by *uucleanup* internally
/usr/spool/uucp	spool directory

SEE ALSO

cron(1M).
uucp(1C), uux(1C) in the *User's Reference Manual.*

NAME

uugetty — set terminal type, modes, speed, and line discipline

SYNOPSIS

/usr/lib/uucp/getty [**—h**] [**—t** timeout] [**—r**] line
[speed [type [linedisc]]]
/usr/lib/uucp/getty **—c** file

DESCRIPTION

Uugetty is identical to *getty*(1M) but changes have been made to support using the line for *uucico, cu,* and *ct*; that is, the line can be used in both directions. The *uugetty* will allow users to login, but if the line is free, *uucico, cu,* or *ct* can use it for dialing out. The implementation depends on the fact that *uucico, cu,* and *ct* create lock files when devices are used. When the "open()" returns (or the first character is read when **—r** option is used), the status of the lock file indicates whether the line is being used by *uucico, cu, ct,* or someone trying to login. Note that in the **—r** case, several <carriage-return> characters may be required before the login message is output. The human users will be able to handle this slight inconvenience. *Uucico* trying to login will have to be told by using the following login script:

```
        "" \r\d\r\d\r\d\r in:--in: . . .
```

where the . . . is whatever would normally be used for the login sequence.

An entry for an intelligent modem or direct line that has a *uugetty* on each end must use the **—r** option. (This causes *uugetty* to wait to read a character before it puts out the login message, thus preventing two uugettys from looping.) If there is a *uugetty* on one end of a direct line, there must be a *uugetty* on the other end as well. Here is an **/etc/inittab** entry using *uugetty* on an intelligent modem or direct line:

```
        30:2:respawn:/usr/lib/uucp/uugetty —r —t 60 tty12 1200
```

FILES

/etc/gettydefs
/etc/issue

SEE ALSO

uucico(1M), getty(1M), init(1M), tty(7).
ct(1C), cu(1C), login(1) in the *User's Reference Manual*.
ioctl(2), gettydefs(4), inittab(4) in the *Programmer's Reference Manual*.

BUGS

Ct will not work when *uugetty* is used with an intelligent modem such as penril or ventel.

NAME

uusched — the scheduler for the uucp file transport program

SYNOPSIS

/usr/lib/uucp/uusched [**—x** debug_level] [**—u** debug_level]

DESCRIPTION

Uusched is the *uucp* file transport scheduler. It is usually started by the daemon *uudemon.hour* that is started by *cron*(1M) from an entry in **/usr/spool/cron/crontab:**

```
39 * * * * /bin/su uucp -c "/usr/lib/uucp/uudemon.hour > /dev/null"
```

The two options are for debugging purposes only; **—x** *debug_level* will output debugging messages from *uusched* and **—u** *debug_level* will be passed as **—x** *debug_level* to *uucico*. The *debug_level* is a number between 0 and 9; higher numbers give more detailed information.

FILES

/usr/lib/uucp/Systems
/usr/lib/uucp/Permissions
/usr/lib/uucp/Devices
/usr/spool/uucp/*
/usr/spool/locks/LCK*
/usr/spool/uucppublic/*

SEE ALSO

cron(1M), uucico(1M).
uucp(1C), uustat(1C), uux(1C) in the *User's Reference Manual.*

NAME

uuxqt — execute remote command requests

SYNOPSIS

/usr/lib/uucp/uuxqt [—s system] [—x debug_level]

DESCRIPTION

Uuxqt is the program that executes remote job requests from remote systems generated by the use of the *uux* command. (*Mail* uses *uux* for remote mail requests.) *Uuxqt* searches the spool directories looking for **X.** files. For each **X.** file, *uuxqt* checks to see if all the required data files are available and accessible, and file commands are permitted for the requesting system. The **Permissions** file is used to validate file accessibility and command execution permission.

There are two environment variables that are set before the *uuxqt* command is executed:
UU_MACHINE is the machine that sent the job (the previous one).
UU_USER is the user that sent the job.
These can be used in writing commands that remote systems can execute to provide information, auditing, or restrictions.

The -x *debug_level* is a single digit between 0 and 9. Higher numbers give more detailed debugging information.

FILES

/usr/lib/uucp/Permissions
/usr/lib/uucp/Maxuuxqts
/usr/spool/uucp/*
/usr/spool/locks/LCK*

SEE ALSO

uucico(1M).
uucp(1C), uustat(1C), uux(1C), mail(1) in the *User's Reference Manual.*

NAME

volcopy — make literal copy of file system

SYNOPSIS

/etc/volcopy [options] fsname srcdevice volname1 destdevice volname2

DESCRIPTION

Volcopy makes a literal copy of the file system using a blocksize matched to the device. *Options* are:

—a	invoke a verification sequence requiring a positive operator response instead of the standard 10 second delay before the copy is made
—s	(default) invoke the **DEL if wrong** verification sequence.

The program requests length and density information if it is not given on the command line or is not recorded on an input tape label. If the file system is too large to fit on one reel, *volcopy* will prompt for additional reels. Labels of all reels are checked. Tapes may be mounted alternately on two or more drives. If *volcopy* is interrupted, it will ask if the user wants to quit or wants a shell. In the latter case, the user can perform other operations (*e.g.*, *labelit*) and return to *volcopy* by exiting the new shell.

The *fsname* argument represents the mounted name (*e.g.*, **root**, **u1**, etc.) of the filsystem being copied.

The *srcdevice* or *destdevice* should be the physical disk section or tape (*e.g.*, **/dev/dsk/c0d0s2.**

The *volname* is the physical volume name (*e.g.*, **pk3**, **t0122**, etc.) and should match the external label sticker. Such label names are limited to six or fewer characters. *Volname* may be — to use the existing volume name.

Srcdevice and *volname1* are the device and volume from which the copy of the file system is being extracted. *Destdevice* and *volname2* are the target device and volume.

Fsname and *volname* are recorded in the last 12 characters of the superblock (**char fsname[6], volname[6]**).

FILES

/etc/log/filesave.log a record of file systems/volumes copied

SEE ALSO

labelit(1M).
fs(4) in the *Programmer's Reference Manual.*
sh(1) in the *User's Reference Manual.*

WARNINGS

Volcopy does not support tape-to-tape copying. Use *dd*(1) for tape-to-tape copying.

NAME

whodo — who is doing what

SYNOPSIS

/etc/whodo

DESCRIPTION

Whodo produces formatted and dated output from information in the */etc/utmp* and */etc/ps_data* files.

The display is headed by the date, time and machine name. For each user logged in, device name, user-ID and login time is shown, followed by a list of active processes associated with the user-ID. The list includes the device name, process-ID, CPU minutes and seconds used, and process name.

EXAMPLE

The command:

```
whodo
```

produces a display like this:

```
Tue Mar 12 15:48:03 1985
bailey

console    mcn      8:51
   console   28158    0:29 sh

tty02    bdr     15:23
   tty02   21688    0:05 sh
   tty02   22788    0:01 whodo
   tty02   22017    0:03 vi
   tty02   22549    0:01 sh
```

FILES

/etc/passwd
/etc/ps_data
/etc/utmp

SEE ALSO

ps(1), who(1) in the *User's Reference Manual.*

NAME

intro — introduction to special files

DESCRIPTION

This section describes various special files that refer to specific hardware peripherals, and GENIX V.3 system device drivers. STREAMS [see *intro*(2)] software drivers, modules and the STREAMS-generic set of *ioctl*(2) system calls are also described.

For hardware related files, the names of the entries are generally derived from names for the hardware, as opposed to the names of the special files themselves. Characteristics of both the hardware device and the corresponding GENIX V.3 system device driver are discussed where applicable.

Disk device file names are in the following format:

/dev/{r}dsk/c#d#s#

where **r** indicates a raw interface to the disk, the **c#** indicates the controller number, **d#** indicates the device attached to the controller and **s#** indicates the section number of the partitioned device.

SEE ALSO

Disk/Tape Management in the *Administrator's Guide.*

NAME

clone — open any minor device on a STREAMS driver

DESCRIPTION

Clone is a STREAMS software driver that finds and opens an unused minor device on another STREAMS driver. The minor device passed to *clone* during the open is interpreted as the major device number of another STREAMS driver for which an unused minor device is to be obtained. Each such open results in a separate *stream* to a previously unused minor device.

The *clone* driver consists solely of an open function. This open function performs all of the necessary work so that subsequent system calls (including *close*(2)) require no further involvement of *clone*.

Clone will generate an ENXIO error, without opening the device, if the minor device number provided does not correspond to a valid major device, or if the driver indicated is not a STREAMS driver.

CAVEATS

Multiple opens of the same minor device cannot be done through the *clone* interface. Executing *stat(2)* on the file system node for a cloned device yields a different result from executing *fstat(2)* using a file descriptor obtained from opening the node.

SEE ALSO

log(7).
STREAMS Programmer's Guide.

NAME

console — console interface

DESCRIPTION

The console provides the operator interface to the *Series 32000* computer.

The file */dev/console* is the system console, and refers to an asynchronous serial data line originating from the system board. This special file implements the features described in *termio*(7).

The file */dev/contty* refers to a second asynchronous serial data line originating from the system board. This special file implements the features described in *termio*(7).

FILES

/dev/console
/dev/contty

SEE ALSO

termio(7).

NAME

dsd — *Series 32000* computer Integral Disk Subsystem

DESCRIPTION

The *Series 32000* computer integral disk subsystem may consist of one or two units in two sizes; 40M and 140M. The files **/dev/dsk/c0d***n***s0** ... **/dev/dsk/c0d***n***sF** refer to sections of the drive unit number *n*. This slicing allows the media to be broken up into more manageable pieces.

The **/dev/dsk** files provide access to the disk via the system's normal buffering mechanism. There is also a "raw" interface which provides for direct transfer of a specified number of bytes between the disk and a location in the user's address space. The names of the raw disk files begin with **/dev/rdsk** and end with a number which selects the same disk section as the corresponding **/dev/dsk** file. In raw I/O the read or write must begin on a word boundary; transfer counts can be as small as a single byte.

FILES

/dev/dsk*, /dev/rdsk*

SEE ALSO

Appendix A of the *Administrator's Guide* for tables showing the default disk partitioning of a variety of manufacturers' hard disk units.

NAME

hdelog — hard disk error log interface file

DESCRIPTION

The file */dev/hdelog* is a special file that provides access to the disk error logging mechanism, the equipped disk table, and the disk drivers of the equipped disks for doing physical (non-partitioned) disk I/O. It is an internal interface of bad block handling and a few other disk utilities and is not intended to be used directly by users. You must be superuser to use it.

FILES

/dev/hdelog

SEE ALSO

hdeadd(1M), hdefix(1M), hdelogger(1M), hdeupdate(1M).

NAME

log — interface to STREAMS error logging and event tracing

DESCRIPTION

Log is a STREAMS software device driver that provides an interface for the STREAMS error logging and event tracing processes (*strerr*(1M), *strace*(1M)). *Log* presents two separate interfaces: a function call interface in the kernel through which STREAMS drivers and modules submit *log* messages; and a subset of *ioctl*(2) system calls and STREAMS messages for interaction with a user level error logger, a trace logger, or processes that need to submit their own *log* messages.

Kernel Interface

Log messages are generated within the kernel by calls to the function *strlog*:

```
strlog(mid, sid, level, flags, fmt, arg1, ...)
short mid, sid;
char level;
ushort flags;
char *fmt;
unsigned arg1;
```

Required definitions are contained in **<sys/strlog.h>** and **<sys/log.h>**. *Mid* is the STREAMS module id number for the module or driver submitting the *log* message. *Sid* is an internal sub-id number usually used to identify a particular minor device of a driver. *Level* is a tracing level that allows for selective screening out of low priority messages from the tracer. *Flags* are any combination of SL_ERROR (the message is for the error logger), SL_TRACE (the message is for the tracer), SL_FATAL (advisory notification of a fatal error), and SL_NOTIFY (request that a copy of the message be mailed to the system administrator). *Fmt* is a *printf*(3S) style format string, except that %s, %e, %E, %g, and %G conversion specifications are not handled. Up to NLOGARGS (currently 3) numeric or character arguments can be provided.

User Interface

Log is opened via the clone interface, **/dev/log**. Each open of **/dev/log** obtains a separate *stream* to *log*. In order to receive *log* messages, a process must first notify *log* whether it is an error logger or trace logger via a STREAMS I_STR *ioctl* call (see below). For the error logger, the I_STR *ioctl* has an ic_cmd field of I_ERRLOG, with no accompanying data. For the trace logger, the *ioctl* has an ic_cmd field of I_TRCLOG, and must be accompanied by a data buffer containing an array of one or more struct trace_ids elements. Each trace_ids structure specifies an *mid, sid,* and *level* from which message will be accepted. *Strlog* will accept messages whose *mid* and *sid* exactly match those in the trace_ids structure, and whose level is less than or equal to the level given in the trace_ids structure. A value of -1 in any of the fields of the trace_ids structure indicates that any value is accepted for that field.

At most one trace logger and one error logger can be active at a time. Once the logger process has identified itself via the *ioctl* call, *log* will begin sending up messages subject to the restrictions noted above. These messages are obtained via the *getmsg(2)* system call. The control part of this message contains a log_ctl structure, which specifies the *mid, sid, level, flags,* time in ticks since boot that the message was submitted, the corresponding time in seconds since Jan. 1, 1970, and a sequence number. The time in seconds since 1970 is provided so that the date and time of the message can be easily computed, and the time in ticks since boot is provided so that the relative timing of *log* messages can be determined.

Different sequence numbers are maintained for the error and trace logging *streams,* and are provided so that gaps in the sequence of messages can be determined (during times of high message traffic some messages may not be delivered by the logger to avoid hogging system resources). The data part of the message contains the unexpanded text of the format string (null terminated), followed by NLOGARGS words for the arguments to the format string, aligned on the first word boundary following the format string.

A process may also send a message of the same structure to *log*, even if it is not an error or trace logger. The only fields of the log_ctl structure in the control part of the message that are accepted are the level and flags fields; all other fields are filled in by *log* before being forwarded to the appropriate logger. The data portion must contain a null terminated format string, and any arguments (up to NLOGARGS) must be packed one word each, on the next word boundary following the end of the format string.

Attempting to issue an I_TRCLOG or I_ERRLOG when a logging process of the given type already exists will result in the error ENXIO being returned. Similarly, ENXIO is returned for I_TRCLOG *ioctls* without any trace_ids structures, or for any unrecognized I_STR *ioctl* calls. Incorrectly formatted *log* messages sent to the driver by a user process are silently ignored (no error results).

EXAMPLES

Example of I_ERRLOG notification.

```
struct strioctl ioc;

ioc.ic_cmd = I_ERRLOG;
ioc.ic_timout = 0;              /* default timeout (15 secs.) */
ioc.ic_len = 0;
ioc.ic_dp = NULL;

ioctl(log, I_STR, &ioc);
```

Example of I_TRCLOG notification.

```
struct trace_ids tid[2];

tid[0].ti_mid = 2;
tid[0].ti_sid = 0;
tid[0].ti_level = 1;

tid[1].ti_mid = 1002;
tid[1].ti_sid = -1;         /* any sub-id will be allowed */
tid[1].ti_level = -1;       /* any level will be allowed */

ioc.ic_cmd = I_TRCLOG;
ioc.ic_timout = 0;
ioc.ic_len = 2 * sizeof(struct trace_ids);
ioc.ic_dp = (char *)tid;

ioctl(log, I_STR, &ioc);
```

Example of submitting a *log* message (no arguments).

```
struct strbuf ctl, dat;
struct log_ctl lc;
char *message = "Don't forget to pick up some milk on the way home";

ctl.len = ctl.maxlen = sizeof(lc);
ctl.buf = (char *)&lc;

dat.len = dat.maxlen = strlen(message);
dat.buf = message;

lc.level = 0;
lc.flags = SL_ERROR|SL_NOTIFY;

putmsg(log, &ctl, &dat, 0);
```

FILES

/dev/log, <sys/log.h>, <sys/strlog.h>

SEE ALSO

strace(1M), strerr(1M), clone(7).
intro(2), getmsg(2), putmsg(2) in the *Programmer's Reference Manual.*
STREAMS Programmer's Guide.

NAME

mem, kmem — core memory

DESCRIPTION

The file */dev/mem* is a special file that is an image of the core memory of the computer. It may be used, for example, to examine, and even to patch the system.

Byte addresses in */dev/mem* are interpreted as 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.

The file */dev/kmem* is the same as */dev/mem* except that kernel virtual memory rather than physical memory is accessed.

The per-process data for the current process begins at 0x80880000.

FILES

/dev/mem
/dev/kmem

WARNING

Some of */dev/kmem* cannot be read because of write-only addresses or unequipped memory addresses.

NAME

mt — tape interface

DESCRIPTION

The files **mt/ctape?** and **rmt/ctape?** refer to cartridge tape controllers (CTC) and associated tape drives. These special device files are linked to the standard CTC **SA/ctape?** and **rSA/ctape?** files, respectively.

The *finc*(1M), *frec*(1M), and *labelit*(1M) commands require these magnetic tape file names to work correctly with the CTC. No other CTC commands require these file names.

FILES

/dev/mt/ctape*
/dev/rmt/ctape*

SEE ALSO

finc(1M), frec(1M), labelit(1M).

NAME

null — the null file

DESCRIPTION

Data written on the null special file, */dev/null*, is discarded.

Reads from a null special file always return 0 bytes.

FILES

/dev/null

NAME

prf — operating system profiler

DESCRIPTION

The special file */dev/prf* provides access to activity information in the operating system. Writing the file loads the measurement facility with text addresses to be monitored. Reading the file returns these addresses and a set of counters indicative of activity between adjacent text addresses.

The recording mechanism is driven by the system clock and samples the program counter at line frequency. Samples that catch the operating system are matched against the stored text addresses and increment corresponding counters for later processing.

The file */dev/prf* is a pseudo-device with no associated hardware.

FILES

/dev/prf

SEE ALSO

profiler(1M).

NAME

SA — devices administered by System Administration

DESCRIPTION

The files in the directories **/dev/SA** (for block devices) and the **/dev/rSA** (for raw devices) are used by System Administration to access the devices on which it operates. For devices that support more than one partition (like disks) the **/dev/(r)SA** entry is linked to the partition that spans the entire device. Not all **/dev/(r)SA** entries are used by all System Administration commands.

FILES

/dev/SA
/dev/rSA

SEE ALSO

sysadm(1) in the *User's Reference Manual.*

NAME

streamio — STREAMS ioctl commands

SYNOPSIS

#include <stropts.h>
int ioctl (fildes, command, arg)
int fildes, command;

DESCRIPTION

STREAMS [see *intro*(2)] ioctl commands are a subset of *ioctl*(2) system calls which perform a variety of control functions on *streams*. The arguments *command* and *arg* are passed to the file designated by *fildes* and are interpreted by the *stream head*. Certain combinations of these arguments may be passed to a module or driver in the *stream*.

Fildes is an open file descriptor that refers to a *stream*. *Command* determines the control function to be performed as described below. *Arg* represents additional information that is needed by this command. The type of *arg* depends upon the command, but it is generally an integer or a pointer to a *command*-specific data structure.

Since these STREAMS commands are a subset of *ioctl*, they are subject to the errors described there. In addition to those errors, the call will fail with *errno* set to EINVAL, without processing a control function, if the *stream* referenced by *fildes* is linked below a multiplexor, or if *command* is not a valid value for a *stream*.

Also, as described in *ioctl*, STREAMS modules and drivers can detect errors. In this case, the module or driver sends an error message to the *stream head* containing an error value. This causes subsequent system calls to fail with *errno* set to this value.

COMMAND FUNCTIONS

The following *ioctl* commands, with error values indicated, are applicable to all STREAMS files:

I_PUSH Pushes the module whose name is pointed to by *arg* onto the top of the current *stream*, just below the *stream head*. It then calls the open routine of the newly-pushed module. On failure, *errno* is set to one of the following values:

[EINVAL]	Invalid module name.
[EFAULT]	*Arg* points outside the allocated address space.
[ENXIO]	Open routine of new module failed.
[ENXIO]	Hangup received on *fildes*.

I_POP Removes the module just below the *stream head* of the *stream* pointed to by *fildes*. *Arg* should be 0 in an I_POP request. On failure, *errno* is set to one of the following values:

[EINVAL]	No module present in the *stream*.
[ENXIO]	Hangup received on *fildes*.

I_LOOK Retrieves the name of the module just below the *stream head* of the *stream* pointed to by *fildes*, and places it in a null terminated character string pointed at by *arg*. The buffer pointed to by *arg* should be at least FMNAMESZ+1 bytes long. An "#include <sys/conf.h>" declaration is required. On failure, *errno* is set to one of the following values:

[EFAULT]	*Arg* points outside the allocated address space.
[EINVAL]	No module present in *stream*.

I_FLUSH This request flushes all input and/or output queues, depending on the value of *arg*. Legal *arg* values are:

FLUSHR	Flush read queues.
FLUSHW	Flush write queues.
FLUSHRW	Flush read and write queues.

On failure, *errno* is set to one of the following values:

[EAGAIN]	Unable to allocate buffers for flush message.
[EINVAL]	Invalid *arg* value.
[ENXIO]	Hangup receivcd on *fildes*.

I_SETSIG Informs the *stream head* that the user wishes the kernel to issue the SIGPOLL signal [see *signal*(2) and *sigset*(2)] when a particular event has occurred on the *stream* associated with *fildes*. I_SETSIG supports an asynchronous processing capability in STREAMS. The value of *arg* is a bitmask that specifies the events for which the user should be signaled. It is the bitwise-OR of any combination of the following constants:

S_INPUT	A non-priority message has arrived on a *stream head* read queue, and no other messages existed on that queue before this message was placed there. This is set even if the message is of zero length.
S_HIPRI	A priority message is present on the *stream head* read queue. This is set even if the message is of zero length.
S_OUTPUT	The write queue just below the *stream head* is no longer full. This notifies the user that there is room on the queue for sending (or writing) data downstream.
S_MSG	A STREAMS signal message that contains the SIGPOLL signal has reached the front of the *stream head* read queue.

A user process may choose to be signaled only of priority messages by setting the *arg* bitmask to the value S_HIPRI.

Processes that wish to receive SIGPOLL signals must explicitly register to receive them using I_SETSIG. If several processes register to receive this signal for the same event on the same Stream, each process will be signaled when the event occurs.

If the value of *arg* is zero, the calling process will be unregistered and will not receive further SIGPOLL signals. On failure, *errno* is set to one of the following values:

[EINVAL]	*Arg* value is invalid or *arg* is zero and process is not registered to receive the SIGPOLL signal.
[EAGAIN]	Allocation of a data structure to store the signal request failed.

I_GETSIG Returns the events for which the calling process is currently registered to be sent a SIGPOLL signal. The events are returned as a bitmask pointed to by *arg*, where the events are those specified in the description of I_SETSIG above. On failure, *errno* is set to one of the following values:

[EINVAL]	Process not registered to receive the SIGPOLL signal.
[EFAULT]	*Arg* points outside the allocated address space.

I_FIND This request compares the names of all modules currently present in the *stream* to the name pointed to by *arg*, and returns 1 if the named module is present in the *stream*. It returns 0 if the named module is not present. On failure, *errno* is set to one of the following values:

[EFAULT]	*Arg* points outside the allocated address space.
[EINVAL]	*Arg* does not contain a valid module name.

I_PEEK This request allows a user to retrieve the information in the first message on the *stream head* read queue without taking the message off the queue. *Arg* points to a *strpeek* structure which contains the following members:

```
struct strbuf   ctlbuf;
struct strbuf   databuf;
long            flags;
```

The *maxlen* field in the *ctlbuf* and *databuf strbuf* structures [see *getmsg*(2)] must be set to the number of bytes of control information and/or data information, respectively, to retrieve. If the user sets *flags* to RS_HIPRI, I_PEEK will only look for a priority message on the *stream head* read queue.

I_PEEK returns 1 if a message was retrieved, and returns 0 if no message was found on the *stream head* read queue, or if the RS_HIPRI flag was set in *flags* and a priority message was not present on the *stream head* read queue. It does not wait for a message to arrive. On return, *ctlbuf* specifies information in the control buffer, *databuf* specifies information in the data buffer, and *flags* contains the value 0 or RS_HIPRI. On failure, *errno* is set to the following value:

[EFAULT]	*Arg* points, or the buffer area specified in *ctlbuf* or *databuf* is, outside the allocated address space.

I_SRDOPT Sets the read mode using the value of the argument *arg*. Legal *arg* values are:

RNORM	Byte-stream mode, the default.
RMSGD	Message-discard mode.
RMSGN	Message-nondiscard mode.

Read modes are described in *read*(2). On failure, *errno* is set to the following value:

[EINVAL]	*Arg* is not one of the above legal values.

I_GRDOPT Returns the current read mode setting in an *int* pointed to by the argument *arg*. Read modes are described in *read*(2). On failure, *errno* is set to the following value:

[EFAULT]	*Arg* points outside the allocated address space.

I_NREAD Counts the number of data bytes in data blocks in the first message on the *stream head* read queue, and places this value in the location pointed to by *arg*. The return value for the command is the number of messages on the *stream head* read queue. For example, if zero is returned in *arg*, but the *ioctl* return value is greater than zero, this indicates that a zero-length message is next on the queue. On failure, *errno* is set to the following value:

[EFAULT]	*Arg* points outside the allocated address space.

I_FDINSERT creates a message from user specified buffer(s), adds information about another *stream* and sends the message downstream. The message contains a control part and an optional data part. The data and control parts to be sent are distinguished by placement in separate buffers, as described below.

Arg points to a *strfdinsert* structure which contains the following members:

```
struct strbuf   ctlbuf;
struct strbuf   databuf;
long            flags;
int             fd;
int             offset;
```

The *len* field in the *ctlbuf strbuf* structure [see *putmsg*(2)] must be set to the size of a pointer plus the number of bytes of control information to be sent with the message. *Fd* specifies the file descriptor of the other *stream* and *offset*, which must be word-aligned, specifies the number of bytes beyond the beginning of the control buffer where I_FDINSERT will store a pointer to the *fd stream*'s driver read queue structure. The *len* field in the *databuf strbuf* structure must be set to the number of bytes of data information to be sent with the message or zero if no data part is to be sent.

Flags specifies the type of message to be created. A non-priority message is created if *flags* is set to 0, and a priority message is created if *flags* is set to RS_HIPRI. For non-priority messages, I_FDINSERT will block if the *stream* write queue is full due to internal flow control conditions. For priority messages, I_FDINSERT does not block on this condition. For non-priority messages, I_FDINSERT does not block when the write queue is full and O_NDELAY is set. Instead, it fails and sets *errno* to EAGAIN.

I_FDINSERT also blocks, unless prevented by lack of internal resources, waiting for the availability of message blocks in the *stream*, regardless of priority or whether O_NDELAY has been specified. No partial message is sent. On failure, *errno* is set to one of the following values:

[EAGAIN]	A non-priority message was specified, the O_NDELAY flag is set, and the *stream* write queue is full due to internal flow control conditions.
[EAGAIN]	Buffers could not be allocated for the message that was to be created.
[EFAULT]	*Arg* points, or the buffer area specified in *ctlbuf* or *databuf* is, outside the allocated address space.
[EINVAL]	One of the following: *fd* in the *strfdinsert* structure is not a valid, open *stream* file descriptor; the size of a pointer plus *offset* is greater than the *len* field for the buffer specified through *ctlptr*; *offset* does not specify a properly-aligned location in the data buffer; an undefined value is stored in *flags*.
[ENXIO]	Hangup received on *fildes*.
[ERANGE]	The *len* field for the buffer specified through *databuf* does not fall within the range specified by the maximum and minimum packet sizes of the topmost *stream* module, or the *len* field for the buffer specified through *databuf* is larger than the maximum configured size of the data part of a message, or the *len* field for the buffer specified through *ctlbuf* is larger than the maximum configured size of the control part of a message.

I_STR Constructs an internal STREAMS ioctl message from the data pointed to by *arg*, and sends that message downstream.

This mechanism is provided to send user *ioctl* requests to downstream modules and drivers. It allows information to be sent with the ioctl, and will return to the user any information sent upstream by the downstream recipient. I_STR blocks until the system responds with either a positive or negative acknowledgement message, or until the request "times out" after some period of time. If the request times out, it fails with *errno* set to ETIME.

At most, one I_STR can be active on a *stream*. Further I_STR calls will block until the active I_STR completes at the *stream head*. The default timeout interval for these requests is 15 seconds. The O_NDELAY [see *open*(2)] flag has no effect on this call.

To send requests downstream, *arg* must point to a *strioctl* structure which contains the following members:

```
int     ic_cmd;         /* downstream command */
int     ic_timout;      /* ACK/NAK timeout */
int     ic_len;         /* length of data arg */
char    *ic_dp;         /* ptr to data arg */
```

Ic_cmd is the internal ioctl command intended for a downstream module or driver and *ic_timout* is the number of seconds (-1 = infinite, 0 = use default, >0 = as specified) an I_STR request will wait for acknowledgement before timing out. *Ic_len* is the number of bytes in the data argument and *ic_dp* is a pointer to the data argument. The *ic_len* field has two uses: on input, it contains the length of the data argument passed in, and on return from the command, it contains the number of bytes being returned to the user (the buffer pointed to by *ic_dp* should be large enough to contain the maximum amount of data that any module or the driver in the *stream* can return).

The *stream head* will convert the information pointed to by the *strioctl* structure to an internal ioctl command message and send it downstream. On failure, *errno* is set to one of the following values:

[EAGAIN]	Unable to allocate buffers for the *ioctl* message.
[EFAULT]	*Arg* points, or the buffer area specified by *ic_dp* and *ic_len* (separately for data sent and data returned) is, outside the allocated address space.
[EINVAL]	*Ic_len* is less than 0 or *ic_len* is larger than the maximum configured size of the data part of a message or *ic_timout* is less than -1.
[ENXIO]	Hangup received on *fildes*.
[ETIME]	A downstream *ioctl* timed out before acknowledgement was received.

An I_STR can also fail while waiting for an acknowledgement if a message indicating an error or a hangup is received at the *stream head*. In addition, an error code can be returned in the positive or negative acknowledgement message, in the event the ioctl command sent downstream fails. For these cases, I_STR will fail with *errno* set to the value in the message.

I_SENDFD Requests the *stream* associated with *fildes* to send a message, containing a file pointer, to the *stream head* at the other end of a *stream* pipe. The file pointer corresponds to *arg*, which must be an integer file descriptor.

I_SENDFD converts *arg* into the corresponding system file pointer. It allocates a message block and inserts the file pointer in the block. The user ID and group ID associated with the sending process are also inserted. This message is placed directly on the read queue [see *intro*(2)] of the *stream head* at the other end of the *stream* pipe to which it is connected. On failure, *errno* is set to one of the following values:

[EAGAIN]	The sending *stream* is unable to allocate a message block to contain the file pointer.
[EAGAIN]	The read queue of the receiving *stream head* is full and cannot accept the message sent by I_SENDFD.
[EBADF]	*arg* is not a valid, open file descriptor.
[EINVAL]	*fildes* is not connected to a *stream* pipe.
[ENXIO]	Hangup received on *fildes*.

I_RECVFD Retrieves the file descriptor associated with the message sent by an I_SENDFD *ioctl* over a *stream* pipe. *arg* is a pointer to a data buffer large enough to hold an *strrecvfd* data structure containing the following members:

```
int fd;
unsigned short uid;
unsigned short gid;
char fill[8];
```

Fd is an integer file descriptor. *Uid* and *gid* are the user ID and group ID, respectively, of the sending *stream*.

If O_NDELAY is not set [see *open*(2)], I_RECVFD will block until a message is present at the *stream head*. If O_NDELAY is set, I_RECVFD will fail with *errno* set to EAGAIN if no message is present at the *stream head*.

If the message at the *stream head* is a message sent by an I_SENDFD, a new user file descriptor is allocated for the file pointer contained in the message. The new file descriptor is placed in the *fd* field of the *strrecvfd* structure. The structure is copied into the user data buffer pointed to by *arg*. On failure, *errno* is set to one of the following values:

[EAGAIN]	A message was not present at the *stream head* read queue, and the O_NDELAY flag is set.
[EBADMSG]	The message at the *stream head* read queue was not a message containing a passed file descriptor.
[EFAULT]	*Arg* points outside the allocated address space.
[EMFILE]	NOFILES file descriptors are currently open.
[ENXIO]	Hangup received on *fildes*.

The following two commands are used for connecting and disconnecting multiplexed STREAMS configurations.

I_LINK Connects two *streams*, where *fildes* is the file descriptor of the *stream* connected to the multiplexing driver, and *arg* is the file descriptor of the *stream* connected to another driver. The *stream* designated by *arg* gets connected below the multiplexing driver. I_LINK requires the multiplexing driver to send an acknowledgement message to the *stream head* regarding the linking operation. This call returns a multiplexor ID number (an identifier used to disconnect the multiplexor, see I_UNLINK) on success, and a -1 on failure. On failure, *errno* is set to one of the following values:

[ENXIO]	Hangup received on *fildes.*
[ETIME]	Time out before acknowledgement message was received at *stream head.*
[EAGAIN]	Unable to allocate STREAMS storage to perform the I_LINK.
[EBADF]	*Arg* is not a valid, open file descriptor.
[EINVAL]	*Fildes stream* does not support multiplexing.
[EINVAL]	*Arg* is not a *stream,* or is already linked under a multiplexor.
[EINVAL]	The specified link operation would cause a "cycle" in the resulting configuration; that is, if a given *stream head* is linked into a multiplexing configuration in more than one place.

An I_LINK can also fail while waiting for the multiplexing driver to acknowledge the link request, if a message indicating an error or a hangup is received at the *stream head* of *fildes.* In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, I_LINK will fail with *errno* set to the value in the message.

I_UNLINK Disconnects the two *streams* specified by *fildes* and *arg. Fildes* is the file descriptor of the *stream* connected to the multiplexing driver. *Arg* is the multiplexor ID number that was returned by the *ioctl* I_LINK command when a *stream* was linked below the multiplexing driver. If *arg* is -1, then all Streams which were linked to *fildes* are disconnected. As in I_LINK, this command requires the multiplexing driver to acknowledge the unlink. On failure, *errno* is set to one of the following values:

[ENXIO]	Hangup received on *fildes.*
[ETIME]	Time out before acknowledgement message was received at *stream head.*
[EAGAIN]	Unable to allocate buffers for the acknowledgement message.
[EINVAL]	Invalid multiplexor ID number.

An I_UNLINK can also fail while waiting for the multiplexing driver to acknowledge the link request, if a message indicating an error or a hangup is received at the *stream head* of *fildes.* In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, I_UNLINK will fail with *errno* set to the value in the message.

SEE ALSO

close(2), fcntl(2), intro(2), ioctl(2), open(2), read(2), getmsg(2), poll(2), putmsg(2), signal(2), sigset(2), write(2) in the *Programmer's Reference Manual.*
STREAMS Programmer's Guide.
STREAMS Primer.

DIAGNOSTICS

Unless specified otherwise above, the return value from *ioctl* is 0 upon success and -1 upon failure with *errno* set as indicated.

NAME

sxt — pseudo-device driver

DESCRIPTION

The special file */dev/sxt* is a pseudo-device driver that interposes a discipline between the standard *tty* line disciplines and a real device driver. The standard disciplines manipulate *virtual tty* structures (channels) declared by the */dev/sxt* driver. */Dev/sxt* acts as a discipline manipulating a *real tty* structure declared by a real device driver. The */dev/sxt* driver is currently only used by the *shl* (1) command.

Virtual ttys are named by inodes in the subdirectory **/dev/sxt** and are allocated in groups of up to eight. To allocate a group, a program should exclusively open a file with a name of the form **/dev/sxt/??0** (channel 0) and then execute a SXTIOCLINK *ioctl* call to initiate the multiplexing.

Only one channel, the *controlling* channel, can receive input from the keyboard at a time; others attempting to read will be blocked.

There are two groups of *ioctl*(2) commands supported by *sxt*. The first group contains the standard *ioctl* commands described in *termio*(7), with the addition of the following:

TIOCEXCL	Set *exclusive use* mode: no further opens are permitted until the file has been closed.
TIOCNXCL	Reset *exclusive use* mode: further opens are once again permitted.

The second group are commands to *sxt* itself. Some of these may only be executed on channel 0.

SXTIOCLINK
: Allocate a channel group and multiplex the virtual ttys onto the real tty. The argument is the number of channels to allocate. This command may only be executed on channel 0. Possible errors include:

EINVAL	The argument is out of range.
ENOTTY	The command was not issued from a real tty.
ENXIO	*Linesw* is not configured with *sxt*.
EBUSY	An SXTIOCLINK command has already been issued for this real *tty*.
ENOMEM	There is no system memory available for allocating the virtual tty structures.
EBADF	Channel 0 was not opened before this call.

SXTIOCSWTCH
: Set the controlling channel. Possible errors include:

EINVAL	An invalid channel number was given.
EPERM	The command was not executed from channel 0.

SXTIOCWF
: Cause a channel to wait until it is the controlling channel. This command will return the error, *EINVAL*, if an invalid channel number is given.

SXTIOCUBLK
: Turn off the **loblk** control flag in the virtual tty of the indicated channel. The error *EINVAL* will be returned if an invalid number or channel 0 is given.

SXTIOCSTAT
: Get the status (blocked on input or output) of each channel and store in the *sxtblock* structure referenced by the argument. The error *EFAULT* will be returned if the structure cannot be written.

SXTIOCTRACE	Enable tracing. Tracing information is written to the console on the 3B2 Computer. This command has no effect if tracing is not configured.
SXTIOCNOTRACE	Disable tracing. This command has no effect if tracing is not configured.

FILES

/dev/sxt/??[0-7]	Virtual tty devices

SEE ALSO

termio(7).
shl(1), stty(1) in the *User's Reference Manual.*
ioctl(2), open(2) in the *Programmer's Reference Manual.*

NAME

termio — general terminal interface

DESCRIPTION

All of the asynchronous communications ports use the same general interface, no matter what hardware is involved. The remainder of this section discusses the common features of this interface.

When a terminal file is opened, it normally causes the process to wait until a connection is established. In practice, users' programs seldom open terminal files; they are opened by *getty* and become a user's standard input, output, and error files. The very first terminal file opened by the process group leader of a terminal file not already associated with a process group becomes the *control terminal* for that process group. The control terminal plays a special role in handling quit and interrupt signals, as discussed below. The control terminal is inherited by a child process during a *fork*(2). A process can break this association by changing its process group using *setpgrp*(2).

A terminal associated with one of these files 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 full, 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. When the input limit is reached, the buffer is flushed and all the saved characters are thrown away without notice.

Normally, terminal input is processed in units of lines. A line is delimited by a new-line (ASCII LF) character, an end-of-file (ASCII EOT) character, or an end-of-line character. This means that a program attempting to read will be suspended until an entire line has been typed. Also, 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, erase and kill processing is normally done. By default, the character # erases the last character typed, except that it will not erase beyond the beginning of the line. By default, the character @ kills (deletes) the entire input line, and optionally outputs a new-line character. Both these characters operate on a key-stroke basis, independently of any backspacing or tabbing that may have been done. Both the erase and kill characters may be entered literally by preceding them with the escape character (\). In this case the escape character is not read. The erase and kill characters may be changed.

Certain characters have special functions on input. These functions and their default character values are summarized as follows:

INTR	(Rubout or ASCII DEL) generates an *interrupt* signal which is sent to all processes with the associated control terminal. Normally, each such process is forced to terminate, but arrangements may be made either to ignore the signal or to receive a trap to an agreed-upon location; see *signal*(2).
QUIT	(Control-| or ASCII FS) generates a *quit* signal. Its treatment is identical to the interrupt signal except that, unless a receiving process has made other arrangements, it will not only be terminated but a core image file (called **core**) will be created in the current working directory.
SWTCH	(Control-z or ASCII SUB) is used by the job control facility, *shl*, to change the current layer to the control layer.
ERASE	(#) erases the preceding character. It will not erase beyond the start of a line, as delimited by a NL, EOF, or EOL character.
KILL	(@) deletes the entire line, as delimited by a NL, EOF, or EOL character.

EOF	(Control-d or ASCII EOT) may be used to generate an end-of-file from a terminal. When received, all the characters waiting to be read are immediately passed to the program, without waiting for a new-line, and the EOF is discarded. Thus, if there are no characters waiting, which is to say the EOF occurred at the beginning of a line, zero characters will be passed back, which is the standard end-of-file indication.
NL	(ASCII LF) is the normal line delimiter. It can not be changed or escaped.
EOL	(ASCII NUL) is an additional line delimiter, like NL. It is not normally used.
EOL2	is another additional line delimiter.
STOP	(Control-s or ASCII DC3) can be used to temporarily suspend output. It is useful with CRT terminals to prevent output from disappearing before it can be read. While output is suspended, STOP characters are ignored and not read.
START	(Control-q or ASCII DC1) is used to resume output which has been suspended by a STOP character. While output is not suspended, START characters are ignored and not read. The start/stop characters can not be changed or escaped.

The character values for INTR, QUIT, SWTCH, ERASE, KILL, EOF, and EOL may be changed to suit individual tastes. The ERASE, KILL, and EOF characters may be escaped by a preceding \ character, in which case no special function is done.

When the carrier signal from the data-set drops, a *hang-up* signal is sent to all processes that have this terminal as the control terminal. Unless other arrangements have been made, this signal causes the processes to terminate. If the hang-up signal is ignored, any subsequent read returns with an end-of-file indication. Thus, programs that read a terminal and test for end-of-file can terminate appropriately when hung up on.

When one or more characters are written, they are transmitted to the terminal as soon as previously-written characters have finished typing. Input characters are echoed by putting them in the output queue as they arrive. If 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.

Several *ioctl*(2) system calls apply to terminal files. The primary calls use the following structure, defined in **<termio.h>**:

```
#define  NCC      8
struct   termio {
         unsigned  short  c_iflag;     /* input modes */
         unsigned  short  c_oflag;     /* output modes */
         unsigned  short  c_cflag;     /* control modes */
         unsigned  short  c_lflag;     /* local modes */
         char             c_line;      /* line discipline */
         unsigned  char   c_cc[NCC];   /* control chars */
};
```

The special control characters are defined by the array *c_cc*. The relative positions and initial values for each function are as follows:

0	VINTR	DEL
1	VQUIT	FS
2	VERASE	#
3	VKILL	@
4	VEOF	EOT
5	VEOL	NUL
6	reserved	
7	SWTCH	

The *c_iflag* field describes the basic terminal input control:

IGNBRK	0000001	Ignore break condition.
BRKINT	0000002	Signal interrupt on break.
IGNPAR	0000004	Ignore characters with parity errors.
PARMRK	0000010	Mark parity errors.
INPCK	0000020	Enable input parity check.
ISTRIP	0000040	Strip character.
INLCR	0000100	Map NL to CR on input.
IGNCR	0000200	Ignore CR.
ICRNL	0000400	Map CR to NL on input.
IUCLC	0001000	Map upper-case to lower-case on input.
IXON	0002000	Enable start/stop output control.
IXANY	0004000	Enable any character to restart output.
IXOFF	0010000	Enable start/stop input control.

If IGNBRK is set, the break condition (a character framing error with data all zeros) is ignored, that is, not put on the input queue and therefore not read by any process. Otherwise if BRKINT is set, the break condition will generate an interrupt signal and flush both the input and output queues. If IGNPAR is set, characters with other framing and parity errors are ignored.

If PARMRK is set, a character with a framing or parity error which is not ignored is read as the three-character sequence: 0377, 0, X, where X is the data of the character received in error. To avoid ambiguity in this case, if ISTRIP is not set, a valid character of 0377 is read as 0377, 0377. If PARMRK is not set, a framing or parity error which is not ignored is read as the character NUL (0).

If INPCK is set, input parity checking is enabled. If INPCK is not set, input parity checking is disabled. This allows output parity generation without input parity errors.

If ISTRIP is set, valid input characters are first stripped to 7-bits, otherwise all 8-bits are processed.

If INLCR is set, a received NL character is translated into a CR character. If IGNCR is set, a received CR character is ignored (not read). Otherwise if ICRNL is set, a received CR character is translated into a NL character.

If IUCLC is set, a received upper-case alphabetic character is translated into the corresponding lower-case character.

If IXON is set, start/stop output control is enabled. A received STOP character will suspend output and a received START character will restart output. All start/stop characters are ignored and not read. If IXANY is set, any input character, will restart output which has been suspended.

If IXOFF is set, the system will transmit START/STOP characters when the input queue is nearly empty/full.

The initial input control value is all-bits-clear.

The *c_oflag* field specifies the system treatment of output:

OPOST	0000001	Postprocess output.
OLCUC	0000002	Map lower case to upper on output.
ONLCR	0000004	Map NL to CR-NL on output.
OCRNL	0000010	Map CR to NL on output.
ONOCR	0000020	No CR output at column 0.
ONLRET	0000040	NL performs CR function.
OFILL	0000100	Use fill characters for delay.
OFDEL	0000200	Fill is DEL, else NUL.

NLDLY	0000400	Select new-line delays:
NL0	0	
NL1	0000400	
CRDLY	0003000	Select carriage-return delays:
CR0	0	
CR1	0001000	
CR2	0002000	
CR3	0003000	
TABDLY	0014000	Select horizontal-tab delays:
TAB0	0	
TAB1	0004000	
TAB2	0010000	
TAB3	0014000	Expand tabs to spaces.
BSDLY	0020000	Select backspace delays:
BS0	0	
BS1	0020000	
VTDLY	0040000	Select vertical-tab delays:
VT0	0	
VT1	0040000	
FFDLY	0100000	Select form-feed delays:
FF0	0	
FF1	0100000	

If OPOST is set, output characters are post-processed as indicated by the remaining flags, otherwise characters are transmitted without change.

If OLCUC is set, a lower-case alphabetic character is transmitted as the corresponding upper-case character. This function is often used in conjunction with IUCLC.

If ONLCR is set, the NL character is transmitted as the CR-NL character pair. If OCRNL is set, the CR character is transmitted as the NL character. If ONOCR is set, no CR character is transmitted when at column 0 (first position). If ONLRET is set, the NL character is assumed to do the carriage-return function; the column pointer will be set to 0 and the delays specified for CR will be used. Otherwise the NL character is assumed to do just the line-feed function; the column pointer will remain unchanged. The column pointer is also set to 0 if the CR character is actually transmitted.

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. If OFILL is set, fill characters will be transmitted for delay instead of a timed delay. This is useful for high baud rate terminals which need only a minimal delay. If OFDEL is set, the fill character is DEL, otherwise NUL.

If a form-feed or vertical-tab delay is specified, it lasts for about 2 seconds.

New-line delay lasts about 0.10 seconds. If ONLRET is set, the carriage-return delays are used instead of the new-line delays. If OFILL is set, two fill characters will be transmitted.

Carriage-return delay type 1 is dependent on the current column position, type 2 is about 0.10 seconds, and type 3 is about 0.15 seconds. If OFILL is set, delay type 1 transmits two fill characters, and type 2, four fill characters.

Horizontal-tab delay type 1 is dependent on the current column position. Type 2 is about 0.10 seconds. Type 3 specifies that tabs are to be expanded into spaces. If OFILL is set, two fill characters will be transmitted for any delay.

Backspace delay lasts about 0.05 seconds. If OFILL is set, one fill character will be transmitted.

The actual delays depend on line speed and system load.

The initial output control value is all bits clear.

The *c_cflag* field describes the hardware control of the terminal:

CBAUD	0000017	Baud rate:
B0	0	Hang up
B50	0000001	50 baud
B75	0000002	75 baud
B110	0000003	110 baud
B134	0000004	134 baud
B150	0000005	150 baud
B200	0000006	200 baud
B300	0000007	300 baud
B600	0000010	600 baud
B1200	0000011	1200 baud
B1800	0000012	1800 baud
B2400	0000013	2400 baud
B4800	0000014	4800 baud
B9600	0000015	9600 baud
B19200	0000016	19200 baud
EXTA	0000016	External A
B38400	0000017	38400 baud
EXTB	0000017	External B
CSIZE	0000060	Character size:
CS5	0	5 bits
CS6	0000020	6 bits
CS7	0000040	7 bits
CS8	0000060	8 bits
CSTOPB	0000100	Send two stop bits, else one.
CREAD	0000200	Enable receiver.
PARENB	0000400	Parity enable.
PARODD	0001000	Odd parity, else even.
HUPCL	0002000	Hang up on last close.
CLOCAL	0004000	Local line, else dial-up.
RCV1EN	0010000	
XMT1EN	0020000	
LOBLK	0040000	Block layer output.

The CBAUD bits specify the baud rate. The zero baud rate, B0, is used to hang up the connection. If B0 is specified, the data-terminal-ready signal will not be asserted. Normally, this will disconnect the line. For any particular hardware, impossible speed changes are ignored.

The CSIZE bits specify the character size in bits for both transmission and reception. This size does not include the parity bit, if any. If CSTOPB is set, two stop bits are used, otherwise one stop bit. For example, at 110 baud, two stops bits are required.

If PARENB is set, parity generation and detection is enabled and a parity bit is added to each character. If parity is enabled, the PARODD flag specifies odd parity if set, otherwise even parity is used.

If CREAD is set, the receiver is enabled. Otherwise no characters will be received.

If HUPCL is set, the line will be disconnected when the last process with the line open closes it or terminates. That is, the data-terminal-ready signal will not be asserted.

If CLOCAL is set, the line is assumed to be a local, direct connection with no modem control. Otherwise modem control is assumed.

If LOBLK is set, the output of a job control layer will be blocked when it is not the current layer. Otherwise the output generated by that layer will be multiplexed onto the current layer.

The initial hardware control value after open is B300, CS8, CREAD, HUPCL.

The *c_lflag* field of the argument structure is used by the line discipline to control terminal functions. The basic line discipline (0) provides the following:

ISIG	0000001	Enable signals.
ICANON	0000002	Canonical input (erase and kill processing).
XCASE	0000004	Canonical upper/lower presentation.
ECHO	0000010	Enable echo.
ECHOE	0000020	Echo erase character as BS-SP-BS.
ECHOK	0000040	Echo NL after kill character.
ECHONL	0000100	Echo NL.
NOFLSH	0000200	Disable flush after interrupt or quit.

If ISIG is set, each input character is checked against the special control characters INTR, SWTCH, and QUIT. If an input character matches one of these control characters, the function associated with that character is performed. If ISIG is not set, no checking is done. Thus these special input functions are possible only if ISIG is set. These functions may be disabled individually by changing the value of the control character to an unlikely or impossible value (e.g., 0377).

If ICANON is set, canonical processing is enabled. This enables the erase and kill edit functions, and the assembly of input characters into lines delimited by NL, EOF, and EOL. If ICANON is not set, read requests are satisfied directly from the input queue. A read will not be satisfied until at least MIN characters have been received or the timeout value TIME has expired between characters. This allows fast bursts of input to be read efficiently while still allowing single character input. The MIN and TIME values are stored in the position for the EOF and EOL characters, respectively. The time value represents tenths of seconds.

If XCASE is set, and if ICANON is set, an upper-case letter is accepted on input by preceding it with a \ character, and is output preceded by a \ character. In this mode, the following escape sequences are generated on output and accepted on input:

for:	*use:*
`	\\'
|	\\!
	\\^
{	\\(
}	\\)
\\	\\\\

For example, **A** is input as **\a**, **\n** as **\\n**, and **\N** as **\\\n**.

If ECHO is set, characters are echoed as received.

When ICANON is set, the following echo functions are possible. If ECHO and ECHOE are set, the erase character is echoed as ASCII BS SP BS, which will clear the last character from a CRT screen. If ECHOE is set and ECHO is not set, the erase character is echoed as ASCII SP BS. If ECHOK is set, the NL character will be echoed after the kill character to emphasize that the line will be deleted. Note that an escape character preceding the erase or kill character removes any special function. If ECHONL is set, the NL character will be echoed even if ECHO is not set. This is useful for terminals set to local echo (so-called half duplex). Unless escaped, the EOF character is not echoed. Because EOT is the default EOF character, this prevents terminals that respond to EOT from hanging up.

If NOFLSH is set, the normal flush of the input and output queues associated with the quit, switch, and interrupt characters will not be done.

The initial line-discipline control value is all bits clear.

The primary *ioctl*(2) system calls have the form:

```
ioctl (fildes, command, arg)
struct termio *arg;
```

The commands using this form are:

TCGETA	Get the parameters associated with the terminal and store in the *termio* structure referenced by **arg**.
TCSETA	Set the parameters associated with the terminal from the structure referenced by **arg**. The change is immediate.
TCSETAW	Wait for the output to drain before setting the new parameters. This form should be used when changing parameters that will affect output.
TCSETAF	Wait for the output to drain, then flush the input queue and set the new parameters.

Additional *ioctl*(2) calls have the form:

```
ioctl (fildes, command, arg)
int arg;
```

The commands using this form are:

TCSBRK	Wait for the output to drain. If *arg* is 0, then send a break (zero bits for 0.25 seconds).
TCXONC	Start/stop control. If *arg* is 0, suspend output; if 1, restart suspended output.
TCFLSH	If *arg* is 0, flush the input queue; if 1, flush the output queue; if 2, flush both the input and output queues.

FILES

/dev/tty*

SEE ALSO

stty(1) in the *User's Reference Manual.*
fork(2), ioctl(2), setpgrp(2), signal(2) in the *Programmer's Reference Manual.*

NAME

timod — Transport Interface cooperating STREAMS module

DESCRIPTION

Timod is a STREAMS module for use with the Transport Interface (TI) functions of the Network Services library. The *timod* module converts a set of *ioctl*(2) calls into STREAMS messages that may be consumed by a transport protocol provider which supports the Transport Interface. This allows a user to initiate certain TI functions as atomic operations.

The *timod* module must be pushed (see *Streams Primer*) onto only a *stream* terminated by a transport protocol provider which supports the TI.

All STREAMS messages, with the exception of the message types generated from the *ioctl* commands described below, will be transparently passed to the neighboring STREAMS module or driver. The messages generated from the following *ioctl* commands are recognized and processed by the *timod* module. The format of the *ioctl* call is:

```
#include <sys/stropts.h>
        -
        -
struct strioctl strioctl;
        -
        -
strioctl.ic_cmd = cmd;
strioctl.ic_timeout = INFTIM;
strioctl.ic_len = size;
strioctl.ic_dp = (char *)buf

ioctl(fildes, I_STR, &strioctl);
```

Where, on issuance, *size* is the size of the appropriate TI message to be sent to the transport provider and on return *size* is the size of the appropriate TI message from the transport provider in response to the issued TI message. *Buf* is a pointer to a buffer large enough to hold the contents of the appropriate TI messages. The TI message types are defined in *<sys/tihdr.h>*. The possible values for the *cmd* field are:

TI_BIND	Bind an address to the underlying transport protocol provider. The message issued to the TI_BIND *ioctl* is equivalent to the TI message type T_BIND_REQ and the message returned by the successful completion of the *ioctl* is equivalent to the TI message type T_BIND_ACK.
TI_UNBIND	Unbind an address from the underlying transport protocol provider. The message issued to the TI_UNBIND *ioctl* is equivalent to the TI message type T_UNBIND_REQ and the message returned by the successful completion of the *ioctl* is equivalent to the TI message type T_OK_ACK.
TI_GETINFO	Get the TI protocol specific information from the transport protocol provider. The message issued to the TI_GETINFO *ioctl* is equivalent to the TI message type T_INFO_REQ and the message returned by the successful completion of the *ioctl* is equivalent to the TI message type T_INFO_ACK.
TI_OPTMGMT	Get, set or negotiate protocol specific options with the transport protocol provider. The message issued to the TI_OPTMGMT *ioctl* is equivalent to the TI message type T_OPTMGMT_REQ and the message returned by the successful completion of the *ioctl* is equivalent to the TI message type T_OPTMGMT_ACK.

FILES

<sys/timod.h>
<sys/tiuser.h>

<sys/tihdr.h>
<sys/errno.h>

SEE ALSO

tirdwr(7).
STREAMS Primer.
STREAMS Programmer's Guide.
Network Programmer's Guide.

DIAGNOSTICS

If the *ioctl* system call returns with a value greater than 0, the lower 8 bits of the return value will be one of the TI error codes as defined in *<sys/tiuser.h>*. If the TI error is of type TSYSERR, then the next 8 bits of the return value will contain an error as defined in *<sys/errno.h>* (see *intro*(2)).

NAME

tirdwr — Transport Interface read/write interface STREAMS module

DESCRIPTION

Tirdwr is a STREAMS module that provides an alternate interface to a transport provider which supports the Transport Interface (TI) functions of the Network Services library (see Section 3N). This alternate interface allows a user to communicate with the transport protocol provider using the *read*(2) and *write*(2) system calls. The *putmsg*(2) and *getmsg*(2) system calls may also be used. However, *putmsg* and *getmsg* can only transfer data messages between user and *stream*.

The *tirdwr* module must only be pushed [see I_PUSH in *streamio*(7)] onto a *stream* terminated by a transport protocol provider which supports the TI. After the *tirdwr* module has been pushed onto a *stream*, none of the Transport Interface functions can be used. Subsequent calls to TI functions will cause an error on the *stream*. Once the error is detected, subsequent system calls on the *stream* will return an error with *errno* set to EPROTO.

The following are the actions taken by the *tirdwr* module when pushed on the *stream*, popped [see I_POP in *streamio*(7)] off the *stream*, or when data passes through it.

push — When the module is pushed onto a *stream*, it will check any existing data destined for the user to ensure that only regular data messages are present. It will ignore any messages on the *stream* that relate to process management, such as messages that generate signals to the user processes associated with the *stream*. If any other messages are present, the I_PUSH will return an error with *errno* set to EPROTO.

write — The module will take the following actions on data that originated from a *write* system call:

- All messages with the exception of messages that contain control portions (see the *putmsg* and *getmsg* system calls) will be transparently passed onto the module's downstream neighbor.
- Any zero length data messages will be freed by the module and they will not be passed onto the module's downstream neighbor.
- Any messages with control portions will generate an error, and any further system calls associated with the *stream* will fail with *errno* set to EPROTO.

read — The module will take the following actions on data that originated from the transport protocol provider:

- All messages with the exception of those that contain control portions (see the *putmsg* and *getmsg* system calls) will be transparently passed onto the module's upstream neighbor.
- The action taken on messages with control portions will be as follows:
  + Messages that represent expedited data will generate an error. All further system calls associated with the *stream* will fail with *errno* set to EPROTO.
  + Any data messages with control portions will have the control portions removed from the message prior to passing the message on to the upstream neighbor.
  + Messages that represent an orderly release indication from the transport provider will generate a zero length data message, indicating the end of file, which will be sent to the reader of the *stream*. The orderly release message itself will be freed by the module.
  + Messages that represent an abortive disconnect indication from the transport provider will cause all further *write* and *putmsg* system calls to fail

with *errno* set to ENXIO. All further *read* and *getmsg* system calls will return zero length data (indicating end of file) once all previous data has been read.

+ With the exception of the above rules, all other messages with control portions will generate an error and all further system calls associated with the *stream* will fail with *errno* set to EPROTO.

— Any zero length data messages will be freed by the module and they will not be passed onto the module's upstream neighbor.

pop — When popping the module off or closing the *stream*, the module will take the following actions:

— If an orderly release indication has been previously received, then an orderly release request will sent to the remote side of the transport connection.

— If an abortive disconnect has been previously received, then no action is taken.

— If neither an abortive disconnect nor an orderly release have been previously received, an abortive disconnect will be initiated by the module.

— If an error has occurred previously and an abortive disconnect has not been previously received, an abortive disconnect will be initiated by the module.

SEE ALSO

streamio(7), timod(7).
intro(2), getmsg(2), putmsg(2), read(2), write(2), intro(3) in the *Programmer's Reference Manual.*
STREAMS Primer.
STREAMS Programmer's Guide.
Network Programmer's Guide.

NAME

tty — controlling terminal interface

DESCRIPTION

The file */dev/tty* is, in each process, a synonym for the control terminal associated with the process group of that process, if any. It is useful for programs or shell sequences 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 the name of a file for output, when typed output is desired and it is tiresome to find out what terminal is currently in use.

FILES

/dev/tty
/dev/tty*

SEE ALSO

console(7)

NAME

intro — introduction to system maintenance procedures

DESCRIPTION

This section outlines certain procedures that will be of interest to those charged with the task of system maintenance.

SEE ALSO

Administrator's Guide.

NAME

mk — remake the binary system and commands from source code

DESCRIPTION

All source code for the GENIX V.3 system is distributed in the directory **/usr/src.** The directory tree rooted at **/usr/src** includes source code for the operating system, libraries, commands, miscellaneous data files necessary for the system and procedures to transform this source code into an executable system.

Within the **/usr/src** directory are the **cmd, lib, uts, head,** and **stand** directories, as well as commands to remake the parts of the system found under each of these sub-directories. These commands are named *:mk* and *:mk***dir** where **dir** is the name of the directory to be recreated. Each of these *:mk***dir** commands will rebuild all or part of the directory it is responsible for. The *:mk* command will run each of the other commands in order and thus, recreate the whole system. The *:mk* command is distributed only to source code licensees.

Each command, with its associated directory, is described below.

:mklib The **lib** directory contains the source code for the system libraries. The most important of these is the C library. Each library is in its own sub-directory. If any arguments are specified on the *:mklib* command line then only the given libraries will be rebuilt. The argument * will cause it to rebuild all libraries found under the **lib** directory.

:mkhead
The **head** directory contains the source code versions of the header files found in the **/usr/include** directory. The *:mkhead* command will install the header files given as arguments. The argument * will cause it to install all header files.

:mkuts The **uts** directory contains the source code for the GENIX V.3 Operating System. The *:mkuts* command takes no arguments and invokes a series of makefiles that will recreate the operating system.

Associated with the operating system is a set of header files that describe the user interface to the operating system. The source for these header files is found in a sub-directory within the **uts** directory tree. The user-accessible versions of these header files are found in the **/usr/include/sys** directory. The *:mksyshead* command will install these header files into the **/usr/include/sys** directory.

:mkstand
The **stand** directory contains stand-alone commands and boot programs. The *:mkstand* command will rebuild and install these programs. Note that these stand-alone programs are only applicable to the DEC processors and will not be built for any other machine.

:mkcmd
The **cmd** directory contains the source code for all the commands available on the system. There are two types of entries within the **cmd** directory: commands whose source code consists of only one file with one of the following suffixes: **.l, .y, .c, .s, .sh,** or a sub-directory that contains the multiple source files that comprise a particular command or subsystem. Each sub-directory is assumed to have a makefile (see *make*(1)) with the name *command***.mk** that will take care of creating everything associated with that directory and its sub-directories.

The *:mkcmd* command transforms source code into an executable command based upon a set of predefined rules. If the *:mkcmd* command encounters a sub-directory within the **cmd** directory then it will run the makefile found in that sub-directory. If no makefile is found then an error will be reported. For single file commands, the predefined rules are dependent on the file's suffix. C programs (**.c**) are compiled by the C compiler and loaded stripped with shared text. Assembly language programs (**.s**) are assembled and loaded stripped. Yacc programs (**.y**) and lex programs (**.l**) are processed by *yacc*(1) and *lex*(1) respectively, before C compilation. Shell programs (**.sh**) are copied to create the command. Each of these operations leaves a command in the **./cmd** directory which is then installed into a user-accessible directory by using **/etc/install.**

The arguments to *:mkcmd* are either command names or subsystem names. The subsystems distributed with the GENIX V.3 system are: **acct, graf, sgs, sccs,** and **text.** Prefacing the *:mkcmd* command with an assignment to the shell variable **$ARGS** will cause the indicated components of the subsystem to be rebuilt.

For example, the entire **sccs** subsystem can be rebuilt by:

```
/usr/src/:mkcmd sccs
```

while the *delta* component of **sccs** can be rebuilt by:

```
ARGS="delta" /usr/src/:mkcmd sccs
```

The *log* command, which is a part of the **stat** package, which is itself a part of the **graf** package, can be rebuilt by:

```
ARGS="stat log" /usr/src/:mkcmd graf
```

The argument * will cause all commands and subsystems to be rebuilt.

Makefiles throughout the system, and particularly in the **cmd** directory, have a standard format. In particular, *:mkcmd* depends on each makefile having target entries for *install* and *clobber*. The *install* target should cause everything over which the makefile has jurisdiction to be built and installed by **/etc/install.** The *clobber* target should cause a complete cleanup of all unnecessary files resulting from the previous invocation.

An effort has been made to separate the creation of a command from source and its installation on the running system. The command **/etc/install** is used by *:mkcmd* and most makefiles to install commands in standard directories on the system. The use of *install* allows maximum flexibility in the administration of the system. The *install* command makes very few assumptions about where a command is located, who owns it, and what modes are in effect. All assumptions may be overridden on invocation of the command, or more permanently by redefining a few variables in *install.* The purpose of *install* is to install a new version of a command in the same place, with the same attributes as the prior version.

In addition, the use of a separate command to perform installation allows for the creation of test systems in other than standard places, easy movement of commands to balance load, and independent maintenance of makefiles.

SEE ALSO

install(1M).
lex(1), make(1), yacc(1) in the *Programmer's Reference Manual.*

NAME

sysdump — dump system memory image to floppy disk(s) or tape

SYNOPSIS

sysdump [—f]

DESCRIPTION

The *sysdump* command dumps the system memory image to either a tape or to one or more floppy disks depending on the size of memory and user request. This memory image can later be analyzed by *crash*(1M). *Sysdump* can be invoked only by the superuser.

Sysdump allows the user to back out by answering **No** to the first question. After that, there is no way back. The —f option (force) suppresses this first question and starts the dump process immediately.

When the tape is selected, *sysdump* prompts the user to insert a tape into the tape drive and then dumps all of system memory image to that tape.

When the floppy is selected, *sysdump* begins an interactive procedure that prompts the user to insert the floppies to be loaded. The user has the option of quitting the session any time. This allows only the portion of the system image needed to be dumped.

The output of *sysdump* provides only one of two inputs to *crash*(1). The other input is the text file that was used to boot this system image. This is needed to provide symbolic reference to the system dump. The text file must be manually saved after the machine has been booted. If **/unix** was booted then this should be dumped to floppy to accompany the system dump.

FILES

/dev/mt/0m — device used for tape access
/dev/diskette — device used for floppy access
/unix — the text file typically used to boot the machine

SEE ALSO

crash(1M), ldsysdump(1M).

DIAGNOSTICS

If a floppy diskette cannot be written to, a message is printed. The user is allowed to insert a new diskette and continue the session. The same is true for a tape.

WARNINGS

It is critical to provide access to the text file that was used to boot the machine. This file must be saved.

The diskettes should be labeled clearly so they can be loaded in the proper sequence.

National
Semiconductor

MICROCOMPUTER
SYSTEMS DIVISION

READER'S COMMENT FORM

In the interest of improving our documentation, National Semiconductor invites your comments on this manual.

Please restrict your comments to the documentation. Technical Support may be contacted at:

(800) 538-1866 - U.S. non CA
(800) 672-1811 - CA only
(800) 223-3248 - Canada only

Please rate this document according to the following categories. Include your comments below.

	EXCELLENT	GOOD	ADEQUATE	FAIR	POOR
Readability (style)	☐	☐	☐	☐	☐
Technical Accuracy	☐	☐	☐	☐	☐
Fulfills Needs	☐	☐	☐	☐	☐
Organization	☐	☐	☐	☐	☐
Presentation (format)	☐	☐	☐	☐	☐
Depth of Coverage	☐	☐	☐	☐	☐
Overall Quality	☐	☐	☐	☐	☐

NAME ______________________ DATE __________

TITLE ______________________

COMPANY NAME/DEPARTMENT ______________________

ADDRESS ______________________

CITY ______________ STATE __________ ZIP ________

Do you require a response? ☐ Yes ☐ No PHONE ______________

Comments:

____

____

____

____

____

____

FOLD, STAPLE, AND MAIL

424510771-210A

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 409 SANTA CLARA, CA

POSTAGE WILL BE PAID BY ADDRESSEE

National Semiconductor Corporation
Microcomputer Systems Division
Technical Publications Dept. 8278, M/S 7C261
2900 Semiconductor Drive
P.O. Box 58090
Santa Clara, CA 95052-9968