ConvexOS Man Pages for System Managers

Second Edition

CONVEX COMPUTER CORPORATION

Bryliski

CONVEX Computer Corporation
3000 Waterview Parkway
P.O. Box 833851
Richardson, TX 75083-3851
United States of America
(214)497-4000

ConvexOS Man Pages for System Managers

Order No. DSW-333

Second Edition
December 1991

CONVEX Press
Richardson, Texas USA

ConvexOS Man Pages for System Managers

Order No. DSW-333

Printed in the United States of America

Revision information for

ConvexOS Man Pages for System Managers

Edition	Document No.	Description
Second	710-015930-001	Released with ConvexOS V10.0, December 1991.
First	710-015930-000	Initial release, April 1991.

Contents

Index

Using ConvexOS man pages

What are man pages?

The ConvexOS man pages, online and hardcopy, are a reference for all of the commands available to ConvexOS users. Each command has its own "man page," which can be one or more pages in length.

The ConvexOS man pages are organized into 7 sections; commands for similar functions are grouped together. This book contains one of those sections, Section 8, which describes tools used primarily by system managers. (Traditionally, Section 6 of the man pages is allotted to games. Because CONVEX does not sell games, the ConvexOS man pages do not contain Section 6.)

Where to find other ConvexOS man pages

ConvexOS Man Pages for Users contains Section 1 and Section 7 of the ConvexOS man pages. These sections describe:

- Section 1 man pages—Commands of general utility and commands for communication with other systems.
- Section 7 man pages—Miscellaneous commands, primarily in the areas of text processing and terminal environments.

ConvexOS Man Pages for Programmers contains Section 2 through Section 5 of the ConvexOS man pages. These sections describe:

- Section 2 man pages—System calls and error numbers
- Section 3 man pages—Various library functions
- Section 4 man pages—Special files, related driver functions, and networking support
- Section 5 man pages—The format of various files

How to use ConvexOS man pages

The ConvexOS man pages are available in two formats: the hardcopy man pages contained in this book and the online man pages you can access by using the ConvexOS man command.

Using hard copy man pages

The ConvexOS man pages contained in this book are ordered alphabetically.

Using online man pages

man is a program that formats and displays information from the hard copy man pages. When invoked in the simplest way, without any options and without a topic, it displays the corresponding manual page formatted with nroff. Access online man pages by entering

```
% man entry-name
```

where *entryname* is the name of the man page to be displayed. You can print hard copies of online man pages by entering

```
% man -t
```

For more information on man options, refer to the man(1) man page.

Format of man pages

The contents of each man page is divided into subsections (not all of which are relevant to each entry):

NAME	Lists exact name of command or subroutine and describes its purpose
SYNOPSIS	Summarizes use of command or subroutine being described
DESCRIPTION	Discusses use of command or subroutine
FILES	Names files built into the program
SEE ALSO/REFERENCES	Points to related information
DIAGNOSTICS/ERRORS	Discusses diagnostic messages the system produces
BUGS/NOTES	Explains known bugs or deficiencies and any known solutions

When you can't find the man page you need

One man page can document several commands. For instance, if you enter

```
% man moncontrol
```

the moncontrol(3) man page appears on your screen. However, there is no separate hard copy man page for moncontrol. If you look it up in the index, you are referred to the monitor(3) man page, which describes both the moncontrol and monitor commands, among others.

If you cannot find a hard copy man page for a specific command, look up that command in the index—it will refer you to the man page you need.

Technical assistance

If you have questions that are not answered in this book, contact the CONVEX Technical Assistance Center (TAC).

- Within the continental U.S., call 1(800)952-0379.
- From Canada, call 1(800)345-2384.
- All other locations, contact the local CONVEX office.

Section 8
System management

NAME

intro – introduction to system maintenance and operation commands

DESCRIPTION

This section contains information related to system operation and maintenance. In particular, commands used to create new file systems (*newfs*, *newst*, *mkfs*) and to verify the integrity of the file systems (*fsck*, *preen*) are described here.

LIST OF PROGRAMS

Program	*Appears on Page*	*Description*
ac	ac(8)	login accounting
accton	sa(8)	system accounting
accounting	accounting(8)	system accounting
actwhocheck	edactwho(8)	edit/check */etc/actwho* file
adbccu	adbccu(8)	debugger
arp	arp(8C)	address resolution display and control
atrun	at(1)	execute commands at a later time
biod	nfsd(8)	NFS daemons
catman	catman(8)	create the cat files for the manual
chall	chall(8)	change mode, owner and/or group of a file
chown	chown(8)	change owner
clri	clri(8)	clear i-node
comsat	comsat(8C)	biff server
connecttime	connecttime(8)	compute connect time
cron	cron(8)	clock daemon
ctar	ctar(8)	cartridge tape archiver
dcheck	dcheck(8)	file system directory consistency check
diskuse	diskuse(8)	compute user and group diskspace totals
dump	dump(8)	incremental file system dump
dumpfs	dumpfs(8)	dump file system information
edactwho	edactwho(8)	edit/check */etc/actwho* file
edquota	edquota(8)	edit file system quotas
fastboot	fastboot(8)	reboot/halt the system without checking disks
fasthalt	fastboot(8)	reboot/halt the system without checking disks
fsck	fsck(8)	file system consistency check and interactive repair
fsirand	fsirand(8)	randomize inode generation numbers
ftpd	ftpd(8C)	DARPA Internet File Transfer Protocol server
getNAME	makewhatis(8)	rebuild the whatis database
getst	getst(8)	put the stripe table in the kernel
getty	getty(8)	set terminal mode
halt	halt(8)	stop the processor
icheck	icheck(8)	file system storage consistency check
ifconfig	ifconfig(8C)	configure network interface parameters
inetd	inetd(8C)	internet services daemon
init	init(8)	process control initialization
installsw	installsw(8)	create/install software release tapes
lockd	lockd(8C)	network lock manager
lpc	lpc(8)	line printer control program
lpd	lpd(8)	line printer daemon
lpf	lpf(8)	general line printer filter
lprewind	lprewind(8)	line printer rewind
makedbm	makedbm(8)	make a yellow pages dbm file
makedev	makedev(8)	make system special files
makekey	makekey(8)	generate encryption key

makewhatis	makewhatis(8)	rebuild the whatis database
mkfs	mkfs(8)	construct a file system
mklost+found	mklost+found(8)	make a lost+found directory for fsck
mknf	mknf(8)	create and delete notesfiles
mknod	mknod(8)	build special file
mount	mount(8)	mount and dismount file system
mountd	mountd(8C)	NFS mount request server
ncheck	ncheck(8)	generate name from i-numbers
newfs	newfs(8)	construct a new file system
newst	newst(8)	construct a new striped file system
nfaccess	nfaccess(8)	add access rights to a set of notesfiles
nfarchive	nfarchive(8)	archive notesfiles
nfdump	nfdump(8)	notesfile dump/load program
nfload	nfdump(8)	notesfile dump/load program
nfmail	nfmail(8)	accept mail for a notesfile
nfrcv	nfxmit(8)	notesfile networking programs
nfsd	nfsd(8)	NFS daemons
nfsstat	nfsstat(8)	Network File System statistics
nfxmit	nfxmit(8)	notesfile networking programs
nu	nu(8)	add new user to the system
off	on(8)	enable/disable logins on a tty
pac	pac(8)	printer/plotter accounting information
ping	ping(8C)	send ICMP ECHO_REQUEST packets to a network host
pong	pong(8C)	network debugging
portmap	portmap(8C)	DARPA port to RPC program number mapper
preen	preen(8)	run fsck in parallel over normally mounted disks
pstat	pstat(8)	print system facts
putst	putst(8)	manipulate kernel databases for striped disk partitions
quot	quot(8)	summarize file system ownership
quotacheck	quotacheck(8)	file system quota consistency checker
quotaoff	quotaon(8)	enable/disable file system quotas
quotaon	quotaon(8)	enable/disable file system quotas
rc	rc(8)	command script for auto-reboot and daemons
rdump	rdump(8C)	dump file systems across the network
reboot	reboot(8)	ConvexOS bootstrapping procedures
renice	renice(8)	alter priority of running processes
repquota	repquota(8)	summarize quotas for a file system
restore	restore(8)	incremental file system restore
rexd	rexd(8C)	RPC-based remote execution server
rexecd	rexecd(8C)	remote execution server
rlogind	rlogind(8C)	remote login server
rmnf	mknf(8)	create and delete notesfiles
rmt	rmt(8C)	remote magtape protocol server
route	route(8C)	manually manipulate the routing tables
routed	routed(8C)	network routing daemon
rpcinfo	rpcinfo(8)	remote RPC information
rquotad	rquotad(8C)	remote quota server
rrestore	rrestore(8C)	restore a file system dump across the network
rshd	rshd(8C)	remote shell server
rstatd	rstatd(8C)	kernel statistics server
rusersd	rusersd(8C)	network rusers service
rwalld	rwalld(8C)	network rwall server
rwhod	rwhod(8C)	system status server

sa	sa(8)	system accounting
seestat	seestat(8)	print/collect graphs of system status
sendmail	sendmail(8)	send mail over the internet
showmount	showmount(8)	show al remote mounts
shutdown	shutdown(8)	close down the system at a given time
spray	spray(8)	spray packets
sprayd	sprayd(8C)	spray server
spu	spu(8)	read or write SPU OS file system
stat	seestat(8)	print/collect graphs of system status
statd	statd(8C)	network status monitor
sticky	sticky(8)	file mode sticky bit
sumscripts	sumscripts(8)	accounting summary awk scripts
swapon	swapon(8)	specify additional device for paging and swapping
sync	sync(8)	update the super block
sysex	sysex(8)	system exerciser that simulates stressful load conditions
sysgen	sysgen(8)	build customized operating system images
syslog	syslog(8)	log systems messages
syspic	syspic(8)	report various system activity in windows
talkd	talkd(8)	talk daemon
telnetd	telnetd(8C)	DARPA TELNET protocol server
tpconfig	tpconfig(8)	configure the tape system
tpd	tpd(8)	tape allocation/deallocation monitoring program
trpt	trpt(8C)	transliterate protocol trace
umount	mount(8)	mount and dismount file system
update	update(8)	periodically update the super block
uucico	uucico(8C)	UUCP protocol handler
uuclean	uuclean(8C)	uucp spool directory clean-up
uulook	uulook(8)	monitor inter-system communications
uupoll	uupoll(8C)	poll a remote system via UUCP
uusnap	uusnap(8C)	show snapshot of the UUCP system
uuxqt	uuxqt(8C)	UUCP command execution
vipw	vipw(8)	edit the password file
ypbind	ypserv(8)	yellow pages server and binder processes
ypinit	ypinit(8)	build and install yellow pages database
ypmake	ypmake(8)	rebuild yellow pages database
yppasswdd	yppasswdd(8C)	server for modifying YP passwd file
yppoll	yppoll(8)	what version of a YP map is at a YP server host
yppush	yppush(8)	force propagation of a changed YP map
ypserv	ypserv(8)	yellow pages server and binder processes
ypset	ypset(8)	point ypbind at a particular server
ypxfr	ypxfr(8)	transfer a YP map from some YP server to here

NAME

ac – login accounting

SYNOPSIS

/etc/ac [**–w** wtmp] [**–p**] [**–d**] [people] ...

DESCRIPTION

Ac produces a printout giving connect time for each user who has logged in during the life of the current *wtmp* file. A total is also produced. **–w** is used to specify an alternate *wtmp* file. **–p** prints individual totals; without this option, only totals are printed. **–d** causes a printout for each midnight to midnight period. Any *people* will limit the printout to only the specified login names. If no *wtmp* file is given, */usr/adm/wtmp* is used.

The accounting file */usr/adm/wtmp* is maintained by *init* and *login*. Neither of these programs creates the file, so if it does not exist no connect-time accounting is done. To start accounting, it should be created with length 0. On the other hand if the file is left undisturbed it will grow without bound, so periodically any information desired should be collected and the file truncated.

FILES

/usr/adm/wtmp

SEE ALSO

init(8), sa(8), login(1), utmp(5).

NAME

accounting – periodic accounting utilities

SYNOPSIS

30 * * * * /usr/adm/accounting > /dev/console
37 00 * * * su root < /usr/lib/diskspace
45 4 * * 5 su root < /usr/lib/diskmail
/usr/adm/accounting -i
/usr/adm/accounting.c
/usr/adm/acctsum.awk

DESCRIPTION

The periodic accounting utilities reside mostly in the */usr/adm* directory and provide simple accounting of disk space and CPU usage for both individuals and complete projects.

Cron(1) runs the */usr/adm/accounting* program every hour that the machine is up. If */usr/adm/accounting* finds that the day (or week or month) has changed since the last time it "ran accounting", then */usr/adm/accounting* invokes the appropriate *daily*, *weekly*, and/or *monthly* script(s). These scripts use *sa*(8) to manipulate a number of incremental and cumulative files.

The command-line syntax that */usr/adm/accounting* uses to invoke the scripts is "*script num old now*". The *old* and *now* parameters are timestamps as returned by *time*(3). *Old* and *now* indicate when */usr/adm/accounting* last ran the script and what the current time is, respectively. If this is the first time the script has run since accounting was initialized, then *old* is the time of initialization. The value of *num* depends on the script. For the *daily* script, *num* is the current day of the week; Sunday is 0, Monday is 1, etc. For *weekly*, *num* is the current date in the form *month.day.year*; for example, "6.20.83". For *monthly*, *num* is the current date in the form *month.year*; for example, "6.83".

The *daily* script copies the */usr/adm/acct* accounting information into a file named */usr/adm/dailyZ* where *Z* is *num* as described above. *Daily* then runs user and process accounting. The accounting accrues in one file on a daily basis, another on a weekly basis, and still another on a monthly basis. The daily user accounting summary is placed into the file */usr/adm/daily.u*. The scripts have the capability of automatically printing this file on the printer (in the middle of the night, usually). The script places the process accounting summary into the */usr/adm/daily.p* file. This file, too, can be printed in the middle of the night if desired.

The weekly and monthly scripts place their output in */usr/adm/week.u*, */usr/adm/week.p*, */usr/adm/month.u*, and */usr/adm/month.p*. These scripts normally print summaries on some printer. The *weekly* script copies cumulative weekly accounting from */usr/adm/usracctw* and */usr/adm/savacctw* into files */usr/adm/u.w.Z* and */usr/adm/s.w.Z* respectively, where *Z* is *num* as described above. Similarly, the *monthly* script copies each month's accounting into files */usr/adm/u.m.Z* and */usr/adm/s.m.Z*.

The scripts take advantage of the **-F** switch of *sa*(8) which allows the use of alternate accounting files with names like */usr/adm/savacctZ* for accumulation of intermediate totals.

An awk script called */usr/adm/acctsum.awk* defines the various groups into which users' accounting is summed. Change it by hand unless an automatic regeneration program can deduce groups from some other scheme.

The **-i** switch initializes the times that */usr/adm/accounting* uses in determining when to run daily, weekly, and monthly accounting. These times are kept in file */usr/adm/lastacct*. Use of the **-i** switch will cause daily accounting to run at the beginning of the next day, weekly accounting to run at the beginning of the next week, and monthly accounting to run at the beginning of the next month. This should only be done at installation time or when making a change to the local accounting setup.

A copy of the source for */usr/adm/accounting* is in */usr/adm/accounting.c*. The source may be copied and modified to produce a version of */usr/adm/accounting* which meets local accounting requirements. The *daily*, *weekly*, and *monthly* scripts may also be copied and modified.

The disk accounting runs each day as specified by */usr/lib/crontab* and produces unsorted (*/usr/adm/disk.today*), sorted (*/usr/adm/disk.today.1*), and grouped (*/usr/adm/disk.today.2*) lists of disk usage. The groupings rely on the */usr/adm/disksum.awk* script which has the same caveats as the */usr/adm/acctsum.awk* script mentioned above. The */usr/lib/diskspace* scripts contain references to the various mounted file systems and hence must be updated when new file systems are created.

Finally, the */usr/lib/diskmail* script sends users summaries of the recent disk usage as recalled by the */usr/adm/diskuse/** files (one for each user). The script relies upon having an up-to-date list of users and their various directories as part of its source.

FILES

/usr/adm/daily	daily accounting script
/usr/adm/weekly	weekly accounting script
/usr/adm/monthly	monthly accounting script
/usr/adm/lastacct	timestamps controlling when accounting is run
/usr/adm/accounting	program that executes daily, weekly, monthly scripts
/usr/adm/accounting.c	source for */usr/adm/accounting*
/usr/adm/acctsum.awk	awk script for summarizing sa output
/usr/lib/crontab	contains an entry for */usr/adm/accounting*

SEE ALSO

cron(1), sa(8),
"Accounting" chapter in the *Managing ConvexOS* doumentation set.

BUGS

After accounting is initialized, weekly accounting will run on the first Sunday the machine is up, but if the machine happens to be down all day on this day, accounting will not run until one week after accounting was initialized.

NAME

adbccu – CCU device driver debugger

SYNOPSIS

adbccu [**–nvwEC**] [**–t***isa*] [**–I***dir*] [**–f***file*] [**–T***sec*] [**–r***base*] [**-s***pages*] [*objfil* [*corfil*]]

DESCRIPTION

adbccu is a special-purpose channel control unit (CCU) device driver debugging program. *adbccu* can be used to control the execution of the CCU side of a device driver or to examine CCU memory image (crashdump) files and SPU OS core files. When *adbccu* is started on a running CCU, the CCU is normally suspended. The registers and stack can thus be inspected and breakpoints set.

There are two instances of the program. One version runs on the SPU and is shipped with the base OS. The second version runs on the CPU and is part of the 68KTOOLS optional product. The SPU version is primarily used to develop and patch driver programs and the CPU version finds its primary use as a crashdump analysis tool. (See *dumpccu(8)* man page.) Both versions send debug messages via the Message Based System (MBS) to the target CCU, when *corfil* is a CCU.

objfil is normally an executable CCU program file, preferably containing a symbol table; if not, the symbolic features of *adbccu* cannot be used although the file can still be examined. *adbccu* supports two types of CCU executables, those for the MC68K-based CCUs (iop, hsp, and viop), and those for the MC88K-based CCUs (idc, tli, hippi, and itc). The selection of target instruction set architecture (isa) is automatic, based upon the header of *objfil*. It may also be specified with the **-t** command-line option. This option is required for examining SPU OS core files. The default name for *objfil* is "viop". *corfil* is assumed to be memory image file produced by *dumpccu(8)*, *cdump(8)* or a SPU OS core file (from Convex SPU, SP2, SP4 or SP5 service processor only.) The default name for *corfil* is "viop.core0".

The following options are interpreted by *adbccu*:

–n Nostop. (valid only when *corfil* is */dev/ccuN.)* This option causes the debugger to restart the CCU immediately, resulting in only a momentary pause (less than 1 sec.) in CCU operation. Normally, the CCU is suspended when the */dev/ccu* special file is opened because the open causes the driver to send a STOP message to the CCU processor executive. The executive then suspends normal driver execution (not really halting the CCU) and services debugger requests. Using *adbccu* with the **–n** option provides a convenient way to monitor CCU variables on a running CCU by permitting one to enter debugger peek commands without stopping the CCU. See also the description of SIGQUIT and SIGINT handling, below.

–v Print the program version number, and exit. Intended for use by installation scripts.

–w Enable write mode. Both *objfil* and *corfil* are created (if necessary), then opened for reading and writing such that they can be modified using *adbccu*. Write mode is always enabled for */dev/ccuN.*

–E Use the EPROM message interface. This option is only valid on *dev/ccuN* core files, and supports only peek and poke debugger commands. This can be useful to development, manufacturing engineers and technicians as an interactive alternative to taking a full crashdump. It depends upon the EPROM having already been placed in loader-mode, which is done by sending it an **0xFE** interrupt. The normal way to do this is from the SPU console by typing "sendint fe". This interrupt is required following a CCU reset because the self test does not automatically jump to the message loader code but polls its interrupt status register, looking for the **0xFE** interrupt before jumping to the message handler module.

–C *corfil* is an X.25 or Vasync (68K) controller memory image file taken during

crashdump. The format of the X.25 and Vasync memory image files created by *cdown(8)* is different than those of the CCU corefiles created with *dumpccu(8)*. In particular, the registers are written in a header area, separate from the memory image. This option informs the debugger to retrieve registers from this area instead of the default area for MC68K executables. The -C option is not valid for */dev/ccuN* core files.

–t*isa* Specify the target instruction set architecture (isa) and default file maps. (See ADDRESSES and ADDRESS MAPS.) The three valid values for target are: *spu, 68k,* and *88k*. If *corfil* is a CCU memory image or */dev/ccu,* the debugger is able to use the object file header to select target instruction set and corefile map. However, the object header for SPU OS has the same magic number as for the MC68K CCUs, so one must use the "-tspu" option to examine SPU OS core files. This option of *adbccu* supercedes the funtionality previously available in the *adb68* utility shipped with the 68K tools product. One may make a hardlink to adbccu, e.g. : "ln adbccu adb68" and invoke the program by the link name to get the effect of having used the **–tspu** option.

–I*dir* Specify command file directory. *dir* is a directory where files to be read with the $< and $<< commands (see below) will be sought. Default search rule is to first look in the current directory ".", then to look in */usr/lib/adbccu.* This option specifies a search list to use **before** looking in the default directories. Multiple directories in the list must be separated by the colon, ":" character. Multiple -I switches may be used, in which case the last -I option will be the first list searched.

For example:

adbccu -I abc:def viop viop.core0

creates a search path of "(abc, def/ ./, /usr/lib/adbccu/)", and

adbccu -I abc -I def viop viop.core0

creates a search path of "(def/, abc/ ./, /usr/lib/adbccu/)".

Spaces between the **-I** and *dir* are optional.

–f*file* Immediately following setup, the debugger will open and execute debugger commands in the named file before reading commands from stdin. This provides a convenient method to setup the special address map segments that define CCU specific memory-mapped devices. One could also easily run patch scripts from another script this way. (See ADDRESS MAPS, below.)

–T*sec* This option overrides the default MBS timeout and must be specified in decimal seconds. *adbccu* uses messages to communicate with the CCU, and in order to prevent the program from hanging if the CCU dies without returning an expected message, there is a timer started whenever a message is to be received. If the debugger is waiting for the message, such as when waiting for a breakpoint to occur, and the timer expires, the debugger is interrupted and returns to the command level. The default timeout is 10 minutes (600 seconds). One may also change the timeout dynamically by storing to the built-in variable *timeout.* (See VARIABLES.)

–r*base* This option overrides the built-in knowledge of where the processor registers are saved in RAM when a breakpoint occurs. It is ONLY intended for use by development engineers, and not with standard released drivers and CCU executives, which always do what is expected by the debugger. The address should be entered in decimal.

–spages This option sets the number of 4096-byte pages of scratchpad memory to allocate. See VARIABLES for a description of scratchpad. The default size of the scratchpad is 1/4page (1K). The maximum allowed is 15 pages (**0xf000** bytes).

Requests to *adbccu* are read from the standard input and responses are to the standard output. Error messages are written to standard error. *adbccu* ignores the SIGQUIT signal when examining a crashdump corefile, but will send a stop message to the CCU when controlling a CCU corefile. The stop message effectively *forces* a special kind of breakpoint so that the CCU will be suspended and the debugger notified just as though a regular breakpoint had occurred. On all SPUs except the C3800 workstation SPU, a SIGQUIT signal is sent by typing a ^B on the SPU console. A SIGINT signal (^C) will cause a return to command processing level without stopping the CCU. This permits examining and altering driver state variables on a running CCU. Note that one can easily crash a running CCU if the wrong change is made to CCU memory.

In general, requests to *adbccu* are of the form:

[*address*] [, *count*] *command*] [;]

If *address* is present, then *dot* is set to *address*. Initially, *dot* is set to **0**. For most commands, *count* specifies how many times the command will be executed. The default *count* is **1**. *address* and *count* are expressions. The semicolon is a command separator and may be used to place more than one command on the same line. The interpretation of an address depends on the context in which it is used. If a running CCU is being debugged, then addresses are interpreted in the usual way in the address space of the CCU. For details of address interpretation in crashdump corefiles, see ADDRESSES and ADDRESS MAPS.

EXPRESSIONS

. The value of *dot*. (*dot* is the value of each command's *address* expression.)

\+ The value of *dot* incremented by the current increment.

^ The value of *dot* decremented by the current increment.

" The last *address* typed.

integer A number. The prefixes **0o** and **0O** ("zero oh") force interpretation in octal radix; the prefixes **0t** and **0T** force interpretation in decimal radix; the prefixes **0x** and **0X** force interpretation in hexadecimal radix. Thus 0o20 = 0t16 = 0x10 = 16. If no prefix appears, then the default radix is used; see the $*z* command. The default radix is initially hexadecimal. The hexadecimal digits are **0123456789** and **abcdef** or **ABCDEF** with the obvious values. Note that a hexadecimal number whose most significant digit is an alphabetic character must have a **0x** (or **0X**) prefix if the current input radix is *not* hexadecmial.

'cccc' The ASCII value of up to 4 characters. A backslash ("\") may be used to escape a single quote ("'"). Note that *adbccu* always uses *big-endian* storage so that the command **'abcd'=w** will print **0x61626364**.

< name The value of *name*, which is either a variable, register, or field name or field expression. *adbccu* maintains a number of variables named by single letters or digits. See the VARIABLES and FIELDS sections for further details. If *name* is a register name, the value of the register is obtained from the system header in *corfil*. The register names are those displayed by the $*r* command.

> name This expression returns the contents of the register or variable somewhat like *< name* but it is actually a command. (See COMMANDS, below.) Because the command *address>name* returns an expression whose value is *address* it can be used in statements similar to the C language operator '=.' For example, the command **4>a>b** stores the value **4** in both of the debugger variables **a** and **b** just as the C statement 'a = b = 4;' would do for the variables a and b.

<- This expression permits user input to be taken by a script. It causes *adbccu* to read one line of input from the console and evaluate *only one expression* according to the rules for expression evaluation.

<= This expression returns 0 if the last use of <- did not evaluate to a valid expression. Otherwise it returns non-zero (which is actually the size in bytes of the evaluated expression).

symbol A *symbol* is a sequence of upper-case or lower-case letters, underscores or digits, not starting with a digit. The backslash character ("\") can be used to escape other characters. The value of the *symbol* is taken from the symbol table in *objfil*. For convenience, an initial underscore ("_") is prepended to *symbol* if *symbol* doesn't match any symbol in the symbol table.

(*exp*) The value of the expression *exp*.

Monadic operators.

@ *exp* The contents of the location addressed by *exp* in *objfil*.

* *exp* The contents of the location addressed by *exp* in *corfil*.

' *exp* The contents of the location addressed by *exp* in *debugger space*

– *exp* Integer (two's complement) negation.

~ *exp* Bitwise complement.

exp Logical negation.

Dyadic operators are left associative and are less binding than monadic operators.

e1 + *e2* Integer addition.

e1 – *e2* Integer subtraction.

e1 * *e2* Integer multiplication.

e1 % *e2* Integer division.

e1 & *e2* Bitwise conjunction. (AND)

e1 | *e2* Bitwise disjunction. (OR)

e1 # *e2* *e1* rounded up to the next multiple of *e2*. A special case for *e2* equal to **0** is often useful: The expression returns **1** if *e1* is non-zero otherwise it returns **0**.

COMMANDS

Most commands consist of a verb followed by a modifier or list of modifiers. The following verbs are available.

?[*n*]*f* Locations starting at *address* in *objfil* are displayed according to the format *f*. *dot* is incremented by the sum of the increments for each format letter when *count* is specified. If *n* is given, format repetition is used in which case *dot* is not incremented. The following two examples display the same values but *dot* is modified only by the first example.

```
(adbccu) 0x1000,4?w;.=w
kernentry:      0x46FC2700      0x70084E7B      0x27001         0x4E7B0002
                0x100C

(adbccu) 0x1000?4w;.=w
kernentry:      0x46FC2700      0x70084E7B      0x27001         0x4E7B0002
                0x1000
```

/[*n*]*f* Locations starting at *address* in *corfil* are displayed according to the format *f* and *dot* is incremented as for "?".

=[*n*]*f* The value of *address* itself is displayed in the style indicated by the format *f*.

][*n*]*f* Locations starting at *address* in *debugger space* are displayed according to the format *f* and *dot* is incremented as for "?". *debugger space* is a portion of the debugger program internal memory that is used to implement message variables, and the scratch pad which can be used by scripts to implement data structures. It is an address space that can be read and written similar to the ? (*objfil*) and / (*corfil*) spaces. The base address of the scratch portion of debugger space, is returned by the expression: *<scratch*.

A *format* consists of one or more characters that specify a style of printing. Additionally, an optional *format repetition*, **n**, may qualify the format letter. The *format repetition* implements a sort of implied do-loop that is a bit different from the command's execution *count*. Format repetition does not increment *dot* whereas each time a the entire format command is executied, *dot is incremented by the ammount appropriate for the format letter. If specified, this repetition count* must be a decimal number. Without an explicit format repetition, a count of 1 is used. If no format is given, the last format is used. Furthermore, when debugging a CCU, the memory reference made by the processor on behalf of the debugger, will match the width of the format specifier. This feature can be used to avoid bus errors when accessing memory-mapped I/O devices in the CCU hardware that must be accessed as only as bytes, halfwords or words. Note that it is not possible to read in one format and and display in another format, i.e., read as bytes and display as words, for instance. One can, however, copy the data from the CCU using one format into some portion of the scratchpad buffer, then display it using a different format. The original relevance of *dot* is lost when this is done.

The format letters available are:

b[*radix*]

h[*radix*]

w[*radix*] Display a byte, halfword, or word according to the radix specified by *radix*. Values for *radix* are:

x	hexadecimal.
t	signed decimal.
u	unsigned decimal.
q	signed octal.
o	unsigned octal.

Hexadecimal is the default when a radix is not specified. Note that all hexadecimal numbers are printed with a "**0x**" prefix and octal numbers are printed with a "**0**" prefix. The default output radix may be changed by using the $*X* command. or by assigning to variable *oradix*.

a Display the value of *dot* in symbolic form. Useful in conjunction with the **i** or **I** format characters. Symbols are checked to ensure that they have an appropriate type as indicated below.

/	global data symbol.
?	global text symbol.
=	any symbol (text, data, absolute).

c Display the addressed character.

C Display the addressed character using the following escape convention: Character values 0x0 to 0x20 are printed as @ followed by the corresponding character in the range 0x40 to 0x60. The character @ is printed as @@. Non-printable characters whose values are >0x7E are printed as @?.

k Keep on the same line but flush the output (don't print a newline). This is useful in scripts. (Note that in extended formats (see below), the format **h** behaves like format **k**.)

i Display as machine instructions. Integer values in the instructions are printed as hexadecmial. Note: for target isa (**-t68k**), instructions are initially interpreted 2 bytes at a time. *adbccu* interprets MC88K processor instructions four bytes at a time.

I Display as machine instructions, as above except that integer values in the instructions are printed as signed decimal.

n Display a newline.

p Display the addressed value in symbolic form using the same rules for symbol lookup as **a**.

r Display a space.

s Display the addressed characters until a zero character is reached. Dot is not moved when strings are printed.

S Display a string using the **@** escape convention (see **C** above).

A Display the bytes of a string as their ASCII numeric values in the current output radix.

t When preceded by an integer, tabs to the next appropriate tab stop. For example, **8t** moves to the next 8-space tab stop.

Y Display four bytes in date format (see *ctime*(3)).

"..." Display the enclosed string.

^ *dot* is decremented by the current increment multiplied by the *format count*. Nothing is displayed.

+ *dot* is incremented by the *format count*. Nothing is displayed.

– *dot* is decremented by the *format count*. Nothing is displayed.

%[-][*w*][.*p*]*f*

Extended formats. This provides output formats similar to the C language standard I/O function *printf()* . For numeric formats, the field width of the printed format is given by *w*, and the precision, i.e., number of digits is given by *p*, both of which must be given in base 10. One visible difference between the *printf()* formats and *adbccu* formats is in the way hexadecimal formats are printed. *adbccu* prints a "**0x**" prefix when printing hexadecimal numbers, and the precision specifier must account for these two implicit characters. If the - is used, the number will be left-justified within the field. If *precision* is specified, the number will be filled with leading zeros even if the *zero-fill* flag is turned off. (See Miscellaneous command $Z). If width is less than precision, then the width will be determined by the precision.

The supported **%** formats are:

d signed decimal.

D signed decimal.

u unsigned decimal.

U	unsigned decimal.
o	unsigned octal, printed with leading **0**.
O	unsigned octal, printed with leading **0**.
q	signed octal, printed with leading **0**.
Q	signed octal, printed with leading **0**.
X	hexadecimal, printed with leading **0x**.
x	hexadecimal, printed with leading **0x**.
k, h	flush output but don't print newline.
c	print as ASCII character.
***n*m**	Print *n* spaces. If *n* is omitted, nothing is printed.
M	Print the number of spaces equal to the last expression, which will be *dot* if no format expression (see below) is used.
***n*t**	Print to the next *nth* tab stop.
T	Print to the next tab stop specified by the last expression.
Y	Print the last expression in date and time format (see *ctime(3)*).

(exp)f Format expression. This permits expressions to be part of a format specification. A parenthesized expression precededing a format letter or extended format specification will be evaluated left to right within the line. Useful side-effects may thus be created. See EXAMPLES for clarification.

newline Repeat the previous command with a *count* of **1**. This works only for the data formatting commands listed above.

[?/]]g*format value* [*mask*]
Data in *objfil, corfil* or *debugger space* starting at *dot* is masked with *mask* and compared with *value* until a match is found. If no match is found, *dot* is unchanged; otherwise, *dot* is set to the matched location. If *mask* is omitted, a mask of all **1**'s is used. The format specifier *format* determines the amount of data that is examined each time as well as the increment applied to *dot* for commands where *count* is nonzero. For example, **w**, increments *dot* by 4 and will try to match words (32 bit integers). The format specifiers are identical to those in the display commands: **b** (byte), **h** (halfword), and **w** (word).

[?/]]=*format value* ...
Write data into memory starting at *dot*. The format specifier *format* determines the amount of data written each time. The same format specifiers used in the display commands are accepted: **b** (byte), **h** (halfword), and **w** (word). **?** references *objfil*, **/** references *corfil*, and **]** references debugger space.

[?/]m*n b e f*[*name*][?/]
New map segment values for (*b*, *e*, *f*) are recorded. A map segment is an address ranges and its corresponding *objfil* offset. See ADDRESSES for details. *b* is the beginning address of a segment, *e* is the ending address of the segment, and *f* is the offset in the file of the segment. If less than three expressions are given, the remaining map address parameters are left unchanged. The segment map to be modified is indicated by *n* and is required. *name* is the name of the segment that will be displayed with the segment addresses when the map is printed. If no name is given, the segment will be given the name "*unnamed*". If the list is terminated by **?** or **/**, the file (*objfil* or *corfil*, respectively) is used for subsequent requests. (*/m?* causes **/** to refer to *objfil*, for example).

>*name* *dot* is assigned to the variable or register named.

>>*name* a new user-named variable or structure field is created. If only *address* is given, a variable is created and *dot* is assigned to it. If **both** *address* and *count* are specified, a new field definition is added to the built-in table of structure field descriptions. *address* defines the byte-offset of the structure field and its size (in bytes) is defined by *count*. See FIELDS, below, for further details. There are 16 user-definable variables, which can be used as regular variables. User-defined variables and fields may have names of up to 15 characters in length. All variable and field names are stored and matched in lower-case, regardless of their input case. Alpha, numeric, and the "_" (underscore) characters are valid variable and field name characters.

$*modifier* Miscellaneous commands. The available *modifiers* are:

<*f* Read commands from the file *f*. If this command is executed in a file, further commands in the file are ignored. If *f* is omitted, the current input stream is terminated.

<- Rewind the current input file to its beginning and read commands. This permits a script to re-execute itself without knowing its own name. It's also faster than closing the current script file then searching through the filesystem to reopen the same script. One example of how this can be useful is for scripts that implement a menu-selection.

<<*f* Similar to $< except it can be used in a file of commands without causing the file to be closed. There is a limit to the number of << files that can be open at once. If there are two files that need to be executed in the order so that half of file1 then file2 and then the remaining half of file1 are executed, then $<<*file2* should be inserted in file1 at the place where the commands from file2 should be executed. If the second half of file1 does not need to be executed then $<*file2* should be inserted in file1.

>*f* Write output to the file *f*, which is created if it does not exist. Any previous contents of *f* are overwritten. If *f* is omitted, output is returned to the terminal.

>>*f* Append output to the file *f*, which is created if it does not exist.

? Display the MBS (Message Based System) processor id (number) of the processor being debugged. If *corfil* is a memory image file, this command is a no-op. The built-in variable *proc_num* also returns the MBS processor id.

a Toggle absolute (non-symbolic) address display. When the *a* format is used and this flag is set, the numeric value of dot is displayed in the current output radix and zero-fill mode in parentheses preceding the symbol plus offset. This toggle is also available as the built-in variable *absdisplay*.

b Display all breakpoints and their associated counts and commands.

c C stack backtrace. The behavior of this command depends upon the target processor type. For the MC68K CCUs, if *address* is given, it is taken as the address of the current frame instead of the contents of the frame-pointer register. If *count* is given, only the first *count* frames are displayed. For the MC88K CCUs, if *address* is given, it is taken as the address of the stackpointer. If *count* is given, it is used as the instruction pointer. For both processor types, if neither address or count is given, the debugger will attempt to use the registers saved when the breakpoint or crashdump was taken.

e Display the names and values of all external data variables in the current output radix.

E Display the names and values of all external functions (text symbols) in the current output radix.

F Display the names, offsets and sizes of built-in fields. If *address* is a field expression, e.g. "<me_flags", the description of that field alone will be displayed rather than listing the entire table.

I*path* Add *path* to the default search path that is used by *adbccu* when processing the $< or $<< commands. The syntax for specifying *path* is given above in the description of command-line option **-I**.

l Set the maximum offset for symbol matches to *address* (default 32767).

m Display the file address maps. (See ADDRESSES.)

q Exit from *adbccu*. If debugging a running CCU, the CCU will be resumed and all breakpoints removed.

r Display the general and status register(s) and the instruction addressed by the PC in hexadecimal. Registers saved by the CCU executive in its breakpoint handler are displayed. Dot is set to the program counter.

x Set the default input radix to *address* and report the new value. Only the radix values 8, 10 and 16 (base 10) are accepted. Note that *address* is interpreted in the (old) current radix. Thus 10$x never changes the default radix. To make decimal the default radix, use 0t10$x. The input radix is also available through the built-in variable *iradix*.

X Set the default output radix to *address* and report the new value. Legal values are 0 (default old style), 0t8 (octal), 0t10 (decimal), and 0t16 (hexadecimal). When the program starts, the default output radix is hexadecimal. The output radix is also available through the built-in variable *oradix*.

v Print all integer variables whose values are non zero.

V Print all integer variables that have been stored into (valid variables), including those whose value is zero.

w Set the output page width to *address*. The default is 80 columns.

z Zero a variable or destroy a field. *Address* must be a variable or field expression such as '*<a*', '*<msg0*', or '*<my_field*'. For variables, if *count* is specified and it is nonzero, the variable is zeroed, otherwise it is unaffected. If it is a user-defined variable, the variable is reset (undefined), thus freeing one of these 16 nameable variables for reuse under a different name. This command is intended primarily for use with user-defined variables. It can also be used to quickly zero non-integer-sized variables. For example: **<msg0$z** will clear all 64 bytes of the message variable *msg0* and **<scratch$z** will zero the entire scratchpad space. Destroying a field completely removes it from the field table.

Z Toggle the zero-fill flag. When the output radix is hexadecimal and the flag is true, numbers are zero filled. Default value for the flag at startup is 0. This flag is also available as the built-in variable *zerofill*.

:*modifier* Control a running CCU. The available *modifiers* are:

bc Set breakpoint at *address* in the CCU. The breakpoint is executed

count–1 times before causing a stop. The command *c* is executed when the stop occurs. Setting breakpoints in EPROM mode is not possible.

c The CCU is continued and the debugger will wait (command processing is suspended) until a breakpoint is encountered, the MBS timeout expires or the debugger is interrupted by the console interrupts SIGINT (^C) or SIGQUIT (^B). If the CCU had already been continued, and the debugger returned to command level with the **^C**, this command will simply cause the debugger to wait, without sending the continue message to the CCU. If *address* is given, execution is continued at this address, but with the same stack that was in use when the breakpoint was taken or the CCU stopped by a ^B SIGQUIT interrupt. Therefore it is not usually useful to give it an address except when the address would be within the scope of the same stack frame (i.e. to skip over a few instructions.)

C The CCU is continued as in **c** but the debugger returns to execute commands instead of waiting for a breakpoint message from the CCU. If the CCU is already running, this command does nothing.

d Delete breakpoint at *address* in the CCU.

D Delete all breakpoints in the CCU.

E Cause the CCU to enter its diagnostic EPROM. To continue to use *adbccu* to control the CCU, the user will have to force the CCU EPROM into message mode by doing a shell command: ("!sendint fe" on the SPU version of *adbccu*, and "!spucmd sendint fe" on the CPU version).

g Get the next message out of the internal message fifo and copy it into the message variable specified by *address*, which must be a message variable reference (e.g. <msg0:g). If *count* is specified and its value is zero, the command is not executed. If it is nonzero or simply not specified, the command will be executed. If the command was successful, the read-only built-in variable **msgstat** is set to 1. (See VARIABLES below.) Though there are only 16 message variables, the fifo has room to receive 64 messages before overflowing.

j Jump to EPROM entry (diagnostic) or RAM address. If no *address* is given, the command will re-enter EPROM at the self-test diagnostic entry point as it does when the board is reset. If *address* is specified, the CCU_ENTER message will be sent to the CCU with address as the entry point. This causes some board state to be initialized and the PC loaded with the specified RAM address. CCU_ENTER is the message that *loadccu(8)* uses when the CCU is first booted. *count* is ignored in this command.

p Put a copy of the message variable named by *address*, which must be a message variable reference (e.g. <msg0) into the processor queue. If *count* is specified and its value is zero, the command is not executed. The destination processor queue is determined by the current value of the *proc_num* variable. *proc_num* is initialized to be the MBS queue number of the */dev/ccuN* corfil.) If successful, the built-in variable *msgstat* will be 1 when the command is complete, otherwise it will be 0. Due to the way the message system call functions, the only fields in the message header that can be set by the user are **me_dst_id** and the upper 5 bits of the **me_flags** field. Indirectly, (not based on the contents of the message variable) the user can also control **me_dst_prc** by changing the value of the *proc_num* variable, but this is tricky.

P This command is like **:p** except it will wait for a reply message to return and be placed in the message fifo. The reply message is NOT received into the same message variable that is sent; a **:g** command must be used to accomplish this function. If *count* is specified and is non-zero, its value will be used as the MBS timeout during this one message operation. If the timeout is not specified (normal usage), then the default MBS timeout value will be used. If a message is successfully sent AND a reply message was received, *msgstat* will be 1 when the command is complete, otherwise it will be 0. Note: success does not imply that the received message was EXACTLY the reply to the sent message; simply that a reply was received that was not a breakpoint message. If a breakpoint message arrives while waiting for the reply to the **:P** command, the debugger will remember this, and print a "Stopped at ->" message on stdout when the **:P** completes or times-out. Any command that may be part of the breakpoint will be executed at this time.

s Like **:c** except that the CCU is single-stepped *count* times.

S The processor is stepped through a subroutine if the PC is at a subroutine call, otherwise the command functions the same as **:s**.

! Shell command. A shell is created to execute the rest of the line following the *!*. */bin/sh* is used to execute the command.

)*command* Extended commands. The remainder of the command line is interpreted as one of several extended commands:

comment Make a (non-executed) remark. Usually used in *adbccu* command scripts.

help Display a file containing information about *adbccu* functionality.

status Display status about various *adbccu* state, including the current input radix, the current output radix, script search path and the CCU number if *corfil* is a channel control unit.

VARIABLES

adbccu provides a number of integer (4 byte) variables useful in writing scripts. There are 26 single-letter variables (**a** through **z**) and 10 numbered variables (**0** through **9**). In addition to these, there are also 16 variables that may be named and initialized with the **>>** command. For example, "0x500>>my_var" creates a variable named my_var and stores the value 0x500 into it. This command is also used to define structure fields. (See FIELDS for the use of this feature.)

There exist also, 16 message variables, named **msg0** through **msg15**, each of which is a 64 byte MBS message buffer. These buffers are structures defined by *struct msg_entry_def* in the *mbs.h* include file. A message is sent to the CCU using the **:p** (put message) command. Message replies are collected by the debugger into an internal 64-slot fifo. The user receives a reply by copying from the fifo into one of the 16 message variables by using the **:g** (get message) command. The user can fill in a message by assigning to them as variables or storing to debugger space. Contents of a message variable can also be displayed by peeking debugger space or by dereferencing fields in the message variable of interest. See FIELDS for details on syntax for use of data structure fields.

In addition to the message buffers, there are several special-purpose integer variables that are part of the debugger interface to MBS.

msgout is incremented every time a message is successfully sent by the **:p** or **:P** command. Its initial value is zero. It is never reset by the system, but can be reset by the user with the command **0>msgout** or by **<msgout$z**.

msgin is incremented every time a **:g** is successful. Its initial value is zero. **msgin** is never

reset by the system, but can be reset by the user with **0>msgin** or by *<msgin$z*.

msgrdy reflects the number of messages ready in the fifo: that is, messages received by the debugger but not yet received by the user with **:g**. It is incremented when a message is placed in the fifo, and decremented when one is removed with **:g**.

msgstat is a read-only variable that reflects the status of the most recently executed **:g**, **:p** or **:P** command. If the fifo is empty when **:g** is invoked, then the value of *msgstat* is 0. if the **:g** was successful, and a message was successfully transferred out of the fifo into the message variable, *msgstat* will be 1. If **:p** (put a message and don't wait for a reply) successfuly sent the message, *msgstat* will be 1, otherwise it will be zero. For the **:P** command (put a message and wait for a reply to return) *msgstat* will be 1 if a message was sent and a reply was received, otherwise it will be zero.

timeout is the number of seconds that *adbccu* will wait for a reply to an MBS message sent internally, such as for a **:b** command or **:c** commands or sent directly by the user with the synchronous **:P** command. If the timeout expires, the wait is interrupted and a message is printed to standard error.

proc_num
returns the MBS message queue to which messages will be sent when using the **:p** or **:P** commands. It's value is the MBS queue id of the CCU being examined/controlled that is derived from the minor number of the special file.

Other built in, named variables, not related to the MBS features are:

iradix is the current value of the default *input radix*, and provides scripts with a way to query (and set) this state variable. The normal interactive way of setting and displaying *iradix* is with the **$x** command. Of course, the radix of each input number may be individually controlled with the **0x**, **0t** and **0o** prefixes described above.

oradix is the current value of the default *output radix*, and provides scripts with a way to query (and set) this state variable. The normal interactive way of setting and displaying *oradix* is with the **$X** command. Note: one historical artifact is that an *oradix* value of **0**, means the same thing as a value of **16** (decimal).

zerofill is the current value of the *zero-fill* toggle, which is also controlled by the miscellaneous command, **$Z**. If *zerofill* is set (i.e. nonzero), hex numbers are displayed with leading zeros.

absdisplay
is the current value of the *absolute address display* toggle, which is also controlled by the miscellaneous command, **$a**. If *absdisplay* is set, and when the **a** format is used, the numeric value of address will be displayed along with the symbol plus offset form.

scratch The 1K-byte *scratch* buffer is a debugger space memory area that scripts can use to maintain unnamed variables or implement structures. While dereferenced like simple variables, debugger space variables (which include the message variables) are larger than 4 bytes in size. Internally, scratch is implemented as an array of unsigned integers. The value of the expression *<scratch* is the address of this buffer. As with message variables, *scratch* space is accessed with the debugger space command] or by using field notation. Note that the command line option **-s** permits the user to allocate a larger scratchpad, up to 15 pages in size when starting *adbccu* .

ifile This debugger space variable contains the name of last file opened for input. When the debugger is first started, it is *objfil*. After a $< or $<< command is executed, *ifile* contains the name of the script file.

ofile This debugger space variable contains the name of the file opened for output by the $> or $>> commands. If no file is opened for output, it contains a string of null.

Some single-letter named variables are set initially by *adbccu* but are not used subsequently:

0 last value displayed.

9 *count* specified on the $< or $<< command which started the current script.

During startup, *adbccu* sets the following are set from the system header in *objfil* for the CCU targets and from *corfil* for the SPU target.

b base address of the data segment.

d data segment size.

e entry point.

m magic number.

s stack segment size (SPU core only - all others set to 0)

t text segment size.

FIELDS

The field notation is an extension to the syntax for specifying addresses and variables. It is based upon the syntax used in the C programming language for direct component selection of fields within data structures. Like C, the period "." is used as the field operator. The basic syntax is *address.field*, where *field* is an expression, a built-in field name or a user-defined field name. Fields can be used in any address or variable expression. Multiple fields (sub-fields) may be used to access fields in nested structures.

If the last field element is larger than 4 bytes, the whole expression is an *address*. For example, *<msg0.me_udata* returns the address in debugger space, of the 48-byte user data portion (array of 12 integers) of message variable *msg0*.

If the last field element in a field list is a named field whose size is 4 bytes or smaller, the reference is *by value*, meaning that the contents of the addressed field is returned. Therfore field notation can also be used like array subscripts because the last field can be an expression, e.g. **<msg0.me_udata.4** which corresponds to the C expression **msg0.me_udata[4]**. In this case, the whole expression is evaluated as a pointer to an integer and will return the contents of the addressed location. Another example, **<scratch.5>scratch.6** copies the 6th element in the scratch pad to the 7th element, just as the C statement **scratch[6] = scratch[5];** would do.

The built-in byte and halfword fields provide a convenient way for accessing, masking and shifting bytes and shorts within integer fields or objects. For example, **<d0.b0>foo.b1** copies byte 0 (big-endian) from MC68000 register **d0** into byte 1 of user-variable *foo*. Other bytes of *foo* are unmodified. See the EXAMPLES section below for amplification on these features.

There is one special case where a field name can be used without the "." notation and doesn't return a field address or value. The value of an expression containing a single field, e.g. *<me_udata*, is a 32 bit integer whose upper 16 bits contains the field offset and whose lower 16 bits contains the field size. For example, "*<me_udata=w*" will print 0x100030, since the **me_udata** field begins at byte offset 0x10 and is 0x30 bytes in length. The purpose of defining the value of a field in this way is to provide scripts with a syntax for accessing the built-in field definitions. For example, it may be used to zero all but the header part of a message variable. The majority of the built-in field names correspond to the Common Messsage Interface (CMI), a specific MBS message format used in some device drivers.

The built-in field names are:

Name	offset	size	Name	offset	size	Name	offset	size
me_next	0	2	modifier	1	1	address	0	4
me_msgindx	2	2	extend	4	4	seq_no	4	4

me_flags	4	1	dev	28	12	un	40	24
me_owner	5	1	class	0	1	data	0	24
me_dst_prc	6	1	type	1	1	vhandle	0	4
me_src_prc	7	1	unit	2	1	vbuffer	4	4
me_dst_id	8	4	desc	4	8	size	8	4
me_src_id	12	4	block	0	4	sdr0	12	4
me_udata	16	48	partition	6	2	sdr7	16	4
next	0	2	cylinder	4	2	pbuffer	4	4
msgindx	2	2	track	6	1	xbuffer	12	4
flags	4	1	sector	7	1	vbuffer	16	4
owner	5	1	density	0	1	ptable	4	4
dst_prc	6	1	outputnum	1	1	in_filter	20	2
src_prc	7	1	rpcmode	2	1	out_filter	24	2
dst_id	8	4	paddr	0	8	b0	0	1
src_id	12	4	slot	0	1	b1	1	1

Name	**offset**	**size**	**Name**	**offset**	**size**	**Name**	**offset**	**size**
cmd	16	4	bus_num	1	1	b2	2	1
specific	0	1	int_pri	2	1	b3	3	1
common	1	1	int_num	3	1	h0	0	2
code	2	1	csr_base	4	4	h1	2	2
status	20	8	paddr1	0	4	w0	0	4
errcnt	0	1	paddr2	4	4	w1	4	4

ADDRESSES

The address in a file associated with a written address is determined by a set of mappings associated with that file. Each mapping represents a segment, and appears as three numbers – *b*, *e*, and *f*. *b* is the beginning address of the segment, *e* is the ending address of the segment, and *f* is the offset in the file of the segment. The *file address* corresponding to a written *address* in one of the segments is calculated in the following manner:

$$b \leq address < e \implies file\ address = address + f - b$$

If *address* does not fall into one of the segment ranges, it is not considered legal and an access error occurs.

Two initial settings of mappings are made, one for *objfil* and one for *corfil*. The *objfil* mapping defines text and data segments based on information in the file header. The *corfil* map is isa-dependent. For SPU OS core files, mappings are determined by the corefile header. For the CCU memory image core files, mappings are determined by the CCU architecture. The MC68K CCUs have one "data" core file segment that includes the entire 512K RAM. The MC88K CCUs have two segments, "text" and "data", each 256K in size. If either file is not of the kind expected, for that file, *b* is set to **0**, *e* is set to the maximum file size and *f* is set to **0**; in this way, the whole file can be examined with no offset translation.

ADDRESS MAPS

The following tables describe the address maps of each of the Convex CCUs and the formats of their crashdump memory images. It can be used to setup the debugger segment address maps for easily accessing memory mapped I/O space for which segments are not automatically defined by the object file headers. Example scripts for initializing the VIOP *corfil* (/) maps for both the crashdump file and CCU corefile are presented.

VIOP crashdump file map (viop.coreN)

begin	**end**	**file offset**	**segment name**

0x00000000	0x00080000	0x00000000	RAM
0x00ff0000	0x00ff8000	0x00080000	cache dirty bits
0x00ff8000	0x00ff8200	0x0007e000	local RAM map
0x00ff9000	0x00ffa000	0x00088000	cache tag bits
0x00ffa000	0x00ffb000	0x00089000	window registers
0x00ffb810	0x00ffb830	0x0008a000	control registers
0x00ffb850	0x00ffb870	0x0008a020	diagnostic registers
0x00ffba00	0x00ffbc00	0x0008a040	PBUS interrupt map

VIOP /dev/ccu file map

begin	**end**	**file offset**	**segment name**
0x00000000	0x00080000	0x00000000	RAM
0x00ff0000	0x00ff8000	0x00ff0000	cache dirty bits
0x00ff8000	0x00ff8200	0x00ff8000	local RAM map
0x00ff9000	0x00ffa000	0x00ff9000	cache tag bits
0x00ffa000	0x00ffb000	0x00ffa000	window registers
0x00ffb810	0x00ffb830	0x00ffb810	control registers
0x00ffb850	0x00ffb870	0x00ffb850	diagnostic registers
0x00ffba00	0x00ffbc00	0x00ffba00	PBUS interrupt map

IOP crashdump file map (iop.coreN)

begin	**end**	**file offset**	**segment name**
0x00000000	0x00080000	0x00000000	RAM
0x00ff0000	0x00ff2000	0x00080000	cache dirty bits
0x00ff8000	0x00ff8200	0x0007e000	local RAM protection bits
0x00ff9000	0x00ffa000	0x00082000	cache tag bits
0x00ffa000	0x00ffa400	0x00083000	window registers
0x00ffb810	0x00ffb830	0x00083400	control registers
0x00ffb850	0x00ffb870	0x00083420	diagnostic registers
0x00ffba00	0x00ffbc00	0x00083440	PBUS interrupt map

IOP /dev/ccu file map

begin	**end**	**file offset**	**segment name**
0x00000000	0x00080000	0x00000000	RAM
0x00ff0000	0x00ff2000	0x00ff0000	cache dirty bits
0x00ff8000	0x00ff8400	0x00ff8000	local RAM protection bits
0x00ff9000	0x00ffa000	0x00ff9000	cache tag bits
0x00ffa000	0x00ffa400	0x00ffa000	window registers
0x00ffb810	0x00ffb830	0x00ffb810	control registers
0x00ffb850	0x00ffb870	0x00ffb850	diagnostic registers
0x00ffba00	0x00ffbc00	0x00ffba00	PBUS interrupt map

HSP crashdump file map (hsp.coreN)

begin	**end**	**file offset**	**segment name**
0x00000000	0x00080000	0x00000000	RAM

```
0x00ffba00 0x00ffbc00 0x00080000      PBUS interrupt map
```

HSP /dev/ccu file map

begin	end	file offset	segment name
0x00000000	0x00080000	0x00000000	RAM
0x00ffba00	0x00ffbc00	0x00ffba00	PBUS interrupt map

In addition to the crashdump file map segments listed above, the 68K CCU executive saves to RAM other data at crash time, notably the local RAM protection map (segment 0xff8000), general registers and the program counter and status registers. Also, upon entry to EPROM the contents of certain registers are saved in page 0x7f000 of RAM. When the CCU crash results in a return to EPROM, as it often does, these data may be useful in determining the cause of the crash. The following table identifes these saved data and their crashdump file offsets. (Note: In the HSP, only the general registers, IER, ISR, and 16 PMAP entries are saved by the EPROM.)

offset	size	description
0x7f000	0x40	register dump on EPROM entry
0x7f040	0x40	register dump at crash
0x7f080	0x04	program counter at crash time
0x7f084	0x04	status register at crash time (in upper 16 bits)
0x7f088	0x04	Interrupt Enable Register at EPROM entry
0x7f08c	0x04	Interrupt Status Register at EPROM entry
0x7f090	0x04	PBUS error log at EPROM entry
0x7f094	0x04	1st word cache error log at EPROM entry
0x7f098	0x04	2nd word cache error log at EPROM entry
0x7f0a0	0x40	PMAP of first 16 windows at EPROM entry
0x7f0e0	0x40	Cache Tags of first 16 windows at EPROM entry
0x7f120	0x200	Cache dirty bits of first 16 windows at EPROM entry
0x7f3e0	0x800	Cache lines of of first 16 windows at EPROM entry
0x7fbe0	0x04	guard word == 0xdeadbad1

IDC crashdump file map (idc.coreN)

begin	end	file offset	segment name
0x01000000	0x00140000	0x00000000	IRAM
0x02000000	0x00240000	0x00040000	DRAM
0x00c00000	0x00c10000	0x00080000	dma buffers
0x00000000	0x00008000	0x00090000	EEPROM
0x00ffa000	0x00ffb000	0x00098000	window registers
0x00ff7000	0x00ff7200	0x00099000	PIGA register file
0x00ff5000	0x00ff5020	0x00099200	BARB registers

IDC /dev/ccu file map

begin	end	file offset	segment name
0x00000000	0x00008000	0x00000000	EEPROM
0x00c00000	0x00c10000	0x00c00000	dma buffers

0x00ff5000	0x00ff5020	0x00ff5000	BARB registers
0x00ff7000	0x00ff7200	0x00ff7000	PIGA register file
0x00ffa000	0x00ffb000	0x00ffa000	window registers
0x01000000	0x00140000	0x01000000	IRAM
0x02000000	0x00240000	0x02000000	DRAM

TLI crashdump file map (tli.coreN)

begin	end	file offset	segment name
0x01000000	0x00140000	0x00000000	IRAM
0x02000000	0x00240000	0x00040000	DRAM
0x00c00000	0x00c04000	0x00080000	port 0 buffer
0x00c04000	0x00c08000	0x00084000	port 1 buffer
0x00000000	0x00008000	0x00088000	EEPROM
0x00ff7000	0x00ff7200	0x00090000	PIGA register file

TLI /dev/ccu file map

begin	end	file offset	segment name
0x00000000	0x00008000	0x00000000	EEPROM
0x00c00000	0x00c04000	0x00c00000	port 0 buffer
0x00c04000	0x00c08000	0x00c04000	port 1 buffer
0x00ff7000	0x00ff7200	0x00ff7000	PIGA register file
0x00ffa000	0x00ffb000	0x00ffa000	window registers
0x01000000	0x00140000	0x01000000	IRAM
0x02000000	0x00240000	0x02000000	DRAM

HIPPI crashdump file map (hippi.coreN)

begin	end	file offset	segment name
0x01000000	0x00140000	0x00000000	IRAM
0x02000000	0x00240000	0x00040000	DRAM
0x00c00000	0x00c08000	0x00080000	port 0/1 buffers
0x00000000	0x00008000	0x00088000	EEPROM
0x00ffa000	0x00ffb000	0x00090000	window registers
0x00ff7000	0x00ff7200	0x00091000	PIGA register file
0x00ff5000	0x00ff5008	0x00091200	BARB control registers
0x00ff5010	0x00ff5018	0x00091208	BARB block count registers

HIPPI /dev/ccu file map

begin	end	file offset	segment name
0x00000000	0x00008000	0x00000000	EEPROM
0x00c00000	0x00c08000	0x00c00000	port 0/1 buffers
0x00ff5000	0x00ff5008	0x00ff5000	BARB control registers
0x00ff5010	0x00ff5018	0x00ff5010	BARB block count registers
0x00ff7000	0x00ff7200	0x00ff7000	PIGA register file
0x00ffa000	0x00ffb000	0x00ffa000	window registers
0x01000000	0x00140000	0x01000000	IRAM
0x02000000	0x00240000	0x02000000	DRAM

EXAMPLES

Display the contents of message variable 0 as four rows of four integers:

<msg0,4=]4w

Display the contents of the me_udata part of msg0 (12 integers):

<msg0.me_udata,0t12]w

Copy contents of message variable 0 to message variable 1 using contents of variable i "<i" as an index.

0>i;<msg1,0x10]=w(<msg0.(((<i+1)>i)-1))

FILES

/dev/ccu
/usr/include/interface/msg_if/mbs/mbs.h

SEE ALSO

ccu(4), adb(1), cdown(8), msg_oper(SPU), crashdump(8)

DIAGNOSTICS

"adbccu" when there is no current command or format. Comments about inaccessible files, syntax errors, abnormal termination of commands, etc. Exit status is **0**, unless the last command failed or returned nonzero status.

BUGS

Output radix signed octal (q) doesn't print negative octal numbers if the format width is h or b. Floating point formats are not supported. Longword formats are not supported (yet). Local variables are not accessible symbolically; only external variables and functions are visible. If the send message command *:p* is used in scripts on a running CCU that also has breakpoints set, it is possible (though rare) for the debugger to confuse returned messages. Symbol types are not checked for **?** or **/** or **=** in the **a** format as documented. Option 1 is not yet implemented.

NAME

ansidaemon – ANSI-labelled-tape processing filter/daemon

SYNOPSIS

/usr/lib/tape/ansidaemon -l *VSN login access physdev*
/usr/lib/tape/ansidaemon -u *login physdev*
/usr/lib/tape/ansidaemon -d *RPCprognum physdev reqhost symlink*

DESCRIPTION

ansidaemon is a program that is only to be invoked by the master daemon of the tape subsystem (*tpdaemon*(8)). This program is almost exclusively an RPC server which will perform I/O requests for ANSI-labelled tapes. It handles all processing of the ANSI magnetic tape labels, the blocking and unblocking of logical records, as well as the translation of *ioctl* commands into a similar functionality for ANSI tapes (where possible). With ANSI-labelled tapes, attempts to read beyond the last file in the fileset yields the error ENOENT "No such file or directory".

CONVEX conforms roughly to Level 3 of the *ANSI X3.27-1978* specification. In addition to the fixed (**F**) and variable (**D**) length record formats that are defined within the ANSI specification, there is also support for the old **U** ("undefined") format. CONVEX currently defines this to operate as raw tape would; that is, for each *read* or *write* system call, one physical block is read from or written to tape, without any interpretation of the data included in that block. This format seems useful in creating a binary file that is surrounded by ANSI labels.

The label (**-l**) and unlabel (**-u**) options are designed to operate only on one physical tape volume, but the driver (**-d**) mode processes I/O requests for an entire ANSI tapeset (possibly more than one physical tape).

Options:

-l Labels an unlabeled tape. The first three arguments are used in creating the initial VOL1 label for the tape.

VSN is a string (up to six characters) to be used as the unique volume identifier, or volume serial number.

login is the login name of the user who invoked *tplabel*(1). It is converted into uppercase and used as the owner of the newly initialized ANSI volume.

access is a single character that is placed directly into the VOL1 accessibility field. It should be a blank if the volume is not to have restricted access; this is the default condition.

physdev
specifies the physical path of the drive where the tape has been mounted.

-u Unlabels an ANSI-labelled tape. Only the owner of a tape or the superuser is allowed to execute the *tpunlabel*(1) command, which results in the *ansidaemon* being invoked with this option.

login is the login name of the user running *tpunlabel*(1). It is converted to uppercase before being compared with the owner that has been stored in the tape's VOL1 label when verifying authorization to unlabel the mounted tape.

physdev
specifies the physical tape drive that contains the tape being unlabeled.

-d Invokes the daemon in driver mode. The daemon is automatically started by *tpdaemon*(8) once a labelled tape has been mounted. There is one *ansidaemon* running per active (mounted) ANSI tapeset. The *ansidaemon* program then serves as a pseudo device driver which will accept *open, close, read, write,* and *ioctl* calls from the kernel and translate each request into its ANSI equivalent. Labels are transparently written on output, and verified on input. Existing programs such as *cp*(1) and *cat*(1) are able to write

ANSI-labelled tapes " transparently" using this interface. The required arguments are listed below:

RPCprognum
is the RPC(3) program number of this invocation of the *ansidaemon*.

physdev
as before, this is the path to the physical tape device where the first tape in the ANSI tapeset is mounted.

reqhost is the hostname of the machine originating the request to use the device.

symlink is the symbolic link name associated with this ANSI tapeset (given by the user in the *tpmount*(1) command).

SEE ALSO

tpmount(1), tpunmount(1), tpattr(1), tplabel(1), tpunlabel(1), tpqueue(1), tpwait(1), tape(3), tpconfig(8), tpdaemon(8), mt(1)
ANSI X3.27-1978: Magnetic Tape Labels and File Structure for Information Interchange.

NAME

arp – address resolution display and control

SYNOPSIS

arp *hostname*
arp -a [*vmunix*] [*kmem*]
arp -d *hostname*
arp -s *hostname ether_addr* [**temp**] [**pub**] [**trail**]
arp -f *filename*
arp -e *filename*

DESCRIPTION

The *arp* program displays and modifies Internet-to-Ethernet address translation tables used by the address resolution protocol (*arp*(4p)).

With no flags, the program displays the current ARP entry for *hostname*. The host may be specified by name or by number, using Internet dot notation. With the **-a** flag, the program displays all of the current ARP entries by reading the table from the file *kmem* (default */dev/kmem*) based on the kernel file *vmunix* (default */vmunix*).

With the **-d** flag, a super-user may delete an entry for the host called *hostname*.

The **-s** flag is given to create an ARP entry for the host called *hostname* with the Ethernet address *ether_addr*. The Ethernet address is given as six hex bytes separated by colons. The entry will be permanent unless the word **temp** is given in the command. If the word **pub** is given, the entry will be "published"; i.e., this system will act as an ARP server, responding to requests for *hostname* even though the host address is not its own. The word **trail** indicates that trailer encapsulations may be sent to this host.

The **-f** flag causes the file *filename* to be read and multiple entries to be set in the ARP tables. Entries in the file should be of the form

hostname ether_addr [**temp**] [**pub**]

with argument meanings as given above.

The **-e** flag causes the file *filename* to be read and multiple entries to be set in the ARP tables. Entries in the file should be of the form

ether_addr hostname [**temp**] [**pub**] [**trail**]

with argument meanings as given above. Note that this flag is the same as the **-f** flag except that the first two fields in the file to be read are reversed. This format is suitable for reading the **/etc/ethers** file used by the yellow pages.

FILES

/vmunix	Kernel name list
/dev/mem	Kernel data values
/etc/ethers	For permanent arp entries
*/lib/kernsyms/symdata_**	Kernel symbol addresses

SEE ALSO

inet(3N), arp(4P), ethers(5), ifconfig(8C)

NOTES

arp is an optional product; for more information, contact your CONVEX sales representative.

NAME

avail – maintain and process system availability information

SYNOPSIS

avail [**–m addr**] [**–t**]

DESCRIPTION

avail maintains, processes, and delivers availability information on CONVEX systems. */usr/spool/convex/avail* should only be invoked automatically from root's crontab at 15 minute intervals. To activate the availability reporting system, the following line should be added to */usr/lib/crontab*

00,15,30,45 * * * * /usr/spool/convex/avail

and the following line should be added to */etc/rc.local*

/usr/spool/convex/reboot_script < /dev/console > /dev/console

When activated, the system will write a status line to */usr/spool/convex/availlog* containing local time, number of users, and 15 minute load average. At multi-user reboot, information is logged to */usr/spool/convex/reboot_log*. This data includes the local time, the results of a call to *sysinfo(2)*, and information provided to *shutdown(8)*, if any.

If the file, */usr/spool/convex/avail.conf* exists, it's contents, in addition to an option to input free-form text, are used to generate a menu of shutdown reasons at reboot. This menu is displayed and *avail(8)* will wait for a response for 2 minutes before timing out and continuing the reboot. If a reason is chosen, it is added to */usr/spool/convex/reboot_log*. This file contains a set of default reasons, but may be edited by system administrators.

At 00:00:00 each Sunday morning, the following SPU files are compared with those from the previous week and differences are recorded:

/mnt/usr/scn/cop.out
/mnt/DIAG_DB_REV
/mnt/DIAG_REV
/ioconfig
/mnt/usr/scn/cop.mem
/mnt/usr/lib/softlog
/UNIX_REV
/mnt/errlog
/mnt/usr/ucode/UCODE_REV

These differences, */usr/spool/convex/reboot_log*, and */usr/spool/convex/availlog* are either mailed to an address at CONVEX, or archived in a *tar(1)* format, according to options below.

The command line options to *avail(8)* may be added to the line in */usr/lib/crontab*. The **–m** option specifies an alternate mail address for availability data. By default, the data is mailed to "convex!avail".

The **–t** option specifies that availability data be archived using *tar(1)*. If this option is used, a file with the following format will be produced.

/usr/spool/convex/avail.m.d.y

Where m = month, d = day, and y = year.

SEE ALSO

shutdown(8), sysinfo(2), tar(1)

NAME

catinfo – create the formatted files for the info system

SYNOPSIS

/usr/convex/catinfo

DESCRIPTION

catinfo creates the preformatted versions of the on-line info system screens from the nroff input files. Each command description file used by the CONVEX Information System is successively run through *nroff*, and the resulting formatted output is placed in a corresponding file in a separate subdirectory.

The input files, each named according to the command it describes, are found in /usr/infosys/screens. The output files, whose names are formed by appending a '.f' to the input filename, are placed in /usr/infosys/fscreens.

Similarly, the information screens provided with any optional products your site might have, are also processed by *nroff*. These optional input files are found in /usr/infosys/optscreens, with the corresponding formatted output files being placed in /usr/infosys/optfscreens.

FILES

/usr/infosys/screens/*	raw (nroff input) information screens
/usr/infosys/fscreens/*.f	formatted (nroff output) information screens
/usr/infosys/optscreens/*	raw (nroff input) optional information screens
/usr/infosys/optfscreens/*.f	formatted (nroff output) optional information screens

SEE ALSO

info(1)

NAME

catman – create the cat files for the manual

SYNOPSIS

/etc/catman [**–p**] [**–n**] [**–w**] [sections]

DESCRIPTION

Catman creates the preformatted versions of the on-line manual from the nroff input files. Each manual page is examined and those whose preformatted versions are missing or out of date are recreated. If any changes are made, *catman* will recreate the **/usr/lib/whatis** database.

If there is one parameter not starting with a '–', it is interpreted to be a list of manual sections in which to look. Each character in that list can be the name of any "chapter" in the /usr/man directory, where each character *X* specifies the subdirectory man*X*. The specified sections do not have to be numeric. For example

catman 123l

will cause the updating to only happen to manual sections 1, 2, and 3, and the /usr/man/manl "chapter."

Options:

–n prevents creations of **/usr/lib/whatis**.

–p prints what would be done instead of doing it.

–w causes only the **/usr/lib/whatis** database to be created. No manual reformatting is done.

FILES

/usr/man/man?/*.*	raw (nroff input) manual sections
/usr/man/cat?/*.*	preformatted manual pages
/usr/lib/makewhatis	commands to make whatis database

SEE ALSO

man(1)

NAME

chall – change mode, owner and/or group of a file

SYNOPSIS

chall [-m *mode*] [-g *group*] [-o *owner*] [-l] *filename...*

DESCRIPTION

Chall changes the access mode, owner-ID, and group-ID of the list of *filenames* to the values specified for *mode, owner* and *group* . Any combination of -l, -m, -o, and -g may be specified. If a *filename* is a directory then the modifications are made recursively in that directory. The *group* may be either a decimal group ID or a name found in the */etc/group* file. The *owner* may be either a decimal user ID or a name found in the */etc/passwd* file. The access *mode* may be either an octal value or a mode string as described in *chmod(1)* .

The user invoking *chall* must be the super-user.

The -l flag specifies that action on symbolic links, both to normal files and directories, be allowed. Without this flag, links encountered will have -o and -g changes applied, but not -m. With this flag, -m changes, along with directory traversal, are allowed.

FILES

/etc/group
/etc/passwd

SEE ALSO

chgrp(1), chmod(1), group(5), passwd(5), chown(8)

NAME

chown – change owner

SYNOPSIS

chown [**–Rf**] owner[.group] file ...

DESCRIPTION

Chown changes the owner of the *files* to *owner*. The owner may be either a decimal UID or a login name found in the password file. An optional group may also be specified. The group may be either a decimal GID or a group name found in the group-ID file.

In order to simplify accounting procedures, only the super-user can change owner.

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

When the **–R** option is given, *chown* recursively descends its directory arguments setting the specified owner. When symbolic links are encountered, their ownership is changed, but they are not traversed.

SEE ALSO

chgrp(1), chown(2)

NAME

clri – clear i-node

SYNOPSIS

/etc/clri filesystem i-number ...

DESCRIPTION

N.B.: *Clri* is obsoleted for normal file system repair work by *fsck*(8).

Clri writes zeros on the i-nodes with the decimal *i-numbers* on the *filesystem*. After *clri*, any blocks in the affected file will show up as 'missing' in an *fsck*(8) of the *filesystem*.

Read and write permission is required on the specified file system device. The i-node becomes allocatable.

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

NOTES

clri understands that file systems may be larger than two gigabytes in size, and is able to correctly work in such situations.

SEE ALSO

fsck(8)

BUGS

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

NAME

comsat, in.comsat – biff server

SYNOPSIS

/usr/etc/in.comsat [**–t** *timeout*]

DESCRIPTION

comsat is the server process which listens for reports of incoming mail and notifies users if they have requested this service. It is invoked as needed by *inetd*(8C), and times out if inactive for two minutes. The **–t** option can be specified to use a different timeout length than the default. The timeout length is specified in minutes.

comsat listens on a datagram port associated with the "biff" service specification (see *services*(5)) for one line messages of the form

user@ mailbox-offset

If the specified *user* is logged in to the system and the associated terminal has the owner execute bit turned on (by a **biff y** command, the *offset* is used as a seek offset into the appropriate mailbox file and the first 7 lines or 560 characters of the message are printed on the user's terminal. Lines which appear to be part of the message header other than the "From", "To", "Date", or "Subject" lines are not included in the displayed message.

"localhost" must be defined in */etc/hosts* in order for *comsat* to work properly.

FILES

/etc/utmp contains data on who is logged in to what terminals.

SEE ALSO

biff(1), inetd(8C)

BUGS

The message header filtering is prone to error.

Users should be notified of mail arrives on machines other than the one they are currently logged in to.

The notification should appear in a separate window so it does not alter the screen.

NAME

connecttime – compute connect time from wtmp and bill-acct

SYNOPSIS

connecttime [**-o** *time*] [**-n** *time*] [**-w** *wtmpfile*] [**-b** *billfile*]

DESCRIPTION

connecttime takes the log files generated by *login* and *bill* and merges them. The output of *connecttime* is typically used as the input to the *connecttime.awk* script, which is described in *sumscripts*(8).

connecttime produces two types of entries. The first type begins with the keyword "started" and indicates either a login or a user billing to a new account. The second type begins with the keyword "terminated" and indicates either a logout or a user ending billing to an account. Note that the execution of a single *bill* command will show up as two entries: one for the termination of billing to the old account and one for the initiation of billing to the new account.

The formats of these two types are described below.

started	*time*	*tty*	*uid*	*gid*	*aid*	*remote*
terminated	*time*	*tty*				

"Started" and "terminated" appear as text. *time* is the time as returned by *time()*; *tty*, *uid*, *gid*, and *aid* are self-explanatory. If the user logged on from another system, this will show up in the *remote* field.

The command-line options are:

-b *billfile* Specify a bill log file to use instead of the default file */usr/adm/bill-acct*.

-n *time* Specify the time of the newest entry. Entries made in the log files after this time will be ignored.

-o *time* Specify the time of the oldest entry. Entries made in the log files before this time will be ignored.

-w *wtmpfile* Specify a login log file to use instead of the default file */usr/adm/wtmp*.

NOTE

The *awk* script *connecttime.awk* counts remote logging in its *connectime* calculation. If this is not desired, a locally modified version of *connecttime.awk* may be used, that will compute connect time in a way that is acceptable to a given installation.

SEE ALSO

bill(1), time(3c), bill-acct(5), utmp(5), sumscripts(8),
"Accounting" chapter in the *CONVEX System Manager's Guide*.

BUGS

connecttime ignores date change entries in *wtmpfile*.

NAME

convst – convert stripecap file to new *VVM* supported format

SYNOPSIS

convst < *old_stripecap* > *new_stripecap*

DESCRIPTION

This is an executable which will convert existing stripecap files to the new format which supports *Virtual Volume Manager (VVM)* stripes. This utility should be run once; when the system is upgraded to *VVM* capability.

In addition to converting the stripecap file to the new format the utility will also be responsible for translating physical block designations to logical format. There are no switches or parameters, *stdin* and *stdout* are utilized for I/O.

NOTES

This utility performs significant error detection. Invalid entries in the input stream are not copied to output. Note that comments in the input file are lost and a new file header is established.

The user must have root privilege to execute this utility.

Entries in */etc/stripecap* are created with the *newst*(8) utility.

FILES

/etc/stripecap Database of the stripe descriptors.

SEE ALSO

st(4), stripecap(5), newst(8), getst(8), putst(8), mvst(8), rmst(8), qst(8)

NAME

cpuconf – CPU Configuration

SYNOPSIS

/etc/cpuconf [**-e** *cpuid* | **all**] [**-d** *cpuid* | **all**]

DESCRIPTION

If no arguments are given, the current cpu configuration is printed. If a cpuid is specified, the specified cpu is either enabled or disabled. The *-e* flag is used to enable UNIX process scheduling on a specified cpu. (UNIX is a registered trademark of UNIX System Laboratories, Inc.) The *-d* flag is used to disable UNIX process scheduling on a specified cpu. For example, given a 4 cpu system:

```
cpuconf -d0 -e 2 -d1
```

Disables process scheduling on cpus 0 and 1, and enables process scheduling on cpu 2. Process scheduling on cpu 3 remains unchanged.

The command:

```
cpuconf -d all
```

will disable all but one cpu in the system.

This command can be executed at any time without any adverse effects on the systems operation. The disabling of a cpu only inhibits UNIX scheduling of the CPU and does not disable interrupt processing.

DIAGNOSTICS

'cpuconf: cpuconf set failed Not owner' if you try to change the cpu configuration but are not the superuser.

'cpuconf: Attempt to turn off all CPUs' if you try to disable every cpu in the system.

NAME

crashdump – write a crash dump to tape

SYNOPSIS

crashdump [**-DSsuH**] [**+Mcm**] [**-C** *cop.out*] [**-I** *ioconfig*] [**-v** *vmunix*] [**-i** *iop*] [**-h** *hsp*] [**-V** *viop*] [**-a** *acm201.x00*] [**-d** *idc*] [**-L** *bootcmd.local*] [**-B** *bootcmd*] [**-T** *tunables*] [**-t** *tli*]

DESCRIPTION

Taking a crash dump is recommended after a ConvexOS crash or hang, as it may provide the data necessary to determine and fix the cause of the problem.

crashdump writes a crash dump to a local tape drive. ***crashdump* is executed from the SPU of a machine whose JP is down and should not be executed while the JP is running.**

A predetermined set of files is transmitted whose default path names are */mnt/usr/scn/cop.out*, */ioconfig*, *vmunix*, *iop*, *hsp*, *viop*, *acm201.x00*, *idc*, *bootcmd.local*, *bootcmd*, *tunables* and *tli*. These files may be specified explicitly by using the *-C*, *-I*, *-v*, *-i*, *-h*, *-V*, *-a*, *-d*, *-L*, *-B*, *-T* and *-t* switches, respectively. If these files cannot be found in the current directory, *crashdump* looks for them in */mnt/os*; if this fails *crashdump* will print an error message and continue.

The *-D* switch enables the output of debug information. (Note: in previous releases, this was the '-d' switch, which is now used for the optional specification of the IDC disk controller executable pathname.)

The *-S* switch allows the user to select a tape drive other than the default tape drive.

The *-s* switch forces the crash dump to be written on the SPU cartridge tape drive.

The *-u* switch disables unloading of the tape after the crash dump is complete.

The *-H* switch will invoke the hardware dump facility on the SPU. It generates an output file, *hwdump.out*, which will be included with the other miscellaneous files in the *tar* format section of the tape.

The *+M* switch disables the dumping of miscellaneous files to tape.

The *+c* switch disables the dumping of ccu core files to tape.

The *+m* switch disables the dumping of main memory to tape.

crashdump determines whether or not the machine contains a tape drive by looking in */ioconfig*. The default behavior is to dump to a JP tape drive if such a tape drive exists. If not, the *-s* switch must be used to write to the SPU cartridge tape drive. If */ioconfig* contains more than 1 tape controller, and the *-S* option has not been specified, *crashdump* will use the first entry for a 9-track tape drive in */ioconfig*.

SEE ALSO

crashdump(5), crashread(8)

NAME

crashread – read a crash dump tape

SYNOPSIS

crashread [**-H**] [**-M**] [**-m**] [**-s**] [**-f** *tape_device*] [**-h** *hsp_num*] .. [**-d** *idc_num*] .. [**-i** *iop_num*] .. [**-t** *tli_num*] .. [**-v** *viop_num*] ..

DESCRIPTION

crashread is used to read data from a crash dump tape. Files are created in the current directory to hold the dump data. When reading a dump tape, it is best to create a new directory and make the new directory the current directory before invoking *crashread*.

The tape device that the tape is mounted on is specified with the *-f* switch. The default is */dev/rmt20*. If the *-s* switch is used, the SPU tape drive will be used, and the default tape drive will be *SPU:/dev/rmt1*.

The default action of *crashread* is to read everything on the tape(s). Specific items may be requested by command-line switches. If any specific items are requested via arguments, then only those items specifically requested are read. For crash dumps occupying more than one tape, the user will be prompted to mount the appropriate tape before anything is read.

Items may be specifically requested using the following switches:

-H	Header file.
-M	The miscellaneous files such as *ioconfig* and *vmunix*.
-m	Main memory. *crashread* compresses zero-filled pages and holes in the PCM using *lseek*(2) to save disk space.
-h hsp_num	Memory image of HSP number *hsp_num*.
-d idc_num	Memory image of IDC number *idc_num*.
-i iop_num	Memory image of IOP number *iop_num*.
-t tli_num	Memory image of TLI number *iop_num*.
-v viop_num	Memory image of VIOP number *viop_num*.

The 64K *MBS memory* file mentioned in *crashdump*(5) is treated by *crashread* the same as the first 64K of main memory. Whenever *crashread* reads the first 2MB of memory, it also reads the *MBS memory* file and writes it into the location in *mm.core* corresponding to the real MBS memory area. (This location happens to be location 0.)

FILES

Crashread creates the following files in the current directory:

bootcmd	*bootcmd* from the SPU
bootcmd.local	*bootcmd.local* from the SPU, if found
commregs	Communication registers for a C2
cop.out	*/mnt/usr/scn/cop.out* from the SPU
errlog	*/mnt/errlog* from the SPU.
header*X*	Header for tape number *X*.
hsp	HSP executable.
hsp.core*X*	Core image for HSP number *X*.
hwdump.out	Output from hardware dump, if taken
ioconfig	*ioconfig* from the SPU.
iop	IOP executable.
iop.core*X*	Core image for IOP number *X*.
viop	VIOP executable
viop.core*X*	Core image for VIOP number *X*.
idc	IDC executable

idc.core*X* Core image for IDC number *X*.
mm.core Main memory core image.
registers CPU registers
The *registers* file will be present **only** if a hardware dump was taken. If a hardware dump was not taken, *hwdump.out* will contain the CPU registers.
tli TLI executable
tli.core*X* Core image for TLI number *X*.
tunables *tunables* from the SPU
vmunix *vmunix* from the SPU.

WARNING

It is easy to make the mistake of reading different crash dumps into the same directory. Check the contents of the current directory before using *crashread*.

SEE ALSO

dd(1), mt(1), tar(1), tpmount(1), tpunmount(1), crashdump(5), crashdump(8)

NAME

ctar – cartridge tape archiver

SYNOPSIS

ctar *[key] [name ...]*

DESCRIPTION

ctar saves and restores multiple files on a single file SPU cartridge tape. It is functionally identical to *tar* except that the only permitted archive device is the SPU cartridge tape. *ctar*'s actions are controlled by the *key* argument. The *key* is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to *ctar* are file or directory *names* specifying which files to dump or restore. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

To use *ctar,* one's user ID must be that of the superuser.

The function portion of the *key* is specified by one of the following letters:

c Create a new tape; writing begins at the beginning of the tape, instead of after the last file.

r The named files are written after the last file on the tape.

x The named files are extracted from the tape. If the named file matches a directory whose contents had been written onto the tape, this directory is (recursively) extracted. The owner, modification time, and mode are restored (if possible). If no file argument is given, the entire content of the tape is extracted. Note that if multiple entries specifying the same file are on the tape, the last one overwrites all earlier.

t The names of the specified files are listed each time they occur on the tape. If no file argument is given, all of the names on the tape are listed.

u The named files are added to the tape if either they are not already there or have been modified since last put on the tape.

o On output, *ctar* normally places information specifying owner and modes of directories in the archive. Former versions of *ctar,* when encountering this information will give error message of the form

"<name>/: cannot create"

This option will suppress the directory information.

p This option says to restore files to their original modes, ignoring the present *umask*(2). Setuid and *sticky*(8) information will also be restored to the superuser.

The following characters may be used in addition to the letter which selects the function desired.

v Normally *ctar* does its work silently. The **v** (verbose) option make *ctar* type the name of each file it treats preceded by the function letter. With the **t** function, the verbose option gives more information about the tape entries than just their names.

w *ctar* prints the action to be taken followed by filename, then wait for user confirmation. If a word beginning with 'y' is given, the action is done. Any other input means don't do it.

f *ctar* uses the next argument as the name of the archive instead of */dev/rct0e*. Note that the default device on a C2 is */dev/rmt0* or */dev/rmt1*. If the name of the file is –, *ctar* writes to standard output or reads from standard input, whichever is appropriate. Thus, *ctar* can be used as the head or tail of a filter chain. *ctar* can also be used to copy hierarchies with the command:

```
cd fromdir; ctar cf - . | (cd todir; ctar xf -)
```

b *ctar* uses the next argument as the blocking factor for tape records. The default is 238. The block size is determined automatically when reading tapes (key letters **x** and **t**)

l Tells *ctar* to complain if it cannot resolve all of the links to the files dumped. If this is not specified, no error messages are printed.

m Tells *ctar* not to restore the modification times. The modification time will be the time of extraction.

h Forces *ctar* to follow symbolic links as if they were normal files or directories. Normally, *ctar* does not follow symbolic links.

B Forces input and output blocking to 238 blocks per record. This option was added so that *ctar* can work across a communications channel where the blocking may not be maintained.

i Causes *ctar* to ignore directory checksum errors while reading.

If a filename is preceded by **–C**, then *ctar* will perform a *chdir*(2) to that filename. This allows multiple directories not related by a close common parent to be archived using short relative path names. For example, to archive files from */usr/include* and from */etc*, one might use

```
ctar c -C /usr include -C / etc
```

Previous restrictions dealing with *ctar*'s inability to properly handle blocked archives have been lifted.

FILES

/dev/rct0? (SPU device)
/dev/rmt?
*/tmp/tar**

SEE ALSO

tar(1), su(1)

DIAGNOSTICS

Complaints about bad key characters and tape read/write errors.
Complaints if enough memory is not available to hold the link tables.

BUGS

There is no way to ask for the nth occurrence of a file.
Tape errors are handled ungracefully.
The **u** option can be slow.
The current limit on filename length is 100 characters.
There is no way to selectively follow symbolic links.

NAME

dcheck – file system directory consistency check

SYNOPSIS

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

DESCRIPTION

N.B.: *Dcheck* is obsoleted for normal consistency checking by *fsck*(8).

Dcheck reads the directories in a file system and compares the link-count in each i-node with the number of directory entries by which it is referenced. If the file system is not specified, a set of default file systems is checked.

The **–i** flag is followed by a list of i-numbers; when one of those i-numbers turns up in a directory, the number, the i-number of the directory, and the name of the entry are reported.

The program is fastest if the raw version of the special file is used, since the i-list is read in large chunks.

FILES

Default file systems vary with installation.

SEE ALSO

fsck(8), icheck(8), fs(5), clri(8), ncheck(8)

DIAGNOSTICS

When a file turns up for which the link-count and the number of directory entries disagree, the relevant facts are reported. Allocated files which have 0 link-count and no entries are also listed. The only dangerous situation occurs when there are more entries than links; if entries are removed, so the link-count drops to 0, the remaining entries point to thin air. They should be removed. When there are more links than entries, or there is an allocated file with neither links nor entries, some disk space may be lost but the situation will not degenerate.

BUGS

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

Dcheck is obsoleted by *fsck* and remains for historical reasons.

NAME

diskuse – compute user and group disk-space totals

SYNOPSIS

diskuse [**-i**] [**-d** *dir*]

DESCRIPTION

Diskuse recursively visits every file within a given directory and computes kilobyte totals for each user ID and group ID that owns files in the hierarchy. The top of the directory hierarchy is assumed to be the current directory, unless an alternate directory is specified with the **-d** option. If the **-i** option is given, diskuse ignores any NFS mounted directories in the hierarchy. Using the **-i** option will **significantly** increase the time it takes to run *diskuse*.

The output of *diskuse* is in a form suitable for processing by the *disk** awk scripts described in *sumscripts*(8).

SEE ALSO

du(1), sumscripts(8),
"Accounting Reports" chapter in *Managing ConvexOS:Operations Guide.*

NAME

dump, vdump – incremental filesystem dump

SYNOPSIS

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

DESCRIPTION

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

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

a Makes *dump* skip over any files that have changed between the time that the *dump* was started and the time when *dump* actually copies the file. *dump* will not skip over directories and will print out a short message with the inode number of any files that were skipped. The **a** option is recomended when a mounted filesystem is *dumped,* since it will prevent invalid files from being written to the tape.

f Place the dump on the next *argument* file instead of the tape. If the name of the file is –, *dump* writes to standard output. The default is */dev/rmt8.*

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

s The size of the dump tape is specified in feet. The number of feet is taken from the next *argument.* When the specified size is reached, *dump* will wait for reels to be changed. The default tape size is 2300 feet.

d The density of the tape, expressed in BPI, is taken from the next *argument.* This is used in calculating the amount of tape used per reel. The default is 1600.

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

w Is like *W,* but prints only those filesystems which need to be dumped.

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

b Take the next *argument* to be the blocking factor for tape records. This number indicates how many K-bytes are in a tape record. The default is 5.

G Set defaults for a GCR (6250 bpi) dump. This sets the tape unit to be */dev/rmt16,* the density to 6250, and the blocking factor to 50.

I Puts the tape drive into 100 ips streaming mode. The default is 50 ips.

E Causes *dump* to write to tape until finished or until end-of-tape is reached. If end-of-tape is reached, the operator is prompted to change tapes. Even with this option, *dump* will print an estimate of how many 9-track tapes (not cartridges) will be used for the dump.

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

Since making a full dump is time-consuming, *dump* checkpoints itself at the start of each tape volume. If writing that volume fails, *dump* will, with operator permission, restart itself from the checkpoint after the old tape has been rewound and removed, and a new tape has been mounted.

Since *dumps* are used to store potentially important data, filesystems should be unmounted when they are dumped. Mounted and active filesystems are the main causes of unreadable *dump* tapes. If it is not possible to unmount a filesystem, then *vdump* with the **a** option should be used.

dump tells the operator what is going on at periodic intervals, including estimates of the number of blocks to write, the number of tapes it will take, the time to completion, and the time to the tape change.

For each filesystem at your site, establish a sequence of full and incremental dumps on a weekly cycle. Because backing up and restoring a filesystem is a time-consuming process, the sequence you select has an impact on the amount of time required to perform backups and to restore files from dump tapes.

One possible backup schedule for performing dumps follows:

Day	**Dump Level**
Monday	level 0: backs up all files in the filesystem (full dump)
Tuesday	level 5: backs up all files changed since Monday's full dump
Wednesday	level 5: backs up all files changed since Monday's full dump
Thursday	level 5: backs up all files changed since Monday's full dump
Friday	level 5: backs up all files changed since Monday's full dump

With this method, restoring files requires only two sets of dump tapes: tapes for the full dump and tapes for the most recent incremental dump.

vdump performs the same function as *dump* but will also run *restore (8)* with the **V** option after the *dump* finishes. *restore (8)* in the verify mode checks the format of the *dump* tape and makes sure that the tape is readable. It is highly recomended that *vdump* with the **a** option be used for dumping filesystems that cannot be unmounted.

FILES

/dev/rmt8	default tape unit to dump to
/etc/ddate	old format dump date record (obsolete after **–J** option)
/etc/dumpdates	new format dump date record
/etc/fstab	dump table: filesystems and frequency
/etc/group	to find group *operator*

NOTES

dump understands that file systems can be larger than two gigabytes in size, and correctly dumps these file systems. The dump tape format has not changed, so files smaller than two gigabytes may be restored on systems that do not support large files.

SEE ALSO

mtio(4), dump(5), fstab(5), restore(8), xdump(8)

DIAGNOSTICS

Unlike most other utilities, *dump* returns a **1** upon normal completion.

When dumping the / directory it is not possible to unmount it. If the accounting are files changing, then the **a** option will cause them not be dumped.

BUGS

Sizes are based on 1600 bpi blocked tape; the raw magtape device has to be used to approach these densities. Fewer than 32 read errors on the filesystem are ignored (e.g., a warning message is issued). Greater than 32 read errors causes an abort. Each reel requires a new process, so parent processes for reels already written, just hang around until the entire tape is written.

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

If a different tape blocking factor is used, it must also be used with *restore*.

NAME

dumpfs – dump file system information

SYNOPSIS

dumpfs *filesys|device*

DESCRIPTION

Dumpfs prints out the super block and cylinder group information for the file system or special device specified. The listing is very long and detailed. This command is useful mostly for finding out certain file system information such as the file system block size and minimum free space percentage.

SEE ALSO

fs(5), disktab(5), newfs(8), fsck(8)

NAME

edactwho – edit /etc/actwho file

SYNOPSIS

edactwho

actwhocheck *actwhofile*

DESCRIPTION

Edactwho is a program to aid in editing the group-activities access control file */etc/actwho*. *Edactwho* cooperates with the *bill* command to maintain a valid */etc/actwho* file at all times. When *edactwho* is invoked, the user is taken to an editor to make changes in */etc/actwho*. The editor the user is taken to is determined by the environment variable *EDITOR*; if *EDITOR* is not set, the editor *vi* is used. After editing is complete, the new file is checked for syntax, then moved to */etc/actwho*.

Actwhocheck is the program which *edactwho* uses to validate the new */etc/actwho* database; *actwhocheck* checks the syntax of the file given as its argument.

FILES

/etc/actwho

SEE ALSO

bill(1), actwho(5),
"Accounting" chapter in the *Managing ConvexOS* doumentation set.

BUGS

Actwhocheck only checks syntax, not content.

NAME

edquota – edit user quotas

SYNOPSIS

/usr/etc/edquota [**–p** *proto-user*] *users...*
/usr/etc/edquota –t
/usr/etc/edquota [**–b** *bsoft*] [**–B** *bhard*] [**–i** *isoft*] [**–I** *ihard*] *filesys users ...*

DESCRIPTION

edquota is a quota editor. One or more users may be specified on the command line. If none of the batch mode flags are used (**b**, **B** , **i**, and/or **I**), then for each user a temporary file is created with an ASCII representation of the current disk quotas as well as the current disk usage for that user; an editor is then invoked on the file. The quotas may then be modified, new quotas added, etc. Upon leaving the editor, *edquota* reads the temporary file and modifies the binary quota files to reflect the changes made.

The editor invoked is *vi*(1) unless the EDITOR environment variable specifies otherwise.

Only the superuser may change or set quotas.

There is also a batch mode that will modify any or all of the four limits for a group of users for a single file system.

OPTIONS

–p duplicate the quotas of the prototypical user specified for each user specified. This is the normal mechanism used to initialize quotas for groups of users.

–t edit the soft time limits for each file system. If the time limits are zero, the default time limits in *<ufs/quota.h>* are used. Time units of sec(onds), min(utes), hour(s), day(s), week(s), and month(s) are understood. Time limits are printed in the greatest possible time unit such that the value is greater than or equal to one.

–b *blksoftlim*
Set the block soft limit to the given number. This flag indicates batch mode (an editor is not invoked) and requires the additional argument *filesys* before any of the users are listed.

–B *blkhardlim*
Set the block hard limit to the given number. This flag indicates batch mode and requires the additional argument *filesys* before any of the users are listed.

–i *inodesoftlim*
Set the inode soft limit to the given number. This flag indicates batch mode and requires the additional argument *filesys* before any of the users are listed.

–I *inodehardlim*
Set the inode hard limit to the given number. This flag indicates batch mode and requires the additional argument *filesys* before any of the users are listed.

FILES

quotas	quota file at the file system root
/etc/mtab	mounted file systems

SEE ALSO

quota(1), quotactl(2), quotacheck(8), quotaon(8), repquota(8)

BUGS

The format of the temporary file is inscrutable.

NAME

faillogon – log failed file access attempts

SYNOPSIS

/etc/faillogon [*/usr/adm/failure_log*]

DESCRIPTION

faillogon writes a log entry to */usr/adm/failure_log* each time a process attempts to access a file for which it does not have permission. The file */usr/adm/failure_log* must exist at the time you use ***faillogon***. If you invoke ***faillogon*** with no argument, failure logging is turned off.

Failed file access logging is automatically disabled when the file system the log file resides on becomes 98% full; it is enabled again when the file system becomes less than 96% full. The percentages may be changed using *sysgen*(8).

SEE ALSO

faillog(2), *faillogon*(8), *faillogpr*(8).
"System Protection" and "System Generation" chapters in the *Managing ConvexOS* doumentation set.

BUGS

In order to prevent possible file system deadlocks, failed attempts to access */usr/adm/failure_log* are logged to the system console and the SPU file */mnt/errlog* rather than to the specified log file.

NAME

faillogpr – format the failed file access log

SYNOPSIS

/usr/adm/faillogpr [*logfiles*]

DESCRIPTION

faillogpr is a program that formats file access failure log files as produced by *faillogon*(8) in human-readable format. *faillogpr* takes the names of one or more log files, as created by *faillogon*(8) and described in *failure_log*(5). If no arguments are specified, *faillogpr* reads its standard input.

faillogpr's output is made up of one line of text for each log entry. The format is designed to be easily processed by *awk*(1). Each line of text consists of each of the following fields separated by a single space character. There are no space characters in any of the fields except the *time* field.

time uid errno mode command filename

Each of the fields is described below.

time The time field shows the local time of the access. It has the format returned by *ctime*(3). The field has five subfields: day of week, month, day of month, time of day, and year. Day of week is a three letter abbreviation, Sun, Mon, Tue, etc. Month is a three letter abbreviation, Jan, Feb, Mar, etc. Day of month is one or two digits. Time of day is of the form *hh:mm:ss* where *hh* is hours since midnight, *mm* is minutes past the hour, and *ss* is seconds past the minute.

uid This field contains the user ID's of the accessing program. If the program's real and effective user ID's are the same, the ID is only printed once. If they are different, i.e., the program is running set uid, the real ID is printed, followed by the effective ID in parentheses. Each ID is looked up in */etc/passwd*. If a user is found who has that ID, the user's name is printed. Otherwise, the ID is printed as a decimal number.

errno This field contains the symbolic name of the error code that was returned from the system call that failed. The symbolic names are all listed in */usr/include/errno.h* and also are listed and explained in *intro*(2), as well as in the manual page for each system call.

mode The *mode* field is one or more of *R*, *W*, or *X*, indicating that the attempted access was for reading, writing, or execution, respectively. If the access was for none of those, it will be indicated as an underscore character, "_".

command

The name of the command is printed here. If the access is a Network File System (NFS) access from a remote machine, the command name will be printed as "nfsd". If the command name contains non-printing or blank characters, those characters will be printed as *\ooo*, where *ooo* is the character's ASCII value in octal.

filename

The absolute pathname of the accessed file is printed. As with the command name, non-printing and blank characters are printed as octal escapes.

Note that *faillogpr* looks at the current state of the file systems to determine the names of files. The log file generated by the operating system only contains the device number and I-node number of the file. If the file has been deleted between the time when the failing access occurs and the time when *faillogpr* is run, the name will not be reported correctly, and the file name will be printed as */aaa/bbb/ccc/<inumber> where /aaa/bbb/ccc* is the mount point of the file's file system and *inumber* is the file's I-node number. See *fs*(5) for an explanation of file systems, I-nodes and I-node numbers.

Also, *faillogpr* looks in */etc/fstab* to determine where each file system is mounted. If the

information in */etc/fstab* is out of date, *faillogpr* may report pathnames incorrectly. When *faillogpr* can't find where a file system is mounted in */etc/fstab*, the first part of the pathname is printed as *<major/minor>*.

For example, a typical output line would look like this.

```
Wed May  6 21:15:19 1987 smith EACCES RW touch /etc/rc
```

This would indicate that at 9:15 pm local time on Wednesday, May 6th, 1987, the program *touch*, running with *smith*'s user ID attempted to access the file */etc/rc* for both reading and writing and got the error return code *EACCES*.

NOTES

Because the file names generated by *faillogpr* are correct at the time when *faillogpr* is run, rather than the time when the failing access occurred, it is suggested that *faillogpr* should be run frequently so as to minimize the likelihood of incorrect file names being recorded. See the Log Files chapter of *Managing ConvexOS:Configuration Guide* for more information.

FILES

/etc/fstab /etc/ncheck /etc/passwd

SEE ALSO

awk(1), *intro*(2), *faillog*(2), *ctime*(3), *failure_log* R(5), *fs*(5), *fstab*(5), *faillogon*(8).
"Log Files" chapter in *Managing ConvexOS:Configuration Guide*.

BUGS

faillogpr reads its input twice. For that reason, it cannot take its input from a pipe or a socket.

NAME

fastboot, fasthalt – reboot/halt the system without checking the disks

SYNOPSIS

/etc/fastboot [*boot-options*]
/etc/fasthalt [*halt-options*]

DESCRIPTION

Fastboot and *fasthalt* are shell scripts which reboot and halt the system without checking the file systems. This is done by creating a file */fastboot*, then invoking the *reboot* program. The system startup script, */etc/rc*, looks for this file and, if present, skips the normal invocation of *fsck*(8).

SEE ALSO

halt(8), reboot(8), rc(8)

NAME

fingerd – remote user information server

SYNOPSIS

/usr/etc/in.fingerd

DESCRIPTION

fingerd is a simple protocol based on RFC742 that provides an interface to the *name* and *finger* programs at several network sites. The program is supposed to return a friendly, human-oriented status report on either the system at the moment or a particular person in depth. There is no required format and the protocol consists mostly of specifying a single command line.

fingerd listens for TCP requests at port 79. Once connected, it reads a single command line terminated by a <CRLF>, which is passed to *finger*(1). *fingerd* closes its connections as soon as the output is finished.

If the line is null (i.e., just a <CRLF> is sent), then *finger* returns a "default" report that lists all people logged into the system at that moment.

If a user name is specified (e.g., **eric**<CRLF>), then the response lists more extended information for only that particular user, whether logged in or not. Allowable names in the command line include both login names and user names. If a name is ambiguous, all possible derivations are returned.

SEE ALSO

finger(1)

BUGS

Connecting directly to the server from a *tip* command or an equally restrictive TELNET-protocol user program can result in meaningless attempts at option negotiation being sent to the server, which will interfere with command line interpretation. *fingerd* should be refined to filter out IAC's and perhaps even respond negatively (IAC WON'T) to all option commands received.

NAME

fsck – file system consistency check and interactive repair

SYNOPSIS

/etc/fsck **–p** [**–f**] [**–c**] [**–d**] [filesystem ...]
/etc/fsck [**–b** block#] [**–y**] [**–n**] [**–c**] [**–d**] [**–m**] [filesystem ...]

DESCRIPTION

The first form of *fsck* preens a standard set of filesystems or the specified file systems. It is normally used in the script **/etc/rc** during automatic reboot. In this case *fsck* reads the table **/etc/fstab** to determine which file systems to check. It uses the information there to inspect groups of disks in parallel taking maximum advantage of i/o overlap to check the file systems as quickly as possible. Normally, the root file system will be checked on pass 1, other "root" ("a" partition) file systems on pass 2, other small file systems on separate passes (e.g. the "d" file systems on pass 3 and the "e" file systems on pass 4), and finally the large user file systems on the last pass, e.g. pass 5. A pass number of 0 in fstab causes a disk to not be checked; similarly partitions which are not shown as to be mounted "rw" or "ro" are not checked.

The system takes care that only a restricted class of innocuous inconsistencies can happen unless hardware or software failures intervene. These are limited to the following:

Unreferenced inodes

Link counts in inodes too large

Missing blocks in the free map

Blocks in the free list also in files

Counts in the super-block wrong

These are the only inconsistencies which *fsck* with the **–p** option will correct; if it encounters other inconsistencies, it exits with an abnormal return status and an automatic reboot will then fail. For each corrected inconsistency one or more lines will be printed identifying the file system on which the correction will take place, and the nature of the correction. After successfully correcting a file system, *fsck* will print the number of files on that file system and the number of used and free frags.

If sent a QUIT signal, *fsck* will finish the file system checks, then exit with an abnormal return status that causes the automatic reboot to fail. This is useful when you wish to finish the file system checks, but do not want the machine to come up multiuser.

Without the **–p** option, *fsck* audits and interactively repairs inconsistent conditions for file systems. If the file system is inconsistent the operator is prompted for concurrence before each correction is attempted. It should be noted that a number of the corrective actions which are not fixable under the **–p** option will result in some loss of data. The amount and severity of data lost may be determined from the diagnostic output. The default action for each consistency correction is to wait for the operator to respond **yes** or **no**. If the operator does not have write permission *fsck* will default to a **–n** action.

Fsck has more consistency checks than its predecessors *check, dcheck, fcheck,* and *icheck* combined.

The following flags are interpreted by *fsck.*

–b Use the block specified immediately after the flag as the super block for the file system. Block 32 is usually an alternate super block.

–y Assume a yes response to all questions asked by *fsck;* this should be used with great caution as this is a free license to continue after essentially unlimited trouble has been encountered.

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

writing.

-d Certain diagnostic information is written as *fsck* goes about its work.

-f Force *fsck* to check filesystems whose dirty bit is not set. *Fsck* assumes this flag when -p flag IS NOT specified.

-c Inode fields unused prior to version 9.0 of ConvexOS are zeroed; this flag should be use when first upgrading to ConvexOS version 9.0. The following fields of the inode are cleared by this flag: *ic_spare[0..3], ic_dmonflags, ic_dmonperm,* and *ic_sysflags.*

-m The mode given to the lost+found directory if *fsck* needs to create it. The default is 01777.

If no filesystems are given to *fsck* then a default list of file systems is read from the file **/etc/fstab**.

Inconsistencies checked are as follows:

1. Blocks claimed by more than one inode or the free map.
2. Blocks claimed by an inode or the free map outside the range of the file system.
3. Incorrect link counts.
4. Size checks:
   Directory size not of proper format.
   Partially truncated file.
5. Bad inode format.
6. Blocks not accounted for anywhere.
7. Directory checks:
   File pointing to unallocated inode.
   Inode number out of range.
8. Super Block checks:

   More blocks for inodes than there are in the file system.
9. Bad free block mapformat.
10. Total free block and/or free inode count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with the operator's concurrence, reconnected by placing them in the **lost+found** directory. The name assigned is the inode number. If the *lost+found* directory does not exist, it is created.

Checking the raw device is almost always faster.

FILES

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

DIAGNOSTICS

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

SEE ALSO

fstab(5), fs(5), newfs(8), mkfs(8), reboot(8)

BUGS

Inode numbers for . and .. in each directory should be checked for validity.

There should be some way to start a **fsck -p** at pass *n*.

NAME

fsirand – install random inode generation numbers

SYNOPSIS

fsirand [**–p**] special

DESCRIPTION

fsirand installs random inode generation numbers on all the inodes on device *special*, and also installs a filesystem ID in the superblock. This helps increase the security of filesystems exported by NFS.

fsirand must be used only on an unmounted filesystem that has been checked with *fsck*(8). The only exception is that it can be used on the root filesystem in single-user mode, if the system is immediately re-booted afterwards.

OPTIONS

–p Print out the generation numbers for all the inodes, but do not change the generation numbers.

NAME

fstat – identify open files

SYNOPSIS

fstat [**–u** user] [**–p** pid] [**filename...**]

DESCRIPTION

Fstat identifies open files. A file is considered open if a process has it open or it is the working directory for a process. If no options are specified, *fstat* reports on all open files.

OPTIONS

–u requests a report of all files open by a specified user.

–p requests a report of all files open by a specified process id.

filename... restricts the reports to the specified files. If a file is a block special file, *fstat* additionally reports on any open files on that device, treating it as a mounted file system.

OUTPUT

The following fields are printed

USER is the user name of the owner of the process. If the user is not found in the password file, the id is printed out as its numeric equivalent.

CMD is the command name of the process.

PID is the process id.

FD is the file descriptor number in the per–process open file table. The special file descriptor **wd** signifies the working directory of the process.

If the file number is followed by an asterisk (*), then the file is not an inode, but either a socket, process descriptor (see *pattach*(2)), a SPUio descriptor, or has an error of some kind. The format of the rest of the entry is variable, doesn't correspond to the headings, and is enclosed in parenthesis. The **SOCKET** section describes the variable format for sockets.

DEVICE is a display of the major and minor numbers of the device where this file's inode resides.

A device number of the form "255, XXX" shows the item in question to be remote mounted via NFS. The minor number, "XXX", is the number of that NFS mount, as found in */etc/mtab* . For example, "255, 10" means that the file in question is located on the 10th (starting at 0) NFS remote mount found in */etc/mtab*.

INODE is the inode number of the file.

SIZE is the size in bytes of the file, where applicable.

TYPE is the type of the file, as found in */usr/include/sys/vnode.h*

FLAGS There are the process flags as found in */usr/include/sys/vnode.h* An **r** indicates that the file is the root of its file system. An **m** indicates that the file is a virtual memory object associated with the vnode.

NAME are the specified file names, if any.

SOCKET

The information displayed for an open socket depends on its protocol domain. The first three fields are always the same: field one is the domain name; field two is the socket type (stream, dgram, etc); and field three is the socket flags field in hex (see */usr/include/sys/socketvar.h*).

The remaining fields are protocol dependent. There is a final field for TCP sockets:

the address of the TCPCB; for UDP, the INPCB (socket PCB). For UNIX domain sockets, the remaining two fields display the address of the socket PCB and the address of the connected PCB. Otherwise, the protocol number and address of the socket itself are printed as the final field. (UNIX is a registered trademark on UNIX System Laboratories, Inc.)

The intent is to supplement *netstat*(1C). For example, the addresses mentioned above are the addresses that the "netstat -A" command would print for TCP, UDP, and UNIX domain.

Note that, since *pipe*(2) is implemented with sockets, a pipe appears as a connected UNIX domain stream socket. A unidirectional UNIX domain socket indicates the direction of flow with an arrow ("<-" or "->"), and a full duplex socket shows a double arrow ("<->").

EXAMPLES

This example shows the use of *fstat* to display information on the file */usr/spool/lpd/lock*.

```
% fstat /usr/spool/lpd/lock
USER   CMD    PID   FD      DEVICE      INODE SIZE   TYPE   FLAGS NAME
root   lpd    113   3       9,    35    26683 6      reg    x     /usr/spool/lpd/lock
```

This example shows the use of *fstat* to display all the files of a process that has */usr/spool/lpd/lock* open.

```
% fstat -p 113
USER   CMD    PID   FD      DEVICE      INODE SIZE   TYPE   FLAGS
root   lpd    113   wd      9,    35    26624 512    dir
root   lpd    113   0       9,    0     4103  0      chr
root   lpd    113   1       9,    0     4103  0      chr
root   lpd    113   2       9,    35    26634 37     reg
root   lpd    113   3       9,    35    26683 6      reg
root   lpd    113   4*      (inet stream 80 tcp 80f6770c)
```

This example shows the use of *netstat*(1C), *grep*(1) and *fstat* to locate the process that has a socket open on the "smtp" network port. The **-aA** arguments direct *netstat* to display all protocol control block (PCB) addresses, and the *fstat* command displays all files, filtering the output through *grep* to locate the specific PCB.

```
% netstat -aA | grep smtp
80f67a8c tcp 0 0 *.smtp *.* LISTEN
% fstat | grep 80f67a8c
root   sendmail     108   5*      (inet stream 80 tcp 80f67a8c)
```

Finally, *fstat* can be used to help determine why a given filesystem (either locally or remotely mounted) cannot be unmounted. In this example, the local filesystem in question cannot be unmounted because both the regular file used by user john's *vi* process, and the working directory for user bob's *ls* is the root directory of the filesystem:

```
% umount /dev/du3a
/dev/du3a: Mount device busy
% fstat /dev/du3a
USER   CMD    PID   FD      DEVICE      INODE SIZE   TYPE   FLAGS NAME
john   vi     544   5       64,   769   21628 133    reg           /dev/du3a
bob    ls     698   wd      64,   769     2   752    dir    r      /dev/du3a
```

Since NFS mounts have no associated special block devices, one must use a different method to ferret out activity in them. As mentioned above, files on NFS mounts are given 255 as their major device number, and their position relative to other NFS mounts in */etc/mtab* as their minor number. Once this minor number is known, one can look for all NFS files open on that minor

device number. In this example, john's *vi* has a file open in *host2:/mrel* .

```
% umount /rmt/host2/mrel
/rmt/host2/mrel: Mount device busy
% grep : /etc/mtab
host0:/mnt /rmt/host0/mnt nfs bg,intr,timeo=30 0 0
host1:/usr/spool/globdata /rmt/host1/globdata nfs bg,intr,timeo=30 0 0
host2:/mrel /rmt/host2/mrel nfs bg,intr,timeo=30 0 0
% fstat | grep '255, *2'
john   vi     3141  6      255,  2      822    9524  reg
```

Fstat is NOT useful in determining why a filesystem cannot be unmounted in the case of layered mounts. That is, *fstat* will not identify */usr/spool/mail* as the reason preventing */usr/spool* from being unmounted.

BUGS

Socket information clutters the output.

fstat cannot find executing binaries which reside in a particular filesystem, but have no files open nor its working directory in that filesystem.

fstat cannot find inodes that are in use by the kernel, such as the inodes in use for the current accounting file (normally /usr/adm/acct) and the current failure log file (normally /usr/adm/faillog.)

Files being used as virtual memory objects, but which have been closed, are not identified.

Since *fstat* takes a snapshot of the system, it is only correct for a very short period of time.

AUTHORS

Fstat comes from the 4.3BSD Tahoe distribution.

Vic Abell of the Purdue University Computing Center ported it to DYNIX 3.0.1[24] for the Sequent Balance and Symmetry machines, SunOS 4.0 and ULTRIX 2.2.

Tom Christiansen of CONVEX Computer Corporation ported it to the CONVEX, added the FLAGS field, the -m and -n options, the "process descriptor" and spuio types, decoded remote file descriptors, and improved the diagnostics on unknown user id's.

SEE ALSO

ps(1), netstat(1C), pattach(2), pstat(8).

NAME

ftpd – DARPA Internet File Transfer Protocol server

SYNOPSIS

/etc/ftpd [**–d**] [**–l**] [**–t** *timeout*] [**–T** *maxtimeout*]

DESCRIPTION

ftpd is the DARPA Internet File Transfer Protocol (FTP) server process. The server uses the TCP protocol and listens at the port specified in the "ftp" service specification; see *services*(5).

If the **–d** option is specified, debugging information is written to the *syslog*.

If the **–l** option is specified, each *ftp* session is logged in the *syslog*.

The *ftp* server will timeout an inactive session after 15 minutes. If the **–t** option is specified, the inactivity timeout period will be set to *timeout* seconds. A client may also request a different timeout period; the maximum period allowed may be set to *timeout* seconds with the **–T** option. The default limit is 2 hours.

The *ftp* server currently supports the following *ftp* requests; case is not distinguished.

Request	**Description**
ABOR	abort previous command
ACCT	specify account (ignored)
ALLO	allocate storage
APPE	append to a file
CDUP	change to parent of current working directory
CWD	change working directory
DELE	delete a file
HELP	give help information
LIST	give list files in a directory (**ls -lgA**)
MKD	make a directory
MDTM	show last modification time of file
MODE	specify data transfer *mode*
NLST	give name list of files in directory
NOOP	do nothing
PASS	specify password
PASV	prepare for server-to-server transfer
PORT	specify data connection port
PWD	print the current working directory
QUIT	terminate session
RETR	retrieve a file
RMD	remove a directory
RNFR	specify rename-from file name
RNTO	specify rename-to file name
SITE	non-standard commands (see next section)
SIZE	return size of file
STAT	return status of server
STOR	store a file
STOU	store a file with a unique name
STRU	specify data transfer structure
SYST	show operating system type of server system
TYPE	specify data transfer type
USER	specify user name
XCUP	change to parent of current working directory (deprecated)
XCWD	change working directory (deprecated)
XMKD	make a directory (deprecated)
XPWD	print the current working directory (deprecated)

XRMD remove a directory (deprecated)

The following nonstandard commands are supported by the SITE request:

Request	Description
UMASK	change umask, e.g., **SITE UMASK 002**
IDLE	set idle-timer, e.g., **SITE IDLE 60**
CHMOD	change mode of a file, e.g., **SITE CHMOD 755** *filename*
HELP	give help information, e.g., **SITE HELP**

The remaining *ftp* requests specified in Internet RFC 959 are recognized, but not implemented. MDTM and SIZE are not specified in RFC 959, but will appear in the next updated FTP RFC.

The *ftp* server will abort an active file transfer only when the ABOR command is preceded by a Telnet "Interrupt Process" (IP) signal and a Telnet "Synch" signal in the command Telnet stream, as described in Internet RFC 959. If a STAT command is received during a data transfer, preceded by a Telnet IP and Synch, transfer status will be returned.

ftpd interprets file names according to the "globbing" conventions used by *csh*(1). This allows users to utilize the metacharacters, "*?[]{}~."

ftpd authenticates users according to three rules:

1) The user name must be in the password data base, */etc/passwd*, and not have a null password. In this case, a password must be provided by the client before any file operations may be performed.

2) The user name must not appear in the file */etc/ftpusers*.

3) The user must have a standard shell returned by *getusershell*(3).

4) If the user name is "anonymous" or "ftp", an anonymous *ftp* account must be present in the password file (user "ftp"). In this case, the user is allowed to log in by specifying any password (by convention, this is given as the client host's name).

In the last case, *ftpd* takes special measures to restrict the client's access privileges. The server performs a *chroot*(2) command to the home directory of the *ftp* user. So that system security is not breached, it is recommended that the *ftp* subtree be constructed with care; the following rules are recommended:

~ftp) Make the home directory owned by "ftp" and not writable by anyone.

~ftp/bin)
Make this directory owned by the super-user and not writable by anyone. The program *ls*(1) must be present to support the *list* command. This program should have mode 111.

~ftp/etc)
Make this directory owned by the super-user and not writable by anyone. The files *passwd*(5) and *group*(5) must be present for the *ls* command to be able to produce owner names rather than numbers. The password field in *passwd* is not used, and should not contain real encrypted passwords. These files should be mode 444.

~ftp/pub)
Make this directory mode 777 and owned by "ftp." Users should then place files that are to be accessible via the anonymous account into this directory.

SEE ALSO

ftp(1), getusershell(3), syslogd(8)

BUGS

The anonymous account is inherently dangerous and should avoided when possible.

The server must run as the super-user to create sockets with privileged port numbers. It maintains an effective user id of the logged-in user, reverting to the super-user only when binding addresses to sockets. The possible security holes have been extensively scrutinized, but the list of known holes is possibly incomplete.

NAME

genrest – generate password restrictions file

SYNOPSIS

genrest [**–t**] [**–m***n*] [**–M***N*]

DESCRIPTION

Genrest creates the password restrictions file */etc/pwrestrict* from the password file */etc/passwd*. Each line in */etc/passwd* is scanned, and the requisite fields are extracted to make an entry for the restrictions file. Finally, *mkpasswd(8)* is used to build the database files */etc/pwrestrict.dir* and */etc/pwrestrict.pag*.

By default, no restriction are enabled when */etc/pwrestrict* is built. If the **–t** option is specified, type restrictions will be enabled for every user. If either the **–m** or **–M** option is given password aging will be enabled; the **–m** option sets the minimum period in weeks which must lapse before the password can be changed to *n*. The **–M** option sets the maximum number of weeks for which a password is valid to *N*. The default minimum is 1 week; the default maximum is 52 weeks.

To set user-specific password restrictions, edit */etc/pwrestrict* after *genrest* using the *vipw*(8) utility.

Genrest will not build the */etc/pwrestrict* file if it already exists.

The file */etc/pwrestrict* is *optional* and may be omitted; it is only needed by sites desiring password restrictions.

SEE ALSO

passwd(5), pwrestrict(5), mkpasswd(8), vipw(8)

NAME

getst – print information from the kernel's stripe tables

SYNOPSIS

getst *[stripedev ...]*
getst *-H [diskdev...]*

DESCRIPTION

getst optionally accepts comman line arguments in the form of; *st?*, */dev/st?*, or */dev/rst?* (where "?" is the stripe device number found in */dev*). In each case, the actual file referenced will be */dev/rst?*. If an argument exists, getst prints a brief summary of kernel stripe table information for that stripe to standard output. If getst is invoked by itself, kernel stripe table information for each stripe is printed to standard output.

Information retrieved from the kernel stripe table includes; partitions that make up the stripe, the number of blocks used on each partition, and the stripe blocking factor. Redundant stripe information will additionally include information as to the existence of a dead device on a section, reconstruction status and Hot Spare list status.

Switch options available:

–H Scan */etc/stripecap* Hot Spare entries rather than stripe entries. If no command argument is specified, data is returned on all Hot Spares. In this mode the diskdev argument may instead be referenced by hot spare entry name (i.e. getst -H hs0). Information returned is the amount of the Hot Spare partition used, where it's being used, and how much is left, along with stripe affinities. If no *diskdev* argument is given, then information on all Hot Spares will be returned.

The user must have root privilege to execute this utility.

See *st*(4) or *newst*(8) for more information on stripes.

FILES

/etc/stripecap Database of the stripe descriptors.

SEE ALSO

st(4), stripecap(5), newst(8), putst(8), mvst(8), rmst(8), qst(8), convst(8)

NAME

gettable – get NIC format host tables

SYNOPSIS

/etc/gettable [**–v**] *host* [*outfile*]

DESCRIPTION

gettable is a simple program used to obtain the NIC standard host tables from a "nicname" server. The indicated *host* is queried for the tables. The tables, if retrieved, are placed in the file, *outfile*, or by default, *hosts.txt*.

The **–v** option will get just the version number instead of the complete host table and put the output in the file, *outfile*, or by default, *hosts.ver*.

gettable operates by opening a TCP connection to the port indicated in the service specification for "nicname." A request is then made for "ALL" names and the resultant information is placed in the output file.

gettable is best used in conjunction with the *htable*(8C) program, which converts the NIC standard file format to that used by the network library lookup routines.

SEE ALSO

intro(3), htable(8C), named(8)

BUGS

If the name server system provided network name mapping well as host name mapping, *gettable* would no longer be needed.

NAME

getty – set terminal mode

SYNOPSIS

/etc/getty [type]

DESCRIPTION

Getty is invoked by *init*(8) immediately after a terminal is opened, following the making of a connection. While reading the name *getty* attempts to adapt the system to the speed and type of terminal being used.

Init calls *getty* with an argument specified by the *ttys* file entry for the terminal line. The argument can be used to make *getty* treat the line specially. This argument is used as an index into the *gettytab*(5) database, to determine the characteristics of the line. If there is no argument, or there is no such table, the **default** table is used. If there is no */etc/gettytab* a set of system defaults is used. If indicated by the table located, *getty* will clear the terminal screen, print a banner heading, and prompt for a login name. Usually either the banner or the login prompt will include the system hostname. Then the user's name is read, a character at a time. If a null character is received, it is assumed to be the result of the user pushing the 'break' ('interrupt') key. The speed is usually then changed and the 'login:' is typed again; a second 'break' changes the speed again and the 'login:' is typed once more. Successive 'break' characters cycle through some standard set of speeds.

The user's name is terminated by a newline or carriage-return character. The latter results in the system being set to treat carriage returns appropriately (see *tty*(4)).

The user's name is scanned to see if it contains any lowercase alphabetic characters; if not, and if the name is nonempty, the system is told to map any future uppercase characters into the corresponding lowercase characters.

Finally, login is called with the user's name as argument.

Most of the default actions of *getty* can be circumvented, or modified, by a suitable *gettytab* table.

Getty can be set to timeout after some interval, which will cause dial-up lines to hang up if the login name is not entered reasonably quickly.

FILES

/etc/gettytab

SEE ALSO

gettytab(5), init(8), login(1), ioctl(2), tty(4), ttys(5).

NAME

halt – stop the processor

SYNOPSIS

/etc/halt [**–n**] [**–q**] [**–y**]

DESCRIPTION

Halt writes out sandbagged information to the disks and then stops the processor. The machine does not reboot, even if the auto-reboot switch is set on the console.

The **–n** option prevents the sync before stopping. The **–q** option causes a quick halt, no graceful shutdown is attempted. The **–y** option is needed if you are trying to halt the system from a dialup.

SEE ALSO

reboot(8), shutdown(8)

NAME

htable – convert NIC standard format host tables

SYNOPSIS

/etc/htable [**–c** *connected-nets*] [**–l** *local-nets*] *file*

DESCRIPTION

htable is used to convert host files in the format specified in Internet RFC 810 to the format used by the network library routines. Three files are created as a result of running *htable*: *hosts*, *networks*, and *gateways*. The *hosts* file may be used by the *gethostbyname*(3N) routines in mapping host names to addresses if the nameserver, *named*(8), is not used. The *networks* file is used by the *getnetent*(3N) routines in mapping network names to numbers. The *gateways* file may be used by the routing daemon in identifying "passive" Internet gateways; see *routed*(8C) for an explanation.

If any of the files, *localhosts*, *localnetworks*, or *localgateways*, is present in the current directory, the file's contents is prepended to the output file. Of these, only the *gateways* file is interpreted. This allows sites to maintain local aliases and entries which are not normally present in the master database. Only one gateway to each network will be placed in the *gateways* file; a gateway listed in the *localgateways* file will override any in the input file.

If the *gateways* file is to be used, a list of networks to which the host is directly connected is specified with the **–c** flag. The networks, separated by commas, may be given by name or in dot notation, e.g., **-c arpanet,128.32,local-ether-net**. *htable* includes only gateways that are directly connected to one of the networks specified, or that can be reached from another gateway on a connected network.

If the **–l** option is given with a list of networks (in the same format as for **–c**), these networks will be treated as "local," and information about hosts on local networks is taken only from the *localhosts* file. Entries for local hosts from the main database will be omitted. This allows the *localhosts* file to completely override any entries in the input file.

htable is best used in conjunction with the *gettable*(8C) program, which retrieves the NIC database from a host.

SEE ALSO

intro(4n), gettable(8C), named(8)

BUGS

If the name server system provided network name mapping well as host name mapping, *htable* would no longer be needed.

NAME

hyroute – set the HYPERchannel routing tables

SYNOPSIS

hyroute *interface* [**–s**] [**–p**] [**–c**] [*file*]
hyroute *interface* [**–d**]

DESCRIPTION

hyroute manipulates the HYPERchannel routing information.

The *interface* parameter specifies the HYPERchannel interface name as a string of the form "name unit," e.g., "hy0." With the **–s** option, it reads *file* and sets the system's database according to the information in the file (see below). If no input file is given, or if the argument "–" is encountered, *hyroute* reads from standard input.

The **–c** option causes *hyroute* to compare the system's current information to that contained in *file*.

The **–p** option causes a digested version of *file* to be printed.

The **–d** option causes a "dump" of the system's table (used for debugging routing code).

FILE FORMAT

The input file is free format. Comment lines start with an asterisk in column one. Statements end with a semicolon. The command:

direct *host dest control access MTU;*

describes a host that can be reached directly from this adapter. *host* is a host name as listed in */etc/hosts*; *dest*, *control*, *access*, and *MTU* are hexadecimal numbers. Data written to a given host will be sent to HYPERchannel address *dest* using a control value of *control*, an access code of *access* (see adapter manuals for details), and fragmented such that the maximum transmission unit is (*MTU* times 1024) plus 54 bytes of user data available in a HYPERchannel message proper. Valid MTU size inputs range from 1 Kbyte to 64 Kbytes (1 to 40 hex). The default MTU is 4 Kbytes.

The command:

gateway *host gate1 gate2 gate3 ... ;*

describes a host that must be reached indirectly through any one of the designated hosts. Listed hosts are not gateways in the formal sense (they do not run the internet gateway protocols), but are hosts on the HYPERchannel that can "bridge" between subsections of the HYPERchannel network.

A sample file follows:

```
* comment
direct azure      6100 0 0 4;
direct bronze     6101 0 0 4;
direct cyber      2100 1100 0 16;
direct dadcad     6102 0 0 4;
direct tekcad     2400 1100 0 16;
direct tekcrd     2201 1100 0 16;
direct tekid      2500 1100 0 16;
direct teklabs    2200 1100 0 16;
gateway iddic     tekcrd teklabs cyber tekcad tekid;
gateway iddme     tekcrd teklabs cyber tekcad tekid;
gateway metals    tekcrd teklabs cyber;
```

SEE ALSO

hy(4), hystat(8c)

NAME

hystat – acquire and display HYPERchannel adapter statistics

SYNOPSIS

hystat *interface* [**-d**] [**-t** *interval*] [**-r** *repeat*]

DESCRIPTION

hystat requests a given HYPERchannel *interface* to return its statistics and status.

The following describes flags and options:

If **-r** is given, the request and display are repeated the number of times given in the *repeat* count.

If **-t** is given, the interval between repeats is the number of seconds given by *interval*.

If **-d** is given, the display will be of the deltas of counters returned between repeats in a given execution of *hystat*. This display will scroll to *stdout*. Otherwise, information is displayed cumulatively on a static screen template.

Information given on the scrolled display is per trunk; for each of the four adapter trunks, accumulators of the message count and the retransmit count are displayed. In addition, an abort count and cancel count global to the adapter are displayed.

The static display shows these values as well as the adapter type, status byte, last function, trunk response, trunk status, received response, error code, and first message index.

SEE ALSO

netstat(1C)

BUGS

Trunk statistics are valid for the NSC A400 adapter. They may not be visible to the NSC NB400 adapter.

NAME

ibmdaemon – IBM-labelled-tape processing filter/daemon

SYNOPSIS

/usr/lib/tape/ibmdaemon -l *VSN login access physdev*
/usr/lib/tape/ibmdaemon -u *login physdev*
/usr/lib/tape/ibmdaemon -d *RPCprognum physdev reqhost symlink*

DESCRIPTION

ibmdaemon is a program that is only to be invoked by the master daemon of the tape subsystem (*tpdaemon*(8)). This program is almost exclusively an RPC server which will perform I/O requests for IBM-labelled tapes. It handles all processing of the IBM magnetic tape labels, the blocking and unblocking of logical records, as well as the translation of *ioctl* commands into a similar functionality for IBM tapes (where possible). With IBM-labelled tapes, attempts to read beyond the last file in the fileset yields the error ENOENT "No such file or directory".

ibmdaemon will create and maintain tape files to the specification given in IBM's manual "MVS/XA Magnetic Tape Labels and File Structure Administration".

The goal of *ibmdaemon* is to accept normal ConvexOS I/O and format/organize according to the IBM Standard Label Format.

The organization of files may be defined by the user with the tpattr(1) command. *ibmdaemon* will use the values defined by the user to format the data.

ibmdaemon does not fully implement the IBM Standard Label format. Although the daemon accepts a record format of variable blocked spanned records (VBS), the daemon will not support blocked spanned record formats. The daemon will put the correct attributes into the header labels and will treat the record format as undefined. It is up to the application to correctly implement the VBS block structure on the tape.

With the fixed record format, the blocksize defined for the file must be a multiple of the recordsize. If an error is made, the open(2) system call will fail. Each logical record is padded to the defined record length with spaces. If a record exceeds the length of the defined record length, it is truncated. The maximum record size is 64K bytes. IBM's documentation specifies the largest allowable record is 32760 bytes. *ibmdaemon* will enforce the 64K byte limit. If a tape is created with a recordsize greater than 32760 bytes, the tape may not be transportable to all IBM systems.

With the variable record format, the blocksize is the largest record possible plus four bytes. The variable length record will have a four byte integer prepended to the record. The four byte integer will be binary encoded. The maximum variable length record is 64K less 4. IBM's documentation specifies the largest allowable record is 32760 bytes. *ibmdaemon* will enforce the 64K limit. If you create a tape with a recordsize greater than 32760 bytes, the tape may not be transportable to all IBM systems.

An undefined record format will allow the application to treat the tape device as a raw tape device. SYSCALL is the only record delimiter enforced for the undefined record format. Each read(2) and write(2) system call will translate into one physical tape block. The tape system will not preprocess the data. The maximum blocksize for the undefined record format is 128K.

The logical record delimiter is defined using the tpattr(2) command. See the tpattr(2) man page for a definition of how this affects the reading and writing of a labelled tape.

The label daemon will enforce the expiration date of files on tape. If a file is about to be over written and is not expired, the open(2) will fail with errno set to ENOACCESS. The tape must be mounted in bypass mode to override this action. Any file following the file about to be written

over is considered to have the same expiration date. A file is expired if its expiration date is less than or equal to the current date.

The label (**-l**) and unlabel (**-u**) options are designed to operate only on one physical tape volume, but the driver (**-d**) mode processes I/O requests for an entire IBM tapeset (possibly more than one physical tape).

Options:

-l Labels an unlabelled tape. The first three arguments are used in creating the initial VOL1 label for the tape.

VSN is a string (up to six characters) to be used as the unique volume identifier, or volume serial number.

login is the login name of the user who invoked *tplabel*(1). It is converted into uppercase and used as the owner of the newly initialized IBM volume.

access This is ignored by the IBM daemon. Its place is held to maintain argument compatibility with ansidaemon(8).

physdev
specifies the physical path of the drive where the tape has been mounted.

-u Unlabels an IBM-labelled tape. Only the owner of a tape or the superuser is allowed to execute the *tpunlabel*(1) command, which results in the *ibmdaemon* being invoked with this option.

login is the login name of the user running *tpunlabel*(1). It is converted to uppercase before being compared with the owner that has been stored in the tape's VOL1 label when verifying authorization to unlabel the mounted tape.

physdev
specifies the physical tape drive that contains the tape being unlabelled.

-d Invokes the daemon in driver mode. The daemon is automatically started by *tpdaemon*(8) once an IBM-labelled tape has been mounted. There is one *ibmdaemon* running per active (mounted) IBM tapeset. The *ibmdaemon* program then serves as a pseudo device driver which will accept *open, close, read, write,* and *ioctl* calls from the kernel and translate each request into its IBM equivalent. Labels are transparently written on output, and verified on input. Existing programs such as *cp*(1) and *cat*(1) are able to write IBM-labelled tapes "transparently" using this interface. The required arguments are listed below:

RPCprognum
is the RPC(3) program number of this invocation of the *ibmdaemon.*

physdev
this is the path to the physical tape device where the first tape in the IBM tapeset is mounted.

reqhost is the hostname of the machine originating the request to use the device.

symlink is the symbolic link name associated with this IBM tapeset (given by the user in the *tpmount*(1) command).

IBM and MVS/XA are trademarks of International Business Machines.

SEE ALSO

tpmount(1), tpunmount(1), tpattr(1), tplabel(1), tpunlabel(1), tpqueue(1), tpwait(1), tape(3), tpconfig(8), tpdaemon(8), mt(1), ansidaemon(8)

MVS/XA Magnetic Tape Labels and File Structure Administration. IBM Document number GC26-4145-3.

NAME

icheck – file system storage consistency check

SYNOPSIS

/etc/icheck [**–s**] [**–b** numbers] [filesystem]

DESCRIPTION

N.B.: *Icheck* is obsoleted for normal consistency checking by *fsck*(8).

Icheck examines a file system, builds a bit map of used blocks, and compares this bit map against the free list maintained on the file system. If the file system is not specified, a set of default file systems is checked. The normal output of *icheck* includes a report of

The total number of files and the numbers of regular, directory, block special and character special files.

The total number of blocks in use and the numbers of single-, double-, and triple-indirect blocks and directory blocks.

The number of free blocks.

The number of blocks missing; i.e. not in any file nor in the free list.

The **–s** option causes *icheck* to ignore the actual free list and reconstruct a new one by rewriting the super-block of the file system. The file system should be dismounted while this is done; if this is not possible (for example if the root file system has to be salvaged) care should be taken that the system is quiescent and that it is rebooted immediately afterwards so that the old, bad in-core copy of the super-block will not continue to be used. Notice also that the words in the super-block which indicate the size of the free list and of the i-list are believed. If the super-block has been curdled these words will have to be patched. The **–s** option causes the normal output reports to be suppressed.

Following the **–b** option is a list of block numbers; whenever any of the named blocks turns up in a file, a diagnostic is produced.

Icheck is faster if the raw version of the special file is used, since it reads the i-list many blocks at a time.

FILES

Default file systems vary with installation.

SEE ALSO

fsck(8), dcheck(8), ncheck(8), fs(5), clri(8)

DIAGNOSTICS

For duplicate blocks and bad blocks (which lie outside the file system) *icheck* announces the difficulty, the i-number, and the kind of block involved. If a read error is encountered, the block number of the bad block is printed and *icheck* considers it to contain 0. 'Bad freeblock' means that a block number outside the available space was encountered in the free list. '*n* dups in free' means that *n* blocks were found in the free list which duplicate blocks either in some file or in the earlier part of the free list.

BUGS

Since *icheck* is inherently two-pass in nature, extraneous diagnostics may be produced if applied to active file systems.

It believes even preposterous super-blocks and consequently can get core images.

The system should be fixed so that the reboot after fixing the root file system is not necessary.

NAME

ifconfig – configure network interface parameters

SYNOPSIS

/etc/ifconfig *interface* [*address* [*dest_address*]] [*parameters*]
/etc/ifconfig *interface* [*protocol_family*]

DESCRIPTION

ifconfig is used to assign an address to a network interface and/or configure network interface parameters. *ifconfig* must be used at boot time to define the network address of each interface present on a machine; it may also be used at a later time to redefine an interface's address or other operating parameters. The *interface* parameter is a string of the form "name unit", e.g., **ex0.** The address is either a host name present in the host name data base, *hosts*(5), or a DARPA Internet address expressed in the Internet standard "dot notation."

The following parameters may be set with *ifconfig*:

up Mark an interface "up." This may be used to enable an interface after an "ifconfig down." It happens automatically when setting the first address on an interface. If the interface was reset when previously marked down, the hardware will be re-initialized.

down Mark an interface "down." When an interface is marked "down," the system will not attempt to transmit messages through that interface. If possible, the interface will be reset to disable reception as well. This action does not automatically disable routes using the interface.

trailers Request the use of a "trailer" link level encapsulation when sending (default). If a network interface supports *trailers*, the system will, when possible, encapsulate outgoing messages in a manner that minimizes the number of memory-to-memory copy operations performed by the receiver. On networks that support the Address Resolution Protocol (see *arp*(4P); currently, only 10 Mb/s Ethernet), this flag indicates that the system should request that other systems use trailers when sending to this host. Similarly, trailer encapsulations will be sent to other hosts that have made such requests. Currently used by Internet protocols only.

–trailers Disable the use of a "trailer" link level encapsulation.

arp Enable the use of the Address Resolution Protocol in mapping between network level addresses and link level addresses (default). This is currently implemented for mapping between DARPA Internet addresses and 10Mb/s Ethernet addresses.

–arp Disable the use of the Address Resolution Protocol.

metric *n* Set the routing metric of the interface to *n*, default 0. The routing metric is used by the routing protocol (*routed*(8c)). Higher metrics have the effect of making a route less favorable; metrics are counted as addition hops to the destination network or host.

debug Enable driver dependent debugging code; usually, this turns on extra console error logging.

–debug Disable driver dependent debugging code.

netmask *mask* (Inet only) Specify how much of the address to reserve for subdividing networks into subnetworks. The mask includes the network part of the local address and the subnet part, which is taken from the host field of the address. The mask can be specified as a single hexadecimal number with a leading 0x, with a dot-notation Internet address, or with a pseudo-network name listed in the network table, *networks*(5). The mask contains 1's for the bit positions in the 32-bit

address that are to be used for the network and subnet parts, and 0's for the host part. The mask should contain at least the standard network portion, and the subnet field should be contiguous with the network portion.

dest_address Specify the address of the correspondent on the other end of a point-to-point link.

broadcast (Inet only) Specify the address to use to represent broadcasts to the network. The default broadcast address is an address with a host part of all 1's.

ifconfig displays the current configuration for a network interface when no optional parameters are supplied. If a protocol family is specified, *ifconfig* will report only the details specific to that protocol family.

Only the super-user may modify the configuration of a network interface.

DIAGNOSTICS

Messages indicating the specified interface does not exit, the requested address is unknown, or the user is not privileged and tried to alter an interface's configuration.

SEE ALSO

netstat(1C), intro(4n), rc(8), ex(4)

NAME

inetd, inetd.conf – internet super-server daemon and configuration file

SYNOPSIS

/etc/inetd [**–d**] [configuration_file]

DESCRIPTION

inetd is normally run at boot time by the */etc/rc* startup file. It then listens for connections on certain internet sockets. When a connection is found on one of its sockets, it decides what service the socket corresponds to, and invokes a program to service the request. This process is passed the connection as file descriptor 0 and an argument of the form "sourcehost.sourceport" where "sourcehost" is hex and "sourceport" is decimal, as well as any arguments specified in the configuration file. After the program is finished, it continues to listen on the socket (except in some cases, as described below). Essentially, *inetd* allows you to run one daemon to invoke several others, thus reducing load on the system.

Upon execution, *inetd* reads its configuration information from a file that, by default, is */etc/inetd.conf*. There must be an entry for each field of the configuration file; fields are separated by tabs or spaces. Comments are denoted by a "#" at the beginning of a line. The fields of the configuration file are as follows:

service name
socket type
protocol
wait/nowait
user
server program
server program arguments

service name is the name of a valid service in the file */etc/services* or */etc/rpc*, if the entry is for an RPC program (i.e., a version number is given. See *server program*, below). For "internal" services (discussed below), the service name must be the official name of the service (that is, the first entry in */etc/services*).

socket type is one of "stream," "dgram," or "raw," depending on whether the socket handles stream, datagram, or raw data. Most RPC programs use the "dgram" socket type.

protocol must be a valid protocol as listed in */etc/protocols*. Examples might be "tcp" or "udp." In general, if the *socket type* is specified as "stream," then this entry is "tcp." Most RPC programs use the "udp" protocol.

wait/nowait is applicable only to datagram sockets; other sockets should have a "nowait" entry in this position. If a datagram server connects to its peer, freeing the socket so *inetd* can receive further messages on the socket, it is said to be a "multi-threaded" server, and should use the "nowait" entry. For datagram servers that process all incoming datagrams on a socket and eventually time out, the server is said to be "single-threaded," and should use a "wait" entry. *comsat*(8C), *biff*(1), and *talk*(8C) are all examples of the latter type of datagram server. An exception is *tftpd*; it is a datagram server that establishes pseudo-connections. It must be specified as "wait" in order to avoid a race condition; the server reads the first packet, creates a new socket, and then forks and exits to allow *inetd* to check for new service requests to spawn new servers. Most RPC programs use the "wait" entry.

user contains the username of the user as whom the server should run. This allows you to give servers less permission than root.

server program contains the pathname of the program *inetd* should execute when a request is found on its socket. If this is an RPC (Remote Procedure Call) daemon, this entry first specifies the version number(s) supported by the program, followed by a space, then the pathname of the program to execute. The version number may be a single digit or a range specified as two digits separated by "–," as in "1-4." If *inetd* provides this service internally, this entry should be

"internal."

server program arguments are specified just as you normally specify program arguments, starting with argv[0], which is the name of the program. If the service is provided internally, this entry is not used and need not be specified. Up to 8 additional arguments may be specified; they are passed to the program normally.

inetd provides several "trivial" services through the use of internal routines. These services are "echo," "discard," "chargen" (character generator), "daytime" (human-readable time), and "time" (machine-readable time represented as the number of seconds since midnight, January 1, 1900). All of these services are TCP based. For details on these services, consult the appropriate RFC from the Network Information Center.

inetd rereads its configuration file when it receives a hangup signal, *SIGHUP*. Services may be added, deleted, or modified when the configuration file is reread.

RPC SUPPORT

An RPC server can be started from *inetd*. The only differences from the usual code are that *svcudp_create* is called as:

transp = scvudp_create(0)

because *inetd* passes a socket file as descriptor 0; and *svc_register* is called as:

svc_register(PROGNUM, VERSNUM, service, transp, 0)

with the final flag as 0, because the program will already have been registered by *inetd*. If you want to exit from the server process and return control to *inetd*, you must explicitly exit, because *scv_run* never returns.

OPTIONS

–d Specifies that debugging traces are to be turned on for connection-oriented (TCP) services.

configuration_file
Specifies an alternate configuration file to use instead of */etc/inetd.conf*.

EXAMPLE

The following is an example */etc/inetd.conf* configuration file:

```
ftp         stream tcp nowait root /usr/etc/in.ftpd          ftpd -l/tmp/ftp.log
telnet      stream tcp nowait root /usr/etc/in.telnetd       telnetd
shell       stream tcp nowait root /etc/in.rshd              rshd
login       stream tcp nowait root /etc/in.rlogind           rlogind -d
exec        stream tcp nowait root /usr/etc/in.rexecd        rexecd
syslog      dgram  udp wait   root /usr/etc/in.syslog        syslog -t5
comsat      dgram  udp wait   root /usr/etc/in.comsat        comsat
talk        dgram  udp wait   root /usr/etc/in.talkd         talkd
crashdump dgram  udp wait   root /usr/etc/in.crashreceive crashreceive
rusers      dgram  udp wait   root 1-2 /usr/etc/rpc.rusersd rusersd
rup         dgram  udp wait   root 1-3 /usr/etc/rpc.rstatd  rstatd
rwall       dgram  udp wait   root 1 /usr/etc/rpc.rwalld    rwalld
mount       dgram  udp wait   root 1 /usr/etc/rpc.mountd    mountd
quota       dgram  udp wait   root 1 /usr/etc/rpc.rquotad   rquotad
spray       dgram  udp wait   root 1 /usr/etc/rpc.sprayd    sprayd
rex         stream tcp wait   root 1 /usr/etc/rpc.rexd      rexd
echo        stream tcp nowait root internal
discard     stream tcp nowait root internal
chargen     stream tcp nowait root internal
daytime     stream tcp nowait root internal
time        stream tcp nowait root internal
```

```
echo        dgram  udp wait    root  internal
discard     dgram  udp wait    root  internal
chargen     dgram  udp wait    root  internal
daytime     dgram  udp wait    root  internal
time        dgram  udp wait    root  internal
```

FILES

/etc/inetd.conf list of internet server processes

SEE ALSO

ftpd(8C), *rexecd*(8C), *rlogind*(8C), *rshd*(8C), *telnetd*(8C), *comsat*(8C), *crashdump*(8), *syslog*(8), *talkd*(8), *mountd*(8C), *rexd*(8C), *rquotad*(8C), *rstatd*(8C), *rusersd*(8C), *rwalld*(8C), *sprayd*(8C)

NOTES

To selectively invoke TCP debugging packet tracing on a per-service basis, consult the manual page of the server program to determine if it accepts an option to turn on debugging.

NAME

init – process control initialization

SYNOPSIS

/etc/init

DESCRIPTION

init is invoked inside ConvexOS as the last step in the boot procedure. *init* causes */etc/.initrc* to be executed unconditionally, if it exists and is executable. This facility is intended for operations (such as loading the stripe table with *putst(8)*) which must be completed before */etc/rc* is run. In general, site-specific commands should *not* be placed in */etc/.initrc*. Where possible, general startup tasks should be performed in */etc/rc* and */etc/rc.local* instead.

init normally then runs the automatic reboot sequence as described in *reboot*(8), and if this succeeds, begins multi-user operation. If the reboot fails, it commences single user operation by giving the super-user a shell on the console. It is possible to pass parameters from the boot program to *init* so that single user operation is commenced immediately. When such single user operation is terminated by killing the single-user shell (i.e. by hitting ^D), *init* runs */etc/rc* without the reboot parameter. This command file performs housekeeping operations such as removing temporary files, mounting file systems, and starting daemons.

In multi-user operation, *init's* role is to create a process for each terminal port on which a user may log in. To begin such operations, it reads the file */etc/ttys* and forks several times to create a process for each terminal specified in the file. Each of these processes opens the appropriate terminal for reading and writing. These channels thus receive file descriptors 0, 1 and 2, the standard input and output and the diagnostic output. Opening the terminal will usually involve a delay, since the *open* is not completed until someone is dialed up and carrier established on the channel. If a terminal exists but an error occurs when trying to open the terminal *init* complains by writing a message using the *syslog(3)* facility. the message is repeated every 10 minutes for each such terminal until the terminal is shut off in /etc/ttys and init notified (by a hangup, as described below), or the terminal becomes accessible (init checks again every minute). After an open succeeds, */etc/getty* is called with argument as specified by */etc/ttys* file. *getty* reads the user's name and invokes *login* to log in the user and execute the shell.

Ultimately the shell will terminate because of an end-of-file either typed explicitly or generated as a result of hanging up. The main path of *init*, which has been waiting for such an event, wakes up and removes the appropriate entry from the file *utmp*, which records current users, and makes an entry in */usr/adm/wtmp*, which maintains a history of logins and logouts. The *wtmp* entry is made only if a user logged in successfully on the line. Then the appropriate terminal is reopened and *getty* is reinvoked.

init catches the *hangup* signal (signal SIGHUP) and interprets it to mean that the file */etc/ttys* should be read again. The shell process on each line which used to be active in *ttys* but is no longer there is terminated; a new process is created for each added line; lines unchanged in the file are undisturbed. Thus it is possible to drop or add phone lines without rebooting the system by changing the *ttys* file and sending a *hangup* signal to the *init* process: use 'kill –HUP 1.'

init will terminate multi-user operations and resume single-user mode if sent a terminate (TERM) signal, i.e. "kill –TERM 1". If there are processes outstanding which are deadlocked (due to hardware or software failure), *init* will not wait for them all to die (which might take forever), but will time out after 30 seconds and print a warning message.

init will cease creating new *getty*'s and allow the system to slowly die away, if it is sent a terminal stop (TSTP) signal, i.e. "kill –TSTP 1". A later hangup will resume full multi-user operations, or a terminate will initiate a single user shell. This hook is used by *reboot*(8) and *halt*(8).

init's role is so critical that if it dies, the system will reboot itself automatically. If, at bootstrap time, the *init* process cannot be located, the system will loop during initialization.

DIAGNOSTICS

init: *tty*: cannot open. A terminal which is turned on in the *rc* file cannot be opened, likely because the requisite lines are either not configured into the system or the associated device was not attached during boot-time system configuration.

init: *tty*: getty failing, sleeping. Numerous attempts to start *getty* on the named terminal have failed. Attempts will resume after a 30 second wait. Could be inidicative of a malfunctioning port or terminal.

WARNING: Something is hung (won't die); ps axl advised. A process is hung and could not be killed when the system was shutting down. This is usually caused by a process which is stuck in a device driver due to a persistent device error condition.

FILES

/dev/console, /dev/tty*, /etc/utmp, /usr/adm/wtmp, /etc/ttys, /etc/rc /etc/.initrc

SEE ALSO

login(1), kill(1), sh(1), syslog(3), ttys(5), getty(8), rc(8), reboot(8), halt(8), shutdown(8)

NAME

installsw – install or create CONVEX software release tapes

SYNOPSIS

/etc/installsw [**–r**] [**–i**] [**–f** *in_file*] [**–d** *tape_name*] [**–F**]

DESCRIPTION

installsw is used to create and install CONVEX software release tapes.

A release tape contains a header file with the CONVEX copyright notice, and other information describing the software release; a script file to perform software installation; and one or more data files to be installed by the installation script. The installation process involves verification of the header file, followed by loading and execution of the installation script file. The script file controls the remainder of the installation process.

To create an install tape, you must specify text for the header and install script, and specify a build script which is executed to control creation of the release tape data. The build script is not written to the tape.

The default creation/installation device for CONVEX software is */dev/rmt12*. This can be changed by the *device* command shown below, or the **–d** option on the command line. It can be changed to the SPU cartridge tape by using the device name *ct*. When running on the SPU, the creation/installation device is the SPU cartridge tape only.

The **–r**, **–i**, and **–f** options are mutually exclusive.

If the **–r** option is used, the *verify* command (see below) is executed noninteractively to verify the tape header. The default device is used unless specified with **–d** option.

If the **–i** option is used, the *install* command (see below) is automatically executed noninteractively. The default device is used unless specified with **–d** option.

If the **–f** option is used, *installsw* takes all further command input from the file *in_file*. The input must contain valid commands from the list given below. All commands are executed noninteractively. The default device is used unless specified with the **–d** option. Of course, the command file can also respecify the tape device.

installsw normally checks the tape device to make sure it really is a tape device before using it. If the **–F** option is used, this check is not made. That is, *installsw* is "forced" to use the tape device regardless of what it is.

If *installsw* is invoked without the **–r**, **–i**, or **–f** commands, it operates in interactive mode. The initial tape device setting is the default device unless changed with the **–d** option on the command line. The user can also specify the tape device at any time with the *device* command.

When in interactive mode, the user is prompted for commands. Command execution may involve considerably more user interaction. This is true even for command files executed from interactive mode with the *auto* command.

The following commands may be input directly to the interactive prompt, or used in command files invoked with the *auto* command, or the **–f** option on the command line.

d[evice] *dev_name*

This command specifies the device to be used by *install* or *create*. If the device is not set with this command, the default specified in the opening paragraphs applies, unless changed by the **–d** option on the *installsw* command line. If *ct* is specified from the CONVEX computer, the SPU cartridge tape will be accessed.

In interactive mode, *create* also allows the device name to be changed. *Device* with no arguments will print out the current *dev_name*.

h[eader] */hd_file/*

s[cript] */scr_file/*

b[uild] */bld_file/*

These commands specify files to be used for header text, install script, and build script when creating a tape. The files are copied to temporary workfiles for later use by the *create* command. The *edit* and *create* commands allow these temporary copies to be edited.

If no filename is given, and *installsw* is in interactive mode, the temporary file is created from keyboard input. Input is terminated with **^D** as the first character in a line. The *edit* command can also be used to create these temporary files.

e[dit] [h] [s] [b]

This command allows the user to edit or create the temporary copies of *hd_file*, *scr_file*, and *bld_file*, respectively. More than one option may be specified. If no options are specified, all files will be edited. On the CONVEX computer system, the editor that is specified in the EDITOR environment string, is used. *vi* is the default editor. On the SPU, the *xed* editor is used.

This command can only be used in interactive mode.

v[erify] [h] [s] [b] [t]

This command displays the contents of the temporary copies of *hd_file*, *scr_file*, and *bld_file*, respectively; or, for the **t** option, the header and script files from the tape at the current *dev_name*. If no option is given, only the header file from the tape is displayed.

When displaying the header file on tape, the copyright notice, and system generated date/time stamp will be seen prepended to the user's header file.

c[reate]

Create an *installsw* tape. If in interactive mode, the user is allowed to edit the work copies of *hd_file*, *scr_file*, and *bld_file*. He is also allowed to change *dev_name* for CONVEX computer software. When the user considers all of these data acceptable, **create** writes the temporary copies of the header and script files to the release tape. (The header file is prepended with the standard copyright notice and system date/time stamp). All user interaction is skipped if *installsw* is not in interactive mode. After the header and install script have been written, then the shell script in *bld_file* is executed. The first argument supplied to the script is the pathname to which the data is to be written. For magnetic tape on the CONVEX computer, or cartridge tape on the SPU, data files can be written to the device directly or via any utility (such as *tar*). For SPU cartridge tape access from the CONVEX computer, the *bld_file* script must create one or more temporary files and use *ctar* to transfer the files to the device at the supplied pathname. *Ctar* allows superuser to write to the SPU cartridge tape.

i[nstall]

This command installs a tape. The device used may be set with the *d* command, or the **–d** option on the *installsw* command line. The header file is checked to see if it is a legal *installsw* tape. If so, the header file is displayed on the standard output. The user sees the header file prepended with the standard copyright notice and the date/time stamp when the release tape was created. If in interactive mode, the user is prompted for permission to continue. If the response is yes, the script file is read from the tape and

executed. The first argument *installsw* supplies to the script is the pathname of the installation device. If using standard magnetic tape from the CONVEX computer, or cartridge tape from the SPU, the data files may be read directly by the script. If using cartridge tape from the CONVEX computer, the script file should use *ctar* to load the necessary files, and then perform any moving or renaming necessary.

a[uto] *in_file*
This command is the same as specifying the **–f** argument when invoking *installsw*. Further commands are taken from *in_file*, until an end-of-file is reached. *Auto* commands can be nested. *In_file* may also be terminated by typing **quit**, or <.> on a line by itself.

When *installsw* is in interactive mode, command files are run with full user interaction. (Command files executed by invoking *installsw* with the **–f** option are run noninteractively.)

q[uit] and <period>
Terminate the current command file. Terminate *installsw* if at the prompt –>. Command files are automatically terminated by reaching EOF (**quit** or <period> in a command file is unnecessary).

In interactive mode *installsw* may also be terminated by responding to the prompt with a **^D** <control-D>.

!shell_cmd
Execute the shell command following the !.

? Print a list of the available commands to the standard output.

The following text is a comment. No processing is performed. This is provided to allow commentary inside command files.

installsw sets the file creation mask to 022 (see *umask*(2)). Only the owner will have write access to files created by *installsw*. If this value is unsatisfactory, the umask can be set in the user's *build* and *install* scripts.

USER BUILD AND INSTALL SCRIPTS

User *bld_file* and *scr_file* scripts will be executed by the standard Bourne shell. All facilities for controlling the execution of the shell are described in the shell documents.

installsw attempts to gather the exit status from script executions. If error status is returned via "exit" commands, it will be reported, and the *installsw* execution will be terminated. Note that some commands (such as *tar*) that may be used in these scripts do not always report proper status on all exceptions.

installsw inherits its environment, *$PATH, $TERM,* etc., from the user's shell. If the build or install script depends on a special environment, the environment should be set in the build and install scripts.

DEVICES

When executed on the CONVEX CPU, *installsw* may utilize magnetic tape drives, with */dev/rmt12* as the default drive. It is also possible to access the cartridge tape on the SPU from the CONVEX computer. This is done by specifying device name *ct*.

When executed on the SPU, *installsw* uses only the SPU cartridge tape drive.

In cartridge tape usage, the header and script files are written from disk files */tmp/install1* and */tmp/install2*, respectively, to file */dev/rct0b*, using *tar* or *ctar*. Therefore, they can be recovered in another *installsw* session by using the same filenames. The install data is written by the *bld_file* script onto file */dev/rct0e*. When writing from the CONVEX computer system onto the SPU cartridge tape, the user supplied script must use the *ctar* utility. There may be up to approximately 17 Mbytes of data. Only the superuser may create, read, or install software on a cartridge tape, since only the superuser may execute *ctar*.

COMPATIBILITY

This version of *installsw* is fully compatible with the previous version. However, the latest version passes the name of the tape drive as an argument to the user-supplied installation script; the old *installsw* DID NOT. Therefore, installation scripts should be coded to check the argument count, and default to */dev/rmt12* if no tape drive argument is supplied by the version of *installsw* being used. This can be done as shown below:

```
#
# Product foo installation script
#
if $#argv === 0 then
   set drive=/dev/rmt12
else
   set drive=$1
endif

        .
        .       User supplied code
        .

tar xvf $drive
```

EXAMPLE

As a straightforward method of using *installsw*, the following procedure is suggested. Four files are required:

in_cmds	*installsw* command input file
Header	Tape header file
Build	Tape build script
Script	Tape install script

The *installsw* command file, *in_cmds*, should contain the following commands:

```
device      /dev/rmt12
header      Header
script      Script
build       Build
verify      h
create
quit
```

The tape header file, *Header*, can contain any ASCII text and should specify useful information about the tape such as the contents of the tape and the tape release date. For example:

```
Product:        Utilities V2.0
Release date:   Jul 1 1985
Directories:    /mnt/bin, /mnt/test
```

The tape build script, *Build*, should consist of the commands to build the tape:

```
#! /bin/sh
# Utilities Build Script
#
TAPE=/dev/rmt12              # default tape unit
if [ -n "$1" ]               # is $1 set
then
   TAPE="$1"
fi
tar cvf $TAPE bin test
mt -f $TAPE rew
```

Finally, the tape install script, *Script*, should consist of all commands necessary to install the tape on the target system:

```
#! /bin/sh
# Utilities Installation Script
#
TAPE=/dev/rmt12              # default tape unit
if [ -n "$1" ]               # is $1 set
then
   TAPE="$1"
fi
LOADDIR=/usr
#
cd $LOADDIR
/bin/echo "——" V2.0 Utilities Update
/bin/echo -n "—— V2.0 Utilities installation begun "
/bin/date
#
#  load the new release on top of the old one
#
/bin/echo "——" Extracting V2.0 Utilities Update from \'$TAPE\' into \"/bin/pwd\".
/bin/tar xf $TAPE 2>&1 /dev/null
/bin/mt -f $TAPE rew
/bin/echo -n "—— $LOADDIR V2.0 Utilities installation completed "
/bin/date
```

Note that since *installsw* checks the status returned by *Build* and *Script*, it is good practice to include an *exit 0* command at the end of the script. This prevents nonzero exit status from being returned inadvertently as can happen if certain commands (such as *test*) are the last commands to be executed by the script.

Having established these four files, *installsw* format tapes can be created by executing the command:

/etc/installsw -f in_cmds

The resultant tape can then be installed on another system via the command:

/etc/installsw -i

FILES

Temporary files are:

*/tmp/Ins_B**
*/tmp/Ins_H**
*/tmp/Ins_S**
*/tmp/Ins_Y**
*/tmp/Ins_Z**
/tmp/install1
/tmp/install2

SEE ALSO

tar(1), mtio(4), dump(8), restore(8)

NAME

lpc – line printer control program

SYNOPSIS

/etc/lpc [command [argument ...]]

DESCRIPTION

The *lpc* command is used by the system administrator to control the operation of the line printer system. For each line printer configured in /etc/printcap, *lpc* may be used to:

- disable or enable a printer,
- disable or enable a printer's spooling queue,
- rearrange the order of jobs in a spooling queue,
- find the status of printers, and their associated spooling queues and printer daemons.
- redirect spool queues locally or globally.

Without any arguments, *lpc* will prompt for commands from the standard input. If arguments are supplied, *lpc* interprets the first argument as a command and the remaining arguments as parameters to the command. The standard input may be redirected causing *lpc* to read commands from file. Commands may be abbreviated; the following is the list of recognized commands.

? [command ...]

help [command ...]
: Print a short description of each command specified in the argument list, or, if no arguments are given, a list of the recognized commands.

abort { all | printer ... }
: Terminate an active spooling daemon on the local host immediately and then disable printing (preventing new daemons from being started by *lpr*) for the specified printers.

clean { all | printer ... }
: Remove all files beginning with "cf", "tf", or "df" from the specified printer queue(s) on the local machine.

enable { all | printer ... }
: Enable spooling on the local queue for the listed printers. This will allow *lpr* to put new jobs in the spool queue.

exit

quit
: Exit from lpc.

disable { all | printer ... }
: Turn the specified printer queues off. This prevents new printer jobs from being entered into the queue by *lpr*.

down printer
: Stop a spooling daemon after the current job completes and disable spooling.

redirect [all] source-printer destination-printer
: Redirect the jobs from the source-printer to the destination printer. If *all* is specified, redirect all jobs submitted from anywhere in the network to the source-printer to the destination-printer. Otherwise, redirect only those jobs submitted from the machine on which the redirect is issued. If a local redirection is performed on the source-printer's host machine, all jobs submitted to that printer from the network will be redirected to the destination-printer.

restart { all | printer ... }

Attempt to start a new printer daemon. This is useful when some abnormal condition causes the daemon to die unexpectedly leaving jobs in the queue. *Lpq* will report that there is no daemon present when this condition occurs.

start { all | printer ... }
Enable printing and start a spooling daemon for the listed printers.

status [all] [printer ...]
Display the status of daemons and queues on the local machine.

stop { all | printer ... }
Stop a spooling daemon after the current job completes and disable printing.

topq printer [jobnum ...] [user ...]
Place the jobs in the order listed at the top of the printer queue.

undirect [all] printer
Remove the queue redirection from the printer. If *all* is specified, remove the redirection from the printer's host machine. Otherwise, remove the redirection from the local machine.

up printer
Enable the printer and restart the spooling daemon.

FILES

/etc/printcap	printer description file
/usr/spool/*	spool directories
/usr/spool/*/lock	lock file for queue control

SEE ALSO

lpd(8), lpmv(1), lpr(1), lpq(1), lprm(1), printcap(5)

DIAGNOSTICS

?Ambiguous command	abbreviation matches more than one command
?Invalid command	no match was found
?Privileged command	command can be executed by root only

NAME

lpd – line printer daemon

SYNOPSIS

/usr/lib/lpd [–d] [–l] [–f hosts.equivfile]

DESCRIPTION

lpd is the line printer daemon (spool area handler) and is normally invoked at boot time from the *rc*(8) file. It makes a single pass through the *printcap*(5) file to find out about the existing printers and prints any files left after a crash. It then uses the system calls *listen*(2) and *accept*(2) to receive requests to print files in the queue, transfer files to the spooling area, display the queue, or remove jobs from the queue. In each case, it forks a child to handle the request so the parent can continue to listen for more requests. The Internet port number used to rendezvous with other processes is obtained with *getservbyname*(3). The –l flag causes *lpd* to log valid requests received from the network. This can be useful for debugging purposes. Note that *lpd* uses the *syslog* facility to log error and diagnostic messages with facility id **LOG_LPR .**

Access control for UNIX domain (i.e. local) client connections is provided by the file system permissions of the printer daemon socket */dev/printer*. (UNIX is a registered trademark on UNIX System Laboratories, Inc.) Access control for Internet domain connections demands that requests for service arise from sockets bound to privileged ports on the client machine, and that the client is one of the machines listed in the file *hosts.equiv*. Additionally, if the **rs** capability is specified in the *printcap* entry for the printer being accessed, *lpr* requests will only be honored for those users with accounts on the machine with the printer. The **–f** option changes the default *hosts.equiv* file. The default *hosts.equiv* file is */etc/hosts.equiv*.

The file *lock* in each spool directory is used to prevent multiple daemons from becoming active simultaneously, and to store information about the daemon process for *lpr*(1), *lpq*(1), *lpmv*(1), and *lprm*(1). After the daemon has successfully set the lock, it scans the directory for files beginning with *cf*. Lines in each *cf* file specify files to be printed or non-printing actions to be performed. Each such line begins with a key character to specify what to do with the remainder of the line.

J Job Name. String to be used for the job name on the burst page.

C Classification. String to be used for the classification line on the burst page.

L Literal. The line contains identification information from the password file and causes the banner page to be printed.

T Title. String to be used as the title for *pr*(1).

H Host Name. Name of the machine where *lpr* was invoked.

P Person. Login name of the person who invoked *lpr*. This is used to verify ownership by *lprm* and *lpmv*.

M Send mail to the specified user when the current print job completes.

f Formatted File. Name of a file to print which is already formatted.

l Like **f** but passes control characters and does not make page breaks.

p Name of a file to print using *pr*(1) a filter.

t *Troff* File. The file contains *troff*(1) output (*cat* phototypesetter commands).

d DVI File. The file contains *Tex*(l) output (DVI format from Stanford).

g Graph File. The file contains data produced by *plot*(3X).

c *Cifplot* File. The file contains data produced by *cifplot*.

v The file contains a raster image.

r The file contains text data with FORTRAN carriage control characters.

1 *Troff* Font R. Name of the font file to use instead of the default.

2 *Troff* Font I. Name of the font file to use instead of the default.

3 *Troff* Font B. Name of the font file to use instead of the default.

4 *Troff* Font S. Name of the font file to use instead of the default.

W Width. Changes the page width (in characters) used by *pr*(1) and the text filters.

I Indent. The number of characters to indent the output by (in ASCII).

U Unlink. Name of the file to remove upon completion of printing.

N File name. The name of the file that is being printed, or a blank for the standard input when *lpr* is invoked in a pipeline.

A Accounting information: user ID, group ID, activity ID, and printer name. This information is present in the control file if the *pu* capability is set in the selected printer's printcap entry. *lpd* propagates this information to whatever print filter is used to process the job. See *lpd-acct*(5) for more information.

If a file can not be opened, a message will be placed in the log file (normally the console). *lpd* will try up to 20 times to reopen a file it expects to be there, after which it will skip the file to be printed.

lpd uses *flock*(2) to provide exclusive access to the lock file and to prevent multiple daemons from becoming active simultaneously. If the daemon should be killed or die unexpectedly, the lock file need not be removed. The lock file is kept in a readable ASCII form and contains two lines. The first is the process ID of the daemon; the second is the control file name of the current job being printed. The second line is updated to reflect the current status of *lpd* for the programs *lpq*(1), *lprm*(1), and *lpmv*(1).

FILES

/etc/printcap	printer description file
/usr/spool/*	spool directories
/dev/lp*	line printer devices
/dev/printer	socket for local requests
/etc/hosts.equiv	lists machine names allowed printer access

SEE ALSO

lpmv(1), lpq(1), lpr(1), lprm(1), syslog(3), hosts.equiv(5), lpd-acct(5), printcap(5), lpc(8), lprewind(8), pac(8)
Setting Up the Line Printer System in *Managing ConvexOS: Configuration Guide*
Managing the Line Printer System in *Managing ConvexOS: Operations Guide*

NAME

lpf – general line printer filter

SYNOPSIS

lpf [**–c**] [**–h** *host*] [**-i***indent*] [**-l***length*] [**–n** *name*] [**–u** *userinfo*] [**-w***width*] [*acctfile*]

DESCRIPTION

lpf is a general-purpose filter for use with the line printer spooler system. It is not meant to be invoked directly by a user (the filter is normally specified in the *printcap*(5) file with the **"if"** option, and invoked by *lpd*(8) when necessary).

The following options are understood by *lpf*:

–c Print control characters, instead of processing them out of the text.

–h *host* Specify the name of the host from which the file came.

-i*indent* Specify the indent level (left margin).

–l*length* Specify the length of the page to be printed. The default is 66 lines per page.

–n *name* Specify the login name of the user who printed the file.

–u *userinfo* Specify user information, which is included as part of the log entry for this job when the job is logged in the accounting file. This information consists of a user ID, a group ID, an activity ID, and a printer name, and is passed as a single command-line argument with each field separated by a space. This information is passed to *lpf* by *lpd*(8) if the *pu* capability (propagate user information) is set in */etc/printcap*(5).

–w*width* Specify the width of the page to be printed. The default is 132 columns per page.

acctfile Specify the name of a file to receive printer accounting information. This information may be subsequently read and processed by *pac*(8). *acctfile* must be the last argument on the command line.

Besides performing printer accounting, *lpf* processes and correctly prints overstruck lines generated by *nroff*(1), much in the same way that *ul*(1) does.

Some sites may wish to write their own print filters, based on their accounting needs. As a general guide for writing site specific filters, the */usr/lib/filter* directory contains a sample filter program written in **C**, a compiled object form of the *log* routine, a sample makefile (see make(1)), and a README file which describes each of these files in more detail. These files are intended to be used only as a template.

SEE ALSO

lpr(1), nroff(1), printcap(5)
Setting Up the Line Printer System in *Managing ConvexOS: Configuration Guide*
Managing the Line Printer System in *Managing ConvexOS: Operations Guide*

BUGS

If *lpf* is used, *lprm*(1) cannot be used to kill print jobs.

NAME

lprewind, lprew-daemon, lpf-rew – line printer rewind commands

SYNOPSIS

/usr/lib/lprewind *numpages printer*
/usr/lib/lprew-daemon *printer*
/usr/lib/lpf-rew [**–c**] [**–d** *printer*] [**–h** *host*] [**–l***length*] [**–n** *name*] [**–w***width*] [*acctfile*]

DESCRIPTION

Lprewind, *lprew-daemon*, and *lpf-rew* are utilities used to give a *dumb* printer a buffer of 10 printed pages. *Lprew-daemon* is a daemon process whose task is to remember the last 10 pages sent to the line printer and write data to the printing device *printer*. *Lprew-daemon* should be invoked in */etc/rc.local*.

Lprewind is a program that interrupts printing on the device specified in *printer*, tells *lprew-daemon* to re-print the previous *numpages* pages (where *numpages* is a number between 1 and 10), and resumes normal printer operation.

Lpf-rew is a general-purpose filter for use with the line printer spooler rewind system. It is used in conjunction with *lprew-daemon* and *lprewind*. Contrary to normal line printer filter actions, it sends data to a socket named *printer*-sock rather than directly to the printing device *printer*. It is not meant to be invoked directly by a user (the filter is normally specified in the *printcap*(5) file with the **if** option, and invoked by *lpd*(8) when necessary). An example of a *printcap*(5) entry using *lpf-rew* is:

```
lp|local line printer:\
        :af=/usr/adm/lpd-acct:if=/usr/lib/lpf_rew.sh:mx=0:\
        :lp=/dev/null:sd=/usr/spool/lpd:lf=/usr/adm/lpd-errs:
```

Notice that the line printer device in *printcap* is specified as **/dev/null**. This is done because data are sent to the socket *printer-sock* not to the actual printing device. If the actual printing device had been specified, *lpf-rew* would fail; two processes cannot open the printing device simultaneously.

Lpf_rew.sh is a shell script that invokes *lpf-rew* with an additional argument not specified by *lpd*(8). The text of *lpf_rew.sh* is:

```
#! /bin/sh
/usr/lib/lpf-rew –d /dev/lp $*
```

The following options are understood by *lpf-rew*:

–c Print control characters, instead of processing them out of the text.

–d *printer* Specify the name of the printing device.

–h *host* Specify the name of the host from which the file came.

–l*length* Specify the length of the page to be printed. The default is 66 lines per page.

–n *name* Specify the login name of the user who printed the file.

–w*width* Specify the width of the page to be printed. The default is 132 columns per page.

acctfile Specify the name of a file to receive printer accounting information. This information may be subsequently read and processed by *pac*(8). *Acctfile* must be the last argument on the command line.

Besides performing printer accounting, *lpf-rew* processes and correctly prints overstruck lines generated by *nroff*(1), much in the same way that *ul*(1) does.

FILES

/dev/lp*	line printer devices
/dev/*-sock	line printer rewind socket
/dev/*-rew	line printer rewind socket
/etc/printcap	printer description file
/etc/rc.local	local additions to reboot file
/usr/spool/lpd/lprewind.lock	lprew-daemon lock file
/usr/spool/lpd/logfile	lprew-daemon logfile

SEE ALSO

lpr(1), nroff(1), ul(1), printcap(5), lpd(8), rc(8)
Setting Up the Line Printer System in *Managing ConvexOS: Configuration Guide*
Managing the Line Printer System in *Managing ConvexOS: Operations Guide*

BUGS

Lpf-rew prints the burst page for each print job but does not know the jobname for the job.

The burst page is included as a page in the memory of *lprew-daemon*.

The interface between *lprew-daemon*, *lprewind*, *lpf-rew* utilities and *lpd* is cumbersome and should be integrated into the functionality of *lpd*.

NAME

makedev, MAKEDEV – make system special files

SYNOPSIS

/dev/MAKEDEV [NAME...]

DESCRIPTION

MAKEDEV is a shell script normally used to install special files. It resides in the */dev* directory, as this is the normal location of special files. Since all devices are created using *mknod*(8), this shell script is useful only to the super-user.

NAME is the name of the devices to be created. Valid arguments are -

Name	*Description*
std	Standard devices
local	Configuration specific devices
ta*	Multibus and VMEbus tapes (MTC-001, MTC-201).
tc*	VME 3480 tapes (MTC-202)
da*	Multibus disks (DKC-001, DKC-002)
dd*	VMEbus disks (DKC-203 DKC-204)
du*	IDC disks (DKC-IP2)
ca*	Multibus and VMEbus terminal multiplexors (ACM-001, ACM-201)
pty*	set of 16 master and slave pseudo terminals
pa*	Multibus and VMEbus printers (PRC-001, PTR-CEN, PTR-DAT)
pb*	Versatec Plotter (PRC-002)
dm*	Parallel Interface to Raster Tech Model One/80 (GPI-001)

DIAGNOSTICS

Either self-explanatory, or generated by one of the programs called from the script. Use "sh -x MAKEDEV" in case of trouble.

SEE ALSO

intro(4), mknod(8)

NAME

makekey – generate encryption key

SYNOPSIS

/usr/lib/makekey

DESCRIPTION

Makekey improves the usefulness of encryption schemes depending on a key by increasing the amount of time required to search the key space. It reads 10 bytes from its standard input, and writes 13 bytes on its standard output. The output depends on the input in a way intended to be difficult to compute (that is, to require a substantial fraction of a second).

The first eight input bytes (the *input key*) can be arbitrary ASCII characters. The last two (the *salt*) are best chosen from the set of digits, upper- and lower-case letters, and '.' and '/'. The salt characters are repeated as the first two characters of the output. The remaining 11 output characters are chosen from the same set as the salt and constitute the *output key.*

The transformation performed is essentially the following: the salt is used to select one of 4096 cryptographic machines all based on the National Bureau of Standards DES algorithm, but modified in 4096 different ways. Using the input key as key, a constant string is fed into the machine and recirculated a number of times. The 64 bits that come out are distributed into the 66 useful key bits in the result.

Makekey is intended for programs that perform encryption (for instance, *ed* and *crypt*(1)). Usually makekey's input and output will be pipes.

SEE ALSO

crypt(1), ed(1)

NAME

makewhatis – rebuild the whatis database

SYNOPSIS

/usr/lib/makewhatis [**–v**] [**–n**] [**–y**] [[**–M**] manpath]

DESCRIPTION

The *makewhatis* program rebuilds the text and *dbm*(3X) forms of the *whatis* database from the on-line manual pages for a given *manpath*, or from */usr/man* if none is supplied. These files are used by the *man*(1), *whatis*(1), and *apropos*(1) programs to locate man pages and to print out the one-line descriptions given by *whatis* and *apropos*. The *makewhatis* program should be run whenever new man pages are added so these programs can find them.

Each component of the colon-delimited *manpath* is interpreted to be the root of a new tree containing directories of the form *man**, within which are unformatted man pages of the form **.**. A separate *whatis* file and corresponding *dbm* files, *whatis.pag* and *whatis.dir*, will be placed at the root of that man tree. Files or directories ending in tilde (~) or in **.bak** or **.old** (in either case) will be skipped, as will a section named **man0**, should it exist.

In general, files in section **manX** should end in **.X***, where **X** is a section of the manual like **1** or **3x11**. This means it's ok to put things ending in **1c** in **man1**, but not things ending in **3s**. Any man page whose name doesn't match these criteria will generate a warning, but will be processed anyway. The exception to this is the **mano** subdirectory, which has traditionally been the receptacle of all obsolete man pages irrespective of their file extension. A better way to do that would be to have an entire man tree dedicated to this, as in */usr/old/man/*.

Multiple entries occurring in the NAME section or as links (hard, soft, or via **.so** inclusion) will all be stored under the same man page name. This method can save significant amounts of disk space because it guarantees that only one cat page need be generated, regardless of how many ways you can get at the corresponding man page. Maintaining aliases as links or via a one-line file that **.so**'s the real man page (once the only way to do this) is still supported although no longer required; mere inclusion in the NAME section is sufficient.

Embedded point and font changes will be removed from the output, and *troff* string macros for hyphens and underscores will be translated into their corresponding ASCII representations. In general, all that can be done will be done to produce nicely readable output for *whatis*(1).

The **–v** flag will generate verbose output, reporting such things as each new man tree examined, each subdirectory, each file that is opened, and each entry in the NAME section that is found and stored. Simple tracing information is printed to the standard output, while to standard error are directed more serious warnings.

The **–n** option can be used to check whether the database needs rebuilding. It will report on the first file in each man path that is out of date. Note that this is the only automated way to determine this; neither *makewhatis* nor *man* will realize that the database is out of date, so you should be careful to rebuild it whenever new man pages are added.

The **–y** option is similar, but after finding something out of date, *makewhatis* will automatically rebuild its database.

The **-M** option is used to specify a different path than $MANPATH on a *makewhatis* command.

Support for compressed man pages is provided in the following way: if the name of the subdirectory itself ends in **.Z**, as in **man8.Z**, then all its files are assumed to have been compressed with *compress*(1L). Alternatively, individual files ending in **.Z** are also considered to be compressed. Compressed files are processed with *zcat*(1L).

EXAMPLES

```
% makewhatis -y                  # build /usr/man database only if out of date
% makewhatis /usr/local/man      # make local man page database
% makewhatis -v ~/man            # make personal man page database verbosely
```

```
% makewhatis -n $MANPATH          # tell if any dbase in $MANPATH out of date
```

FILES

/usr/man/whatis	default whatis database, text version
/usr/man/whatis.pag	*dbm* index file for default whatis database
/usr/man/whatis.dir	*dbm* data file for default whatis database
/usr/man/man/*.**	default (unformatted) man pages
/usr/man/cat/*.**	formatted man pages

SEE ALSO

man(1), whatis(1), apropos(1), perl(1), compress(1L), dbm(3X), man(7), catman(8)

DIAGNOSTICS

Numerous diagnostics may be issued, especially if the **–v** flag has been given. Here are messages that may require further attention. The **%s** and **%d** fields in the descriptions indicate strings and integers respectively that will be filled in appropriately at run time, while **%m** indicates a standard system error message as described in *intro*(2). See the source for further details.

`Skipping non-man file: %s`
A file was found in a man directory that has no dot in its name.

`%s has a funny extension to be in %s`
An apparent man file was found whose extension doesn't match the name of the man file in which it was found. This restriction does not apply to **mano** subdirectories.

`can't stat %s: %m`
The *stat*(2) call returned an error. This can be caused by a symbolic link pointing to a non-existent file.

`can't open %s: %m`
The *open*(2) call returned an error, which is listed.

`can't chdir back to %s: %m`
After changing directory to one of the man subdirectories, *makewhatis* couldn't return to the initial root man tree. This is a fatal error.

`makewhatis: %s: found %d entries in %d files`
For the given root man directory, this many separate entry points were found for this many different files.

`%s .so references non-existent %s`
A file contains a **.so** reference to a missing man page.

`%s's .TH thinks it's " "" "ConvexOS V9.0 Programmer's Reference" " "" "ConvexOS Programmer's Reference"`
The man page's internal TH section has a different idea of where it lives than the actual file name.

`trimmed troff string macro in NAME section of %s`
An irresolvable *troff* string macro was found within the NAME section of the man page. This may cause peculiar *whatis* output.

`%s: no separated dash in %s`
The NAME section contained no dash in it to separate the list of man entries from their descriptions. This will also cause odd *whatis* output.

%s: truncating cmdlist from %d to %d bytes for DBM's sake
The NAME section contained too many characters. Due to built-in limitations of *dbm*(3X) data entries, this entry was truncated. It will appear in *whatis* output with a trailing ... at the point of truncation.

%s: forgot my own name!
A man page is stored in a file name that doesn't correspond to any of the entries in its NAME section.

%s: no NAME lines, so has no whatis description!
No parsable entries in the NAME section were discovered.

%s was a .so alias for %s, but %s's NAME section doesn't know it!
A **.so** reference was found to point to a man page whose own NAME section didn't contain the name of the **.so** file. This may be due to old man page aliases that were never de-installed when the base man page changed.

can't store %s: would break DBM
There were too many entries for a topic for it to be stored without exceeding inherent *dbm*(3X) data size limitations.

NOTES

This version of *makewhatis* is written entirely in the *perl* programming language; it requires that *perl* be installed on the system to run.

RESTRICTIONS

The NAME section should be a comma-separated list of aliases for this man page, followed by white space, a dash, more white space, and then the description. Pages not conforming to this rule will still be found, but a diagnostic will be printed and their descriptions will be left blank.

Man pages should be formatted using the *man*(7) macro set. The only exceptions to this are the man pages from the *Rand MH Message Handling System*, whose own internal macro set is also recognized, and Larry Wall's **.Sh** section header macro.

makewhatis takes much longer to run if man pages are stored in compressed form.

BUGS

Not all systems have *compress* installed on them.

AUTHOR

Tom Christiansen <*tchrist@convex.com*>

COPYRIGHT

NAME

mkfs – construct a file system

SYNOPSIS

/etc/mkfs [**-E** *pattern*] *special size* [*nsect*] [*ntrack*] [*blksize*] [*fragsize*] [*maxcontig*] [*rotdelay*] [*ncpg*] [*minfree*] [*rps*] [*nbpi*] [*sectspare*]

DESCRIPTION

NOTE: file systems are normally created with the *newfs*(8) and *newst*(8) commands.

Mkfs constructs a file system by writing on the special file *special*. *Size* specifies the number of physical sectors in the file system. *Mkfs* builds a file system with a root directory and a *lost+found* directory (see *fsck*(8)). The number of i-nodes is calculated as a function of the file system size. No boot program is initialized by *mkfs* (see *newfs*(8)).

The optional arguments allow fine tune control over the parameters of the file system. Since the file system parameters are non-trivially inter-related, a complete yet simple description of acceptable values for them is not possible. *Mkfs* does the necessary checking for compatibility, and prints specific error messages when incompatible parameters are given.

The **-E** *pattern* flag specifies that the entire volume should be overwritten with the *pattern* before the file system is initialized. *Pattern* is a 32 bit hex number.

Note that each of the other arguments may optionally be specified, but, if you specify one option, you must also specify all the options to the left of that one.

Nsect specifies the number of physical sectors per track on the disk.

Ntrack specifies the number of tracks per cylinder on the disk.

Blksize gives the primary block size for files on the file system. It must be a power of two within the range 4096 and 65536 (4Kbytes and 64Kbytes).

Fragsize gives the fragment size for files on the file system. The *fragsize* represents the smallest amount of disk space that will be allocated to a file. It must be a power of two within the range 1/8th *blksize* and *blksize*.

Maxcontig sets the default for the maximum number of blocks that may be allocated sequentially. Since ConvexOS drivers are not capable of scheduling multi-block transfers, this defaults to 1.

Rotdelay gives the minimum number of milliseconds to initiate another disk transfer on the same cylinder. It is used in determining the rotationally optimal layout for disk blocks within a file; the default is 4.

Ncpg specifies the number of disk cylinders per cylinder group. This number must be in the range 1 to 32.

Minfree specifies the minimum percentage of free disk space allowed. Once the file system capacity reaches this threshold, only the superuser is allowed to allocate disk blocks. The default value is 10%.

If a disk does not revolve at 60 revolutions per second, the *rps* parameter may be specified. Users with special demands for their file systems are referred to the paper cited below for a discussion of the tradeoffs in using different configurations.

Nbpi specifies the density of i-nodes in the file system. The default is to create an i-node for each 2048 bytes of data space. If fewer i-nodes are desired, a larger number should be used; to create more i-nodes a smaller number should be specified.

sectspare specifies the number of spare physical sectors per cylinder. This is required by the IDC.

SEE ALSO

fs(5), dir(5), fsck(8), newfs(8), newst(8)

McKusick, Joy, Leffler; "A Fast File System for Unix", Computer Systems Research Group, Dept of EECS, Berkeley, CA 94720; TR #7, September 1982.

BUGS

There should be some way to specify bad blocks.

File systems are currently limited to 2 gigabytes in size.

NOTES

Historically, disk sectors have always been 512 bytes in size. With the advent of the IDC, however, sector sizes are now variable. Therefore, *size, nsect,* and *sectspare* should be expressed in terms of the physical sector size of the device, not in terms of 512 byte units.

mkfs is capable of creating file systems that are larger than two gigabytes in size. No special switches or configurations are needed, just using the standard parameters with large enough values will allow the administrator to create file systems of this size.

NAME

mklost+found – make a lost+found directory for fsck

SYNOPSIS

/etc/mklost+found

DESCRIPTION

A directory *lost+found* is created in the current directory and a number of empty files are created therein and then removed so that there will be empty slots for *fsck*(8). This command should not normally be needed since *mkfs*(8) automatically creates the *lost+found* directory when a new file system is created.

SEE ALSO

fsck(8), mkfs(8)

NAME

mknf, rmnf – create and delete notesfiles

SYNOPSIS

mknf [**-aon**] topic [...]

rmnf [**-f**] topic [...]

DESCRIPTION

mknf and *rmnf* create and delete notesfiles respectively. The same parameters apply for each: the "topic" is the name by which the notesfile is known.

As *mknf* processes its arguments, creating new notesfiles, the name of each new notesfile is echoed to the terminal. The new notesfiles are closed and the *notesfile owner* is made the sole director. He customarily turns control over to the user requesting the notesfile by making that person a director.

The **-aon** options apply to *mknf* only. They signify that the notesfiles created are to permit anonymous notes, be open, and be networked respectively.

If the file "/usr/spool/notes/.utilities/access-template" is present, it contains a list of access-rights which are added to the created notesfile. The file contains lines of access-rights similar to those used in the *nfaccess*(8) command. In environments with dynamic creation of notesfiles, such as on USENET nodes, one or more users can automatically be made directors of all new notesfiles. If appropriate, they can be removed from the access list of newly created notesfiles as needed.

rmnf asks for verification of each notesfile before deleting it. The notesfile is deleted if the response line begins with a "y". If invoked with the **-f** option, *rmnf* does not ask for verification before deleting the notesfiles.

Only the *notesfile owner* is allowed to run *mknf* and *rmnf*.

BUGS

rmnf doesn't understand about absolute pathnames for notesfiles. It refuses to remove notesfiles specified by absolute pathnames.

Any user should be allowed to create private notesfiles in directories where he has permission.

FILES

/usr/spool/notes/.utilities	where these programs live.
/usr/spool/notes/.utilities/access-template	Default access-list.
/usr/spool/notes	Default notesfile directory.

SEE ALSO

nfaccess(8), nfxmit(8), notes(1), notesadm(8)
"Notesfile Reference Manual" in the *ConvexOS Tutorial Papers*

NAME

mknod – build special file

SYNOPSIS

/etc/mknod *name* [**c**] [**b**] *major minor*
/etc/mknod name **p**

DESCRIPTION

mknod makes a special file. The first argument is the *name* of the entry. In the first form, the second argument is **b** if the special file is block-type (disks, tape) or **c** if it is character-type (other devices, also called *raw* devices.) The last two arguments are numbers specifying the *major* device type and the *minor* device (e.g. unit, drive, or line number). Only the super-user is permitted to invoke this form of the *mknod* command.

In the second form, *mknod* makes a named pipe (fifo).

The first form of *mknod* is only for use by a system administrator in order to configure a custom device driver into the kernel. Normally you should use */dev/MAKEDEV* instead when making special files.

The assignment of major device numbers is specific to each system release. For the 8.0 release of ConvexOS, the assignments are:

Block Devices

Device Number	*Device Type*	*Filename*
0 - 2	reserved for future use by CONVEX	
3	interface to swap space	/dev/drum
4	magnetic tape	/dev/mt*x*
5	Multibus disk partitions, units 0-31	/dev/da*x*
6	reserved for future use by CONVEX	
7	striped disk partitions	/dev/st*x*
8 - 15	reserved for future use by CONVEX	
16 - 27	reserved for user written device drivers	
28	VME disk partitions, units 0-31	/dev/dd*x*
29	VME disk partitions, units 32-63	/dev/dd*x*
30	VME disk partitions, units 64-95	/dev/dd*x*
31	VME disk partitions, units 96-127	/dev/dd*x*
32 - 47	reserved for user written device drivers	
48	Multibus disk partitions, units 32-63	/dev/da*x*
49	Multibus disk partitions, units 64-95	/dev/da*x*
50	Multibus disk partitions, units 96-127	/dev/da*x*
51 - 60	reserved for future use by CONVEX	
61	VME Ultranet controller interface	/dev/uv*x*
62 - 63	reserved for future use by CONVEX	
64	IDC disk partitions, units 0-511	/dev/du*x*
65 - 127	reserved for future use by CONVEX	
128- 254	reserved for Special Systems products	

Character Devices

Device Number	*Device Type*	*Filename*
0	system console	/dev/console
1	Channel Control Unit	
	minor device is CCU number	/dev/ccu*x*
2	logical memory devices	
	minor device 0 : memory	/dev/mem
	minor device 1 : kernel memory	/dev/kmem
	minor device 2 : the bit bucket	/dev/null
3	terminal driver, units 0-255	/dev/tty*x*
4	raw tape /dev/rmt*x*	
5	Multibus raw disk partitions, units 0-31	/dev/rda*x*
6	Line printer units	/dev/lp*x*
7	subdevice of disk driver	/dev/drum
8	indirect tty driver	/dev/tty
9	pseudo terminal lines	/dev/ttyp*x*
10	pseudo terminal lines	/dev/ptyp*x*
11	interface to ud driver	/dev/ud*x*
12	raw striped disk partitions	/dev/st*x*
13	DR-11W emulator (unsupported)	/dev/dm*x*
14	Versatec(tm) Plotter interface	/dev/pb*x*
15	raw Multibus HYPERchannel driver	/dev/hy*x*
16	HSP/HIA/ECHO test driver	/dev/hsp*xx*
	minor device 0-15 is logical channel number	
17	terminal driver, units 256-511	/dev/tty*x*
18	Reserved for X.25 devices	
19	raw Ultra network driver, units 0-255	/dev/unet*x*
20	net char device - kernel rpc interface	
21	net char device - kernel rpc interface	
22 - 27	reserved for future use by CONVEX	
28	VME raw disk partitions, units 0-31	/dev/rda*x*
29	VME raw disk partitions, units 32-63	/dev/rda*x*
30	VME raw disk partitions, units 64-95	/dev/rda*x*
31	VME raw disk partitions, units 96-127	/dev/rda*x*
32 - 47	reserved for user written device drivers	
48	Multibus raw disk partitions, units 32-63	/dev/rda*x*
49	Multibus raw disk partitions, units 64-95	/dev/rda*x*
50	Multibus raw disk partitions, units 96-127	/dev/rda*x*
51	COVUEnet controller download interface	/dev/xmem
52	COVUEnet network manager interface	/dev/netman
53	COVUEnet logical link interface	/dev/ll
54 - 60	resreved for future use by COVUEnet	
61	raw VME Ultranet controller interface	/dev/ruv*x*
62 - 63	reserved for future use by CONVEX	
64	IDC raw disk partitions, units 0-511	/dev/du*x*
65 - 127	reserved for future use by CONVEX	
128- 254	reserved for Special Systems products	
255	reserved for NFS	

DIAGNOSTICS

mknod exits with a status of 0 if successful, a status of 1 for usage errors and a status of 2 for other errors.

SEE ALSO

mknod(2) makedev(8)

NAME

mkpasswd – generate hashed password table(s)

SYNOPSIS

/etc/mkpasswd [**–v**] [**–p** *pswdfile*] [**–r** *restfile*]

DESCRIPTION

Mkpasswd is used to generate the hashed password database used by the library routines **getpwnam()** and **getpwuid(),** and/or the password restrictions database used by the library routines **getpwrestnam()** and **getpwrestuid().**

If the **–v** option is supplied, each entry will be listed as it is added. If the **–p** option is supplied, the file *pswdfile* will be used instead of the default /etc/passwd to create the passwords database. The file *pswdfile* must be in the format of /etc/passwd (see passwd(5)). If the **–r** option is supplied, the file *restfile* will be used instead of the default /etc/pwrestrict to create the restrictions database. The file *restfile* must be in the format of /etc/pwrestrict (see pwrestrict(5)).

If no options are given *mkpasswd* will use /etc/passwd to generate the passwords database. *Mkpasswd* uses a lock file /tmp/mkpasswd.lock to ensure that only one process at a time is running *mkpasswd. Mkpasswd* will generate database files named *pswdfile*.pag and *pswdfile*.dir and/or *restfile*.pag and *restfile*.dir. *Mkpasswd* will exit with a non-zero exit code if any errors are detected.

FILES

pswdfile.pag - database filenames
pswdfile.dir
restfile.pag
restfile.dir
/tmp/mkpasswd.lock- lock file

SEE ALSO

getpwent(3), getpwrestent(3), vipw(8), passwd(5), pwrestrict(5)

NAME

mount, umount – mount and dismount filesystems

SYNOPSIS

/etc/mount [**–p**]
/etc/mount –a[**fnvF**] [**–h** *host*]
/etc/mount –a[**fnvF**] [**–t** *type*]
/etc/mount [**–fnrvF**] [**–t** *type*] [**–o** *options*] *fsname dir*
/etc/mount [**–vfnF**] [**–o** *options*] *fsname* | *dir*

/etc/umount [**–t** *type*] [**–h** *host*]
/etc/umount –a[**v**]
/etc/umount [**–v**] *fsname* | *dir* ...

DESCRIPTION

mount announces to the system that a filesystem *fsname* is to be attached to the file tree at the directory *dir*. The directory *dir* must already exist. It becomes the name of the newly mounted root. The contents of *dir* are hidden until the filesystem is unmounted. If *fsname* is of the form "host:path", the filesystem type is assumed to be *nfs*.

umount announces to the system that the filesystem *fsname* previously mounted on directory *dir* should be removed. Either the filesystem name or the mounted-on directory may be used.

mount and *umount* maintain a table of mounted filesystems in */etc/mtab*, described in *mtab*(5). If invoked without an argument, *mount* displays the table. Thus the order in which the file systems are listed will be in the order of */etc/mtab*, not */etc/fstab*. If invoked with only one of *fsname* or *dir*, mount searches the file */etc/fstab* (see *fstab*(5)) for an entry whose *dir* or *fsname* field matches the given argument. For example, if this line is in */etc/fstab*:

/dev/da0e /usr 4.2 rw 1 2

then the commands **mount /usr** and **mount /dev/da0e** are shorthand for **mount /dev/da0e /usr**

MOUNT OPTIONS

–p Print the list of mounted filesystems in a format suitable for use in */etc/fstab*.

–a Attempt to mount all the filesystems described in */etc/fstab*. (In this case, *fsname* and *dir* are taken from */etc/fstab*.) If a type is specified all of the filesystems in */etc/fstab* with that type is mounted. Filesystems are not necessarily mounted in the order listed in */etc/fstab*.

–f Fake a new */etc/mtab* entry, but do not actually mount any filesystems.

–n Mount the filesystem without making an entry in **/etc/mtab**.

–v Verbose — *mount* displays a message indicating the filesystem being mounted.

–h The next argument is taken as the host whose **nfs** filesystems are to be mounted.

–t The next argument is the filesystem type. The accepted types are: **4.2**, and **nfs**; see *fstab*(5) for a description of these filesystem types.

–F Force the mount of a dirty filesystem. Normally, the system will not allow the mount of a filesystem with the dirty bit set. The best way to clear the dirty bit is to run *fsck*(8) on the filesystem. Forcing the mount of a dirty filesystem will clear the dirty bit, but the filesystem will remain dirty. This could eventually cause the system to panic.

–r Mount the specified filesystem read-only, even if the entry in **/etc/fstab** specifies that it is to be mounted read-write. This is a shorthand for:

mount –o ro *fsname dir*

Physically write-protected and magnetic tape filesystems must be mounted read-only, or

errors occur when access times are updated, whether or not any explicit write is attempted.

-o Specify *options*, a list of comma separated words from the list below. Some options are valid for all filesystem types, while others apply to a specific type only.

options valid on *all* (both **4.2** and **nfs**) file systems (the default is **rw,suid,noquota**):

rw read/write.

ro read-only.

suid set-uid execution allowed.

nosuid set-uid execution not allowed.

grpid Create files with BSD semantics for the propagation of the group ID. Under this option, files inherit the GID of the directory in which they are created, regardless of the directory's set-GID bit. Note that on ConvexOS, local 4.2 file systems always use BSD semantics for the file group ownership when creating files, regardless of the presence or absence of the grpid option.

quota usage limits enforced.

noquota usage limits not enforced.

hide ignore this entry during a **mount -a** command to allow you to define *fstab* entries for commonly used filesystems you don't want to automatically mount.

noauto a synonym for **hide**.

nolf disallow creation of files greater that two gigabytes in size in this filesystem.

remount If the file system is currently mounted, and if the entry in **/etc/fstab** specifies that it is to be mounted read-write or **rw** was specified along with **remount**, remount the file system making it read-write. If the entry in **/etc/fstab** specifies that it is to be mounted read-only and **rw** was not specified, the file system is not remounted. If the file system is not currently mounted, an error results.

options specific to **nfs** (NFS) file systems (the defaults are:

fg,retry=10000,timeo=7,retrans=3,port=NFS_PORT,hard,intr

with defaults for *rsize* and *wsize* set by the kernel):

bg if the first mount attempt fails, retry in the background.

fg retry in foreground.

retry=*n* set number times to retry mount to *n*.

rsize=*n* set read buffer size to *n* bytes.

wsize=*n* set write buffer size to *n* bytes.

timeo=*n* set NFS timeout to *n* tenths of a second.

retrans=*n* set number of NFS retransmissions to *n*.

port=*n* set server IP port number to *n*.

soft	return error if server doesn't respond.
hard	retry request until server responds.
intr	allow keyboard interrupts on hard mounts.
nointr	don't allow keyboard interrupts on hard mounts.
secure	Use a more secure protocol for NFS transactions.
acregmin=*n*	Hold cached attributes for at least *n* seconds after file modification.
acregmax=*n*	Hold cached attributes for no more than *n* seconds after file modification.
acdirmin=*n*	Hold cached attributes for at least *n* seconds after directory update.
acdirmax=*n*	Hold cached attributes for no more than *n* seconds after directory update.
actimeo=*n*	Set *min* and *max* times for regular files and directories to *n* seconds.

options specific to **local** (UFS) file systems for migration.

blklo=*n*

When *n* percent of the disk is full, a migration daemon will receive a low water event. A full disk is taken to be the point at which normal user allocation requests fail. This point is usually the total number of available blocks when the filesystem is created minus the number of blocks reserved by the filesystem (usually ten percent).

blkhi=*n*

When *n* percent of the disk is full, a migration daemon will receive a high water event.

spaceretry

Retry allocation requests if they fail because of insufficient number of disk blocks.

NFS FILESYSTEMS

Background vs. Foreground

The **bg** option causes *mount* to run in the background if the server's *mountd*(8C) does not respond. *mount* attempts each request **retry=***n* times before giving up. Once the filesystem is mounted, each NFS request made in the kernel waits **timeo=***n* tenths of a second for a response. If no response arrives, the time-out is multiplied by **2** and the request is retransmitted. When **retrans=***n* retransmissions have been sent with no reply a **soft** mounted filesystem returns an error on the request and a **hard** mounted filesystem prints a message and retries the request.

Read-Write vs. Read-Only

Filesystems that are mounted **rw** (read-write) should use the **hard** option to guard against incomplete writes.

Interrupting Processes With Pending NFS Requests

The **intr** option (default) allows keyboard interrupts to kill a process that is hung waiting for a response on a hard mounted filesystem. The **nointr** option does not allow keyboard interrupts to kill a process that is hung waiting for a response on a hard mounted filesystem; instead, the operation is guaranteed to complete when the server reboots.

Transfer size

The number of bytes in a read or write request can be set with the **rsize** and **wsize** options.

Secure Filesystems

The **secure** option must be given if the server requires secure mounting for the filesystem. Otherwise, all file accesses will be performed by the server with the effective uid specified in the "-anon" entry in its */etc/exports* file, which may mean permission failures on all accesses.

File Attributes

The attribute cache retains file attributes on the client. Attributes for a file are assigned a time to be flushed. If the file is modified before the flush time, then the flush time is extended by the time since the last modification (under the assumption that files that changed recently are likely to change soon). There is a minimum and maximum flush time extension for regular files and for directories. Setting **actimeo=***n* extends flush time by *n* seconds for both regular files and directories.

UMOUNT OPTIONS

–h *host* Unmount all filesystems listed in */etc/mtab* that are remote-mounted from *host*.

–t type
Unmount all filesystems listed in **/etc/mtab** that are of a given *type*.

–a Attempt to unmount all the filesystems currently mounted (listed in */etc/mtab*). In this case, *fsname* is taken from */etc/mtab*.

–v Verbose — *umount* displays a message indicating the filesystem being unmounted.

EXAMPLES

mount /dev/da0e /usr	mount a local disk
mount –at 4.2	mount all 4.2 filesystems
mount –t nfs serv:/usr/src /usr/src	mount remote filesystem
mount serv:/usr/src /usr/src	same as above
mount –o soft serv:/usr/src /usr/src	same as above but soft mount
mount –p > /etc/fstab	save current mount state

FILES

/etc/mtab	table of mounted filesystems
/etc/fstab	table of filesystems mounted at boot

DIAGNOSTICS

mount exits with the number of errors it encountered. This number does not include attempts at mounting currently mounted filesystems.

SEE ALSO

mount(2), unmount(2), fstab(5), mtab(5), mountd(8C), nfsd(8)

BUGS

Mounting filesystems full of garbage crashes the system.

NFS mounts directly under / cause problems when the server is down, because any process calling getwd(3) will result in an attempt to find information about the root of the remote file system. To ameliorate this problem, always mount a file system farther down in the local directory tree and make a symbolic link to the mount point. For example:

ln -s /rmt/server/test /test ; mount server:/test /rmt/server/test

If the directory on which a filesystem is to be mounted is a symbolic link, the filesystem is mounted on *the directory to which the symbolic link refers*, rather than being mounted on top of the symbolic link itself. Because of this behavior, the above example could also be done as:

ln -s /rmt/server/test /test ; mount server:/test /test

NOTES

When doing a **umount -a** operation, *umount* forks a copy of itself in order to send a broadcast request to all NFS server machines, instructing them to clear their mount entries for this host; see rmtab(5);

If a user attempts to mount a filesystem at a mount point which is not empty, *mount* will issue a warning message:

Mount point 'path' is not empty.

NFS is an optional product; for more information, contact your CONVEX sales representative.

NAME

mvst – move data from an existing stripe partition to a new disk partition

SYNOPSIS

mvst *[-vn] stripedev from_partition to_partition to_type*
mvst *-H [-vn] stripedev from_partition*

mvst *[-P pid -r o,ss] stripedev from_partition to_partition to_type*

DESCRIPTION

The *mvst* utility moves data from a component disk partition of a stripe to a different partition. For non-redundant stripes the device must be unmounted when the utility is called.

Prior to moving any data, the utility will check to make sure that enough disk space is available on the *'to partition'*. Additionally, if mvst is invoked on a non-redundant stripe, the utility will ensure that sector sizes are compatible and available. The raw device must not be opened by another user, otherwise this utility will refuse to copy data.

Switch options available:

–H Use disk specified in the Hot Spare list. With this option the *to_partition to_type* parameters will be ignored, if given.

–v Verbose mode. Print as much information as possible regarding the actions taking place.

–n Nochange mode. Prepare for reconfiguration, but make no changes to the database files or kernel tables and do not perform the data move.

–P pid For internal use by the VVM utilities only, included as part of the command line written to */etc/streconfig(5)*. This option specifies the original pid of the *mvst* process. It allows *rmst* to control data moves by signaling *mvst* at the pid specified. The option is ignored by *mvst*.

–r o,ss For internal use by the VVM utilities only, included as part of the command line written to */etc/streconfig(5)*. It specifies a restart offset (in logical blocks) into the to_partition and the to_partitions sector size (in bytes), which is information needed to restart an *mvst* process after a system crash.

During the transfer of data from one disk partition to another on a redundant stripe, a status entry is written to the file */etc/streconfig*. The */etc/streconfig* file contains the command line which invoked the *mvst(8)* utility exactly. Upon reboot, after a system crash, the *vvmdaemon (see st(4))* will read */etc/streconfig* and restart any unfinished *mvst* commands.

The user must have root privilege to execute this utility.

Entries in */etc/stripecap* are created with the *newst*(8) utility.

FILES

/etc/stripecap Database of the stripe descriptors.
/etc/streconfig Database of outstanding *mvst* processes.

WARNING

In the non-redundant case, *mvst*(8) does not check for sectional disk conflicts before performing the move. This means that it is possible to end up striping across two partitions from the same disk in a non-redundant stripe, which greatly impedes performance and causes disk arm thrashing.

SEE ALSO

st(4), stripecap(5), streconfig(5), newst(8), getst(8), putst(8), rmst(8), qst(8), convst(8)

NAME

named – Internet domain name server

SYNOPSIS

/usr/etc/named [**–d** *debuglevel*] [**–p** *port#*] [{–b} *bootfile]*

DESCRIPTION

Named is the Internet domain name server. See RFC883 for more information on the Internet name-domain system. Without any arguments, *named* will read the default boot file */etc/named.boot*, read any initial data and listen for queries.

Options are:

–d Print debugging information. A number after the "d" determines the level of messages printed.

–p Use a different port number. The default is the standard port number as listed in /etc/services.

–b Use an alternate boot file. This is optional and allows you to specify a file with a leading dash.

Any additional argument is taken as the name of the boot file. The boot file contains information about where the name server is to get its initial data. If multiple boot files are specified, only the last is used. Lines in the boot file cannot be continued on subsequent lines. The following is a small example:

```
;
;       boot file for name server
;
directory       /usr/local/domain

; type      domain                  source host/file            backup file

cache       .                                                   root.cache
primary     Berkeley.EDU            berkeley.edu.zone
primary     32.128.IN-ADDR.ARPA     ucbhosts.rev
secondary   CC.Berkeley.EDU         128.32.137.8 128.32.137.3   cc.zone.bak
secondary   6.32.128.IN-ADDR.ARPA   128.32.137.8 128.32.137.3   cc.rev.bak
primary     0.0.127.IN-ADDR.ARPA                                localhost.rev
forwarders  10.0.0.78 10.2.0.78
; slave
```

The "directory" line causes the server to change its working directory to the directory specified. This can be important for the correct processing of $INCLUDE files in primary zone files.

The "cache" line specifies that data in "root.cache" is to be placed in the backup cache. Its main use is to specify data such as locations of root domain servers. This cache is not used during normal operation, but is used as "hints" to find the current root servers. The file "root.cache" is in the same format as "berkeley.edu.zone". There can be more than one "cache" file specified. The cache files are processed in such a way as to preserve the time-to-live's of data dumped out. Data for the root nameservers is kept artificially valid if necessary.

The first "primary" line states that the file "berkeley.edu.zone" contains authoritative data for the "Berkeley.EDU" zone. The file "berkeley.edu.zone" contains data in the master file format described in RFC883. All domain names are relative to the origin, in this case, "Berkeley.EDU" (see below for a more detailed description). The second "primary" line states that the file "ucbhosts.rev" contains authoritative data for the domain "32.128.IN-ADDR.ARPA," which is used to translate addresses in network 128.32 to hostnames. Each master file should begin with

an SOA record for the zone (see below).

The first "secondary" line specifies that all authoritative data under "CC.Berkeley.EDU" is to be transferred from the name server at 128.32.137.8. If the transfer fails it will try 128.32.137.3 and continue trying the addresses, up to 10, listed on this line. The secondary copy is also authoritative for the specified domain. The first non-dotted-quad address on this line will be taken as a filename in which to backup the transfered zone. The name server will load the zone from this backup file if it exists when it boots, providing a complete copy even if the master servers are unreachable. Whenever a new copy of the domain is received by automatic zone transfer from one of the master servers, this file will be updated. The second "secondary" line states that the address-to-hostname mapping for the subnet 128.32.136 should be obtained from the same list of master servers as the previous zone.

The "forwarders" line specifies the addresses of sitewide servers that will accept recursive queries from other servers. If the boot file specifies one or more forwarders, then the server will send all queries for data not in the cache to the forwarders first. Each forwarder will be asked in turn until an answer is returned or the list is exhausted. If no answer is forthcoming from a forwarder, the server will continue as it would have without the forwarders line unless it is in "slave" mode. The forwarding facility is useful to cause a large sitewide cache to be generated on a master, and to reduce traffic over links to outside servers. It can also be used to allow servers to run that do not have access directly to the Internet, but wish to act as though they do.

The "slave" line (shown commented out) is used to put the server in slave mode. In this mode, the server will only make queries to forwarders. This option is normally used on machine that wish to run a server but for physical or administrative reasons cannot be given access to the Internet, but have access to a host that does have access.

The "sortlist" line can be used to indicate networks that are to be preferred over other, unlisted networks. Queries for host addresses from hosts on the same network as the server will receive responses with local network addresses listed first, then addresses on the sort list, then other addresses. This line is only acted on at initial startup. When reloading the nameserver with a SIGHUP, this line will be ignored.

The master file consists of control information and a list of resource records for objects in the zone of the forms:

```
$INCLUDE <filename> <opt_domain>
$ORIGIN <domain>
<domain> <opt_ttl> <opt_class> <type> <resource_record_data>
```

where *domain* is "." for root, "@" for the current origin, or a standard domain name. If *domain* is a standard domain name that does not end with ".", the current origin is appended to the domain. Domain names ending with "." are unmodified. The *opt_domain* field is used to define an origin for the data in an included file. It is equivalent to placing a $ORIGIN statement before the first line of the included file. The field is optional. Neither the *opt_domain* field nor $ORIGIN statements in the included file modify the current origin for this file. The *opt_ttl* field is an optional integer number for the time-to-live field. It defaults to zero, meaning the minimum value specified in the SOA record for the zone. The *opt_class* field is the object address type; currently only one type is supported, IN, for objects connected to the DARPA Internet. The *type* field contains one of the following tokens; the data expected in the *resource_record_data* field is in parentheses.

A	a host address (dotted quad)
NS	an authoritative name server (domain)
MX	a mail exchanger (domain)
CNAME	the canonical name for an alias (domain)

SOA marks the start of a zone of authority (domain of originating host, domain address of maintainer, a serial number and the following parameters in seconds: refresh, retry, expire and minimum TTL (see RFC883))

MB a mailbox domain name (domain)

MG a mail group member (domain)

MR a mail rename domain name (domain)

NULL a null resource record (no format or data)

WKS a well know service description (not implemented yet)

PTR a domain name pointer (domain)

HINFO host information (cpu_type OS_type)

MINFO mailbox or mail list information (request_domain error_domain)

Resource records normally end at the end of a line, but may be continued across lines between opening and closing parentheses. Comments are introduced by semicolons and continue to the end of the line.

Each master zone file should begin with an SOA record for the zone. An example SOA record is as follows:

```
@       IN      SOA     ucbvax.Berkeley.EDU. rwh.ucbvax.Berkeley.EDU. (
                                2.89    ; serial
                                10800   ; refresh
                                3600    ; retry
                                3600000; expire
                                86400 ) ; minimum
```

The SOA lists a serial number, which should be changed each time the master file is changed. Secondary servers check the serial number at intervals specified by the refresh time in seconds; if the serial number changes, a zone transfer will be done to load the new data. If a master server cannot be contacted when a refresh is due, the retry time specifies the interval at which refreshes should be attempted until successful. If a master server cannot be contacted within the interval given by the expire time, all data from the zone is discarded by secondary servers. The minimum value is the time-to-live used by records in the file with no explicit time-to-live value.

NOTES

If the empty file **/etc/use_nameserver** does not exist, *named* will not be used. Be sure to use the *touch* command to create this file when working with *named*.

The boot file directives "domain" and "suffixes" have been obsoleted by a more useful resolver based implementation of suffixing for partially qualified domain names. The prior mechanisms could fail under a number of situations, especially when then local nameserver did not have complete information.

The following signals have the specified effect when sent to the server process using the *kill*(1) command.

SIGHUP

Causes server to read named.boot and reload database.

SIGINT

Dumps current data base and cache to /usr/tmp/named_dump.db

SIGIOT

Dumps statistics data into /usr/tmp/named.stats if the server is compiled -DSTATS. Statistics data is appended to the file.

SIGSYS
Dumps the profiling data in /usr/tmp if the server is compiled with profiling (server forks, chdirs and exits).

SIGTERM
Dumps the primary and secondary database files. Used to save modified data on shutdown if the server is compiled with dynamic updating enabled.

SIGUSR1
Turns on debugging; each SIGUSR1 increments debug level. (SIGEMT on older systems without SIGUSR1)

SIGUSR2
Turns off debugging completely. (SIGFPE on older systems without SIGUSR2)

FILES

/etc/named.boot	name server configuration boot file
/etc/named.pid	the process id
/usr/tmp/named.run	debug output
/usr/tmp/named_dump.db	dump of the name server database
/usr/tmp/named.stats	nameserver statistics data
/usr/lib/conf/bind	directory containing RFC's, other information
/etc/use_nameserver	If this empty file does not exist named will not be used.

SEE ALSO
kill(1), gethostbyname(3N), signal(3c), resolver(3), resolver(5), hostname(7), RFC882, RFC883, RFC973, RFC974, *Name Server Operations Guide for BIND*

NAME

ncheck – generate names from i-numbers

SYNOPSIS

/etc/ncheck [**–i** *numbers*] [**–a**] [**–s**] [**–h** *hash_table_size*] [*filesystem*]

DESCRIPTION

Note for most normal file system maintenance, the function of *ncheck* is subsumed by *fsck*(8).

Names of directory files are followed by /.. The **–i** option reduces the report to only those files whose i-*numbers* follow. The **–a** option allows printing of the names . and .. which are ordinarily suppressed. The **–s** option reduces the report to special files and files with set-user-ID mode; it is intended to discover concealed violations of security policy. The **–h** option allows for a user specified *hash_table_size*. The default is 32719, which is adequate for striped *file systems*.

A *filesystem* may be specified.

The report is in no useful order, and probably should be sorted.

NOTES

ncheck is aware that file systems and files may be larger than two gigabytes in size.

SEE ALSO

sort(1), dcheck(8), fsck(8), icheck(8)

DIAGNOSTICS

When the filesystem structure is improper, ?? denotes the 'parent' of a parentless file, and a pathname beginning with ... denotes a loop.

NAME

newfs – construct a new file system

SYNOPSIS

/etc/newfs [**–v** | **–n**] [*mkfs_options*] *special disktype*

DESCRIPTION

newfs is a "friendly" front-end to the *mkfs*(8) program. *newfs* will look up the type of disk on which a file system is being created in the disk description file */etc/disktab*, calculate the appropriate parameters to use in calling *mkfs*, then build the file system by forking *mkfs*.

If the **–v** option is supplied, *newfs* will print out its actions, including the parameters passed to *mkfs*. The **–n** option will also cause *newfs* to display the parameters that would be passed to *mkfs*, but the actual call to *mkfs* is suppressed.

Options which may be used to override default parameters passed to *mkfs* are:

–s size The size of the file system in physical sectors.

–b block-size
The block size of the file system in bytes, e.g. 64k or 65536.

–f frag-size
The fragment size of the file system in bytes, e.g. 8k or 8192.

–M maxcontig
The maximum number of blocks that may be allocated sequentially. The default value used is 1, since ConvexOS drivers are not capable of scheduling multi-block transfers.

–R rotdelay
The minimum number of milliseconds to initiate another disk transfer on the same cylinder. It is used in determining the rotationally optimal layout for disk blocks within a file. The default used is 8 ms.

–N #sectors/track
The number of physical sectors per track on the disk.

–t #tracks/cylinder
The number of tracks per cylinder on the disk.

–c #cylinders/group
The number of cylinders per cylinder group in a file system. The default is to make an estimate based on the file system size that will allow for enough inodes (as inodes are allocated on a cylinder group basis) without creating excessive overhead.

–m free space %
The percentage of space reserved from normal users; the minimum free space threshold. The default value used is 10%.

–r revolutions/minute
The speed of the disk in revolutions per minute (normally 3600).

–i number of bytes per inode
This specifies the density of inodes in the file system. The default is to create an inode for each 2048 bytes of data space. If fewer inodes are desired, a larger number should be used; to create more inodes a smaller number should be given. The –I option could also be tried.

–I number of inodes desired
An attempt will be made by *newfs* to set up a file system with the desired number of inodes. However, the number specified most likely will not be the exact number of inodes created, because of the other requirements and restrictions that affect the

creation of an "optimal" file system on the specified device.

-E erase pattern
Causes the entire volume should be overwritten with the specified pattern before the file system is initialized. The pattern is a 32 bit hex number.

-S spare sectors/cylinder
The number of spare physical sectors per cylinder. Required by the IDC.

Note that the *block-size*, *frag-size*, and *bytes-per-inode* arguments can end with a "k"; e.g. **-b 8k** indicates a block size of 8192 bytes. The argument *special* refers to a special file used to access a disk partition, such as */dev/rda0h*. The last argument, *disktype*, refers to a type of disk whose entry can be found in the */etc/disktab* file, such as *dkd-001* or *eagle*.

FILES

/etc/disktab	for disk geometry and file system partition information
/etc/mkfs	to actually build the file system

NOTES

newfs is capable of creating file systems greater than two gigabytes in size.

SEE ALSO

disktab(5), fs(5), fsck(8), fsirand(8), mkfs(8), newst(8)

BUGS

Should figure out the type of the disk without the user's help.

NAME

newst – make a new striped file system

SYNOPSIS

newst *[-vnebfmrstcIEFSP] stripedev diskdev1 type1 [diskdev2 type2...]*
newst *-H [-nv] [stripedev...] diskdev1 type1*

DESCRIPTION

newst is a program for making "striped file systems". A striped file system is a regular ConvexOS file system which uses a striped disk partition. A striped disk partition is a logical disk partition which is interleaved over several physical disk partitions. Striped disk partitions are used to take advantage of performance improvements made possible by parallel operation of the multiple disk devices which make up striped disk partitions. ConvexOS file systems can be mounted on striped disk partitions just as with "normal" disk partitions.

The "striped file system" is supported by the *Virtual Volume Manager (VVM)*. The *VVM* driver provides redundant and non-redundant stripe capability. Redundant stripes protect against data loss from disk failure by either data mirroring or use of parity. From a user standpoint, redundant stripes will operate identically to non-redundant stripes, the difference being the action and output of some of the *VVM* utilities (i.e. the output of *getst* is enhanced for redundant stripes, and supports the *-H* option, which gives information regarding Hot Spares - see *getst(8)*).

newst has three basic jobs: (1) constructing a stripe descriptor based on the characteristics of the different partitions to be combined, (2) loading the stripe descriptor into a table in the ConvexOS kernel using *putst*(8), and (3) making a ConvexOS file system on the stripe partition using *mkfs*(8). After running *newst*, the newly striped file system can be checked with *fsck*(8) and mounted with *mount*(8).

The arguments to *newst* consist of a striped-disk device name (*/dev/rstX*) and a list of pairs of raw disk partition names and device types. For example, */dev/rda2h* and *dkd-001*, respectively. Each name/type pair specifies a disk partition which will become a part of the "striped disk partition". The device type is identical to the type specified for *newfs*(8) in the file */etc/disktab* (see *disktab*(5)). The exception being the **-H** option, in which the *stripedev* argument is optional and may be repeated, but the *diskdev type* sets may not be repeated.

newst creates a stripe descriptor entry and adds the entry to the stripe descriptor database file */etc/stripecap*. Note that all parameters must be specified in physical units.

Switch options available (redundant or normal stripes):

–v Verbose mode. Displays to *stdout* the generated stripecap entry.

–n No changes. Performs all requested actions, but does not update */etc/stripecap*, and does not invoke *putst*(8) or *mkfs*(8).

–e Entry only. Creates the stripecap entry only; does not invoke *putst*(8) or *mkfs*(8).

–b *block-size*
Specifies the file system block size to be *block-size* bytes, e.g. 64k or 65536.

–f *frag-size*
Specifies the file system fragment size to be *frag-size* bytes, e.g. 8k or 8192.

–m *maxcontig*
Specifies a value for the *mkfs maxcontig* parameter.

–r *rotdelay*
Specifies a value for the *mkfs rotdelay* parameter.

–s *# sectors/track*
Specifies the number of physical disk sectors per track.

–t *# tracks/cylinder*

Specifies the number of disk tracks per cylinder.

–c *# cylinders/group*
The number of cylinders per cylinder group in the file system. The default is to make an estimate based on the file system size that will allow for enough inodes (as inodes are allocated on a cylinder group basis) without creating excessive overhead.

–I *# inodes_desired*
An attempt will be made by *newst* to set up a striped file system with the desired number of inodes. However, the number specifed most likely will not be the exact number of inodes created, because of the other requirements and restrictions that affect the creation of an "optimal" file system on the specified devices.

–E *erase_pattern*
Forces the entire volume to be overwritten with the specified pattern before the file system is initialized. The pattern is a 32 bit hex number.

–F *percentage_free_space*
Specifies the minimum percentage of free disk space allowed. Once the file system capacity reaches this threshold, only the superuser is allowed to allocate disk blocks. The default value is 10%. This parameter is passed to *mkfs* as the "minfree" argument.

–S *# spare_sectors/cylinder*
The number of spare physical sectors per cylinder. Required by the IDC.

–P *# partitions*
Specifies the number of disk partitions per section desired. Stripes with a smaller number of partitions per section have a longer mean time to failure. The extra disk partitions are then stacked in a two-dimensional fashion, yeilding a larger number of sections. The default is to optimize the number of stripe partitions based on the block and frag size.

Switch options available (redundant stripes only):

–R Enable *VVM* data redundancy. The -R option alone implies that parity should be established on the disk device, unless only 2 disk partitions are supplied, in which case mirroring is implemented instead. The case of newst -R -P2 causes automatic mirroring. The default is no redundancy.

–H Adds the specified disk partition to the Hot Spare list. With this option the *stripedev* argument is optional and only one of the *diskdev type* sets may be present. Only *-n* and *-v* options are available in conjunction with this option. A complete disk partition must be specified, and the entire partition is considered initially available. If the *stripedev* argument(s) are present, then the disk specified is given "affinity" towards a failure on those stripes; meaning they will be chosen over other Hot Spares given a failure on a specified redundant stripe. If the disk specified is already in the Hot Spare list, then only its affinities will be updated. If no affinities are specified in the update, then all exisiting affinities for that Hot Spare are deleted. Note that although a non-redundant stripe may be selected as having affinity, no automatic reconstruction is possible, the affinity will only aid in the selection of a new partition during a manual data move.

Note that the *block-size* and *frag-size* arguments can end with a "k", e.g. "-b 8k" to indicate 8192-byte blocks. Disk partition size information is obtained from */etc/disktab* based on the disk types supplied in the argument list. Disk geometric information, such as disk block size, is obtained either from the switches, or else from the */etc/disktab* entry of the first disk device given in the argument list (see *disktab*(5)). Note once again that all size parameters must be specified in physical units, as is used in the */etc/disktab* entries.

newst uses a set of internal rules to determine how to lay out logical stripe sections on the physical devices. These heuristics are derived from performance measurements using stripe file systems. The user need not be aware of them except that file system bandwidth is optimized if only one partition per physical disk drive is used in a striped file system. (It is also currently a requirement for optimum performance that a separate controller be used for each disk drive, and a separate Multibus chassis for each controller).

For redundant stripes, additional heuristics are employed to further assure maximal performance. Several of these optimizations are:

--- Partitions per physical disk. Optimized towards no more than 1 partition from a physical disk per stripe section. In the event of multiple partitions on a physical disk, the physical disk is considered a group, such that multiple stripe sections use the group, but a single stripe section does not contain more than one partition from the group.

--- Disk type (i.e. IDC, VME, etc.). Stripe sections are optimized towards homogeneous disk types per stripe sections. Multiple disk types, if present, will reside on separate stripe sections if possible.

--- When possible, the number of controllers will be maximized across a given stripe section.

--- Sector size for a stripe will be the size of the largest sector of all disks in the stripe.

The user must have root privilege to execute this utility.

See *st*(4) or *stripecap*(5) for more information on stripes.

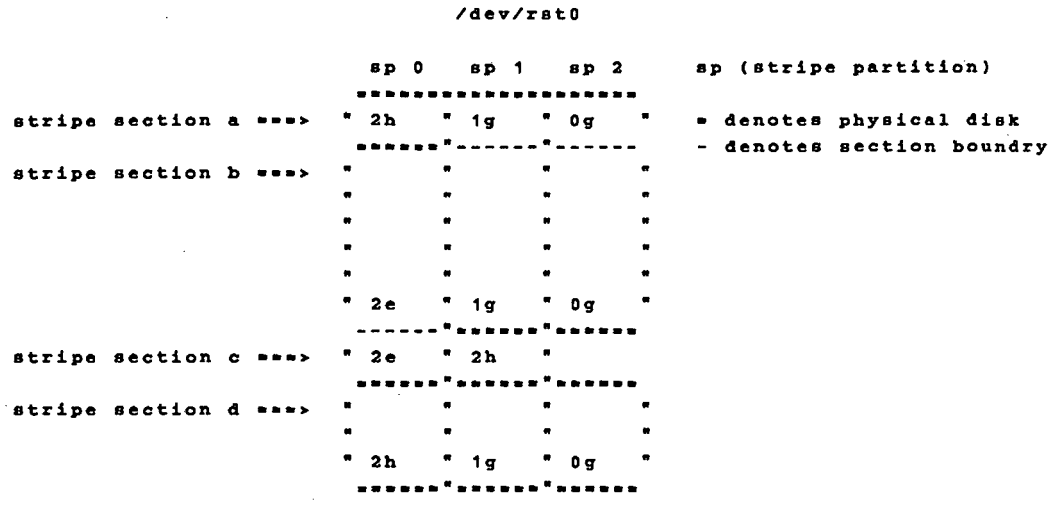
EXAMPLE

```
newst -v -R -P3 /dev/rst0 /dev/du0g dkd502 \
                /dev/du1g dkd502 \
                /dev/du2e dkd502 \
                /dev/du2h dkd502

device du2e (64, 517), type dkd-502, size 294000 Kbytes
device du2h (64, 520), type dkd-502, size 293400 Kbytes
device du1g (64, 263), type dkd-502, size 440400 Kbytes
device du0g (64, 7), type dkd-502, size 440400 Kbytes

stripe st0: redundant, sector size 2048 bytes
    section a: size 73496 Kbytes/partition, blocking factor 8 Kbytes
        partition 0: du2h (64, 520) offset 146400 Kbytes
        partition 1: du1g (64, 263) offset 0 Kbytes
        partition 2: du0g (64, 7) offset 0 Kbytes
    section b: size 220496 Kbytes/partition, blocking factor 8 Kbytes
        partition 0: du2e (64, 517) offset 0 Kbytes
        partition 1: du1g (64, 263) offset 73496 Kbytes
        partition 2: du0g (64, 7) offset 73496 Kbytes
    section c: size 73496 Kbytes/partition, blocking factor 8 Kbytes
        partition 0: du2e (64, 517) offset 220496 Kbytes
        partition 1: du2h (64, 520) offset 219896 Kbytes
    section d: size 146400 Kbytes/partition, blocking factor 8 Kbytes
        partition 0: du2h (64, 520) offset 0 Kbytes
        partition 1: du1g (64, 263) offset 294000 Kbytes
        partition 2: du0g (64, 7) offset 294000 Kbytes
/etc/putst /dev/rst0
/etc/mkfs  /dev/rst0 477140 90 7 16384 2048 2 8 32 10 60 2048 30
/etc/fsirand /dev/rst0
```

Creates a stripe of the following form:



Note there are 3 stripe partitions, as was specified by *-P3*. A "shift and split" operation was performed during the layout such that all disk space could be utilized, while maintaining maximum performance. Note that sections *a*, *b*, and *d* have 2**n+1 (where n is an integer) partitions, therefore parity error correction will be employed. Section *c* has an even number of partitions, therefore data mirroring will be employed for that section.

FILES

/dev/rd???	Raw devices for the disk driver.
/dev/rst?	Raw stripe devices.
/etc/stripecap	Database of the stripe descriptors.
/etc/disktab	Disk geometry and fs partition data.
/etc/mkfs	To actually build the file system

NOTES

It is possible to create file systems larger than two gigabytes using *newst*.

SEE ALSO

st(4), stripecap(5), getst(8), putst(8), mvst(8), rmst(8), qst(8), convst(8), mkfs(8), fsirand(8), newfs(8)

NAME

nfaccess – add access rights to a set of Notesfiles

SYNOPSIS

nfaccess access-right topic [topic ...]

DESCRIPTION

nfaccess simplifies the task of adding an access-right to many notesfiles. The function is somewhat analogous to that of *chmod*(1).

The *access-right* specifies a user, group or system and the permissions to be granted. The format is:

```
<access-right> ::= [<type>:]name=<mode>
<type>         ::= {User, user, Group, group, System, system}
<mode>         ::= {d, r, w, a, n}+
```

The *type* specification can be omitted; when it is omitted, the name is assumed to be a user. The *mode* is additive. A mode of "rw" specifies read and write. The "n" mode specifies null access. The "d" mode specifies director privileges. The "a" mode specifies answer, meaning that the user can respond to notes, but not write them.

The new access right is inserted in each specified notesfiles access list. If the user/group/system already has an entry, the old entry is replaced with the new entry.

All users are allowed to run this program. The changes are only applied to notesfiles for which the executing user is a director.

This program lives in the notesfile utility directory, typically "/usr/spool/notes/.utilities".

To automatically add specific access-rights to newly created notesfiles, the file "/usr/spool/notes/.utilities/access-template" is useful. If it exists, the file contains lines of access-rights which are added to each notesfile as it is created. If several people share the administration of the notesfile system, each can be added to the access-lists of newly created notesfiles by placing appropriate lines in this file.

EXAMPLES

```
nfaccess essick=drw /usr/spool/notes/*
nfaccess group:srg=rw this that other
nfaccess user:smith=rw mynotes
```

In the first example, user "essick" is given director/read/write access to all the notesfiles in /usr/spool/notes. Any permissions (or restrictions) he might have had before are overridden. The second example gives group "srg" read/write access to notesfiles "this", "that" and "other". In the final example, user "smith" is given read/write access to the notesfile "mynotes".

BUGS

Entries can not be removed from the access list with this program.

Entries can not merely be augmented ("just add write permission") with this program.

FILES

/etc/passwd	for the users name
/etc/group	for the users group
/usr/spool/notes	the default notesfile data base

SEE ALSO

mknf(8), notes(1), notesadm(8), nfcomment(3)
"Notesfile Reference Manual" in the *ConvexOS Tutorial Papers*

NAME

nfarchive – archive notesfiles

SYNOPSIS

nfarchive [**-#**] [**-d**] [**-m-** or **-m+**] [**-w#**] [**-f** file] topic [...]

DESCRIPTION

Nfarchive is used to expire notes that have not been modified in a certain amount of time. Archives are stored in "archive notesfiles".

The **-#** parameter is the number of days a notestring must be idle (no new responses) before being eligible for archival.

Expired notestrings are either deleted or placed in an archive. The **-d** parameter tells *nfarchive* to delete expired notestrings. If unselected, the expired notestrings are placed in an archive.

The **-m+** option specifies that only notes marked with a director message are eligible for expiration. **-m-** specifies that only notes without a director message are eligible for expiration. By default, the expiration algorithm is indifferent to a note's director message status.

The **-w#** option specified the working set size for the expired notesfiles. The specified number represents the minimum number of notes to leave in the notesfile.

The expiration threshold, working set size, expiration action, and director message requirements can all be specified as a director's option in each notesfile. Specific values override what is specified on the *nfarchive* command line. A default value specifies using the value specified on the command line.

The **-f** parameter specifies a file containing a list of notesfiles to archive. Notesfiles can also be specified on the command line.

By default, the archive of notesfile */usr/spool/notes/somenotes* is in */usr/spool/oldnotes/somenotes*. The archive of */some/other/place/somenotes* also defaults to */usr/spool/oldnotes/somenotes*. To prevent collisions of this nature, mapping between active and archive notesfiles is implemented. The file */usr/spool/notes/.utilities/net.alias/Archive-into* contains lines of the form:

active-notesfile:archive notesfile

Lines in this file beginning with '#' are comments. Notesfiles without an entry in this file are archived into the */usr/spool/oldnotes* directory with the appropriate last component.

When initially created, an archive notesfile has an access list matching its active counterpart. Currently, only directors are allowed to write in an archive notesfile. *Nfarchive* refuses to archive an archive notesfile.

The user 'notes' and the director(s) of a notesfile are the only users that are permitted to run *nfarchive* on that notesfile.

FILES

/usr/spool/notes/.utilities	where this programs lives.
/usr/spool/notes/.utilities/net.alias/Archive-into	maps active notesfiles into their archives.
/usr/spool/notes	Default notes data base
/usr/spool/oldnotes	Default archive directory

SEE ALSO

notes(1), notesadm(8)
"Notesfile Reference Manual" in the *ConvexOS Tutorial Papers*

AUTHORS

Ray Essick

NAME

nfdump, nfload – notesfile dump/load programs

SYNOPSIS

nfdump notesfile result

nfload [**-D***directory*] notesfile

DESCRIPTION

nfdump and *nfload* are used to convert notesfile data base formats.

nfdump converts the specified notesfile to a portable ASCII format and places it in the file specified by the *result* argument. If *result* is a single minus sign ("-"), the output from *nfdump* is sent to standard output.

nfload is used on the output from *nfdump* to create a new notesfile. The **-D** option specifies an alternate base directory for the notesfile. If unspecified, this defaults to "/usr/spool/notes". *nfload* reads standard input for the ASCII representation of the notesfile.

Typical use of these two programs occurs when converting an existing notesfile data base to a new format. The *nfdump* program should be compiled with the older structure definitions while *nfload* is compiled with the newer structures. The data base can then be converted with a shell script of the following nature:

```
mkdir .OLD
mv * .OLD
for i in `ls .OLD`
do
        echo $i start
        nfdump $i - | nfload -D/usr/spool/newnotes $i
done
        echo $i done
echo ALL DONE
rm -rf .OLD
```

This assumes that the old data base is in "/usr/spool/notes" and the new data base is to be placed in "/usr/spool/newnotes". After the conversion is complete, one can move the old data base from "/usr/spool/notes" to "/usr/spool/oldformat" and the new data base from "/usr/spool/newnotes" to "/usr/spool/notes".

SEE ALSO

notes(1)
"Notesfile Reference Manual" in the *ConvexOS Tutorial Papers*

NAME

nfmail – accept mail for a notesfile

SYNOPSIS

nfmail [**-s**] [**-m** mailrc] topic

DESCRIPTION

Nfmail is a mail-receiving program which takes incoming mail, parses subject and author information, and places the letter in a notesfile. Replies, marked by a "Re:" prefix in the Subject line, are placed in the notesfile as responses if a basenote with the appropriate title can be found.

The **-s** option tells *nfmail* to strip header lines from the letter before placing it in the notesfile. Normally, all header information is retained. The "Subject", and "From" header lines are never removed from the letter. The file */usr/lib/Mail.rc* defines default mail reading characteristics on many BSD systems. This file can contain lists of header lines to be ignored when presenting messages to users. *Nfmail* reads this file to determine which header lines are normally ignored. If this file is missing, *nfmail* doesn't strip any header lines.

The **-m** option specifies further files, typically ".mailrc" files, to search for header lines to ignore. More than one **-m** option can appear on the command line.

Nfmail usually appears as a mail alias in the file "/usr/lib/aliases" in lines such as:

problems: "|/usr/spool/notes/.utilities/nfmail -s problems"

BUGS

The dependence on "/usr/lib/Mail.rc" for lists of headers to be ignored should be cleaned up. Perhaps something where absence of the file means to strip all headers except the Subject and From lines.

FILES

/usr/spool/notes/.utilities/nfmail	where this program lives.
/usr/lib/aliases	Mail aliases
/usr/lib/Mail.rc	Mail configuration templates

SEE ALSO

notes(1), notesadm(8), mail(1)
"Notesfile Reference Manual" in the *ConvexOS Tutorial Papers*

NAME

nfxmit, nfrcv – notesfile networking programs

SYNOPSIS

nfxmit -d*site* [**-r**] [**-i**] [**-p**] [**-a**] [**-t** *date_spec*] [**-f** file] nf [nf2 ...]

nfrcv topic fromsystem

DESCRIPTION

nfxmit and *nfrcv* implement networking of notesfiles. The *nfxmit* program packages eligible new articles and sends them to a remote system. *nfrcv* is the receiving end of this team.

nfxmit sends the specified notesfiles to the *site* specified with the **-d** option. The **-r** option specifies that a request should be queued for the remote site to transmit updates from its copies of the notesfiles sent. The **-r** option is used only if the other site does not automatically queue updates of the notesfile.

The **-i** option indicates that *nfxmit* is to send articles to the destination that it normally would not send. These articles are those that were written there or were received at the local site from that particular site. The **-i** option is separate from the timestamp functions. Articles are first selected on the basis of the time received and criteria for having passed the destination system are applied later.

The **-p** option indicates that *nfxmit* is to send director's message and title changes of articles. If the director's message or title has changed since the last transmission to *site*, then the new title text and director's message state, on or off, are propagated. Note, both the local and remote notesfile must have *Propagate Change* set in the director option page of their respective notesfiles in order for the propagation to be completed.

The **-a** option indicates that *nfxmit* is to send articles which have originated from *news* in addition to *notes*-generated articles. Normally *nfxmit* will only send articles which are originated from within the *notes* system.

A timestamp of the last transmission of each notesfile to each system is maintained. This is used for determining the notes to send. If a different timestamp is desired, use the **-t** option. The supplied timestamp is used for this transfer and the stored timestamp is updated.

Specify **-f** *myfile* on the command line to have *nfxmit* read *myfile* for a list of notesfiles to be sent. This is useful if the number of notesfiles is too numerous to list on a single command line. The shell meta-characters *, ?, [, and] are recognized in both the *topic* parameter and the entries in *myfile*.

nfxmit uses *uux*(1) to invoke *nfrcv* on the remote system in order to process the incoming notes.

FILES

/usr/spool/notes/.utilities/net.how	Specifies connection methods between systems.
/usr/spool/notes/.utilities/net.alias	Directory containing mapping of local and remote notesfile names.
/usr/spool/notes/.sequencer/Sy:*sysname*	Sequencing timestamps for *sysname*.
/usr/spool/notes/.sequencer/Sy:*sysname*,p	Director's message and title change propagation sequencing timestamps for *sysname*.

SEE ALSO

notes(1), notesadm(8), nfcomment(3), uucp(1c)
"Notesfile Reference Manual" in the *ConvexOS Tutorial Papers*

NAME

nu – add a new user to the system

SYNOPSIS

nu [**–f** *batchfile*] [**–n** *nurcfile*]

DESCRIPTION

nu adds a new user to the system, creating directories and setting up startup files as necessary. *nu* can be entirely interactive (if invoked without the **–f** flag), prompting the user for any and all information necessary to complete the user addition.

Many questions will have default values that may be used. These defaults will be displayed in square brackets, along with the question. It is always necessary to enter a login name - there is no default for this value.

Default values to override those hardwired into the code itself can be defined in an *nurc* file, either */etc/nurc* or an alternate file given with the **–n** flag. The format of an *nurc* file is simply lines of the form *fieldname:fieldvalue* to set a default value, or *fieldname* to "turn on" a boolean-type field, such as enabling password type or age restrictions..

The possible fields which can be defined in an *nurc* file are as follows, given with their hardwired defaults, if any:

Name	Default	Description
uid	(see below)	user ID
gid	EMPTY	group ID
group	"staff"	group name (used only if gid field not given)
directory	"/mnt"	path under which home directory will be placed
protection	"0755"	home directory protection
shell	"/bin/csh"	login shell
password	(see below)	new user's initial password
username	"username"	user's full name (for the *finger* command)
office	"office"	user's office (for the *finger* command)
extension	"extension"	user's work extension
homephone	"homephone"	user's home phone number
minwks	"1"	minimum # of weeks for password aging
maxwks	"52"	maximum # of weeks for password aging
diskquota	"6000,8000,1200,1500"	*blksoft, blkhard, inodesoft, inodehard* quotas
skeleton	"/usr/skel"	directory containing files to copy into home directory
homedir	<*directory*>/<*login*>	home directory to create for this user
sgroup	"primaryg"	parent share group of new user
shares	"100"	number of shares assigned to new user

Boolean	Default	Description
typed	OFF	should this user have password typing restrictions?
aged	OFF	should this user have password aging restrictions?
quota	OFF	should home directory file system have quotas initialized?
nouidfile	OFF	ignore the contents of */etc/uidcount*?
newsgroup	OFF	is this login a new share scheduling group?
notshared	OFF	is this login a not-shared scheduling group?

The default uid will be the value contained in the */etc/uidcount* file, unless the *nouidfile* boolean is set. If this is the case, the default will be the number one greater than the highest uid already in use in the password file.

Values for uids must be greater than or equal to zero and less than 32766 (the values 32766 and 32767 are reserved for use with Share; root must always have uid 0.) Values outside this range are illegal.

If zero is specified for the minimum and maximum weeks for password aging, the user will be forced to change his password the next time he logs in (and the "age" will disappear from his entry in the restrictions file). If the minimum is greater than the maximum, only the super-user will be able to change the password. Therefore, when 0 is specified for the aging defaults, password aging is turned off.

In batch mode, the default password will be the user's login name. However, when running *nu* interactively, the *passwd* utility will be invoked, forcing creation of an initial password by the system administrator.

A sample *nurc* file might look like this:

```
shell: /bin/csh
group: swtst
directory: /swtst
protection: 0755
typed
quota
diskquota: 3000, 4096, 1000, 1500
```

The batch input file (given with the **–f** flag) has a slightly different format than the *nurc* file, but the fields that can be defined include all those listed above for *nurc* files. The batch file contains information on any number of new users, one user per line. Each line is a colon-separated list of value specifications for certain variables of the form *fieldname=fieldvalue* to define a specific value, or more simply *fieldname*, to just turn a boolean field "on." There is one additional field that must be set for each new user, namely the *login* field.

The fields which are given for a certain user affect only that user. Before processing each user in the batch file, all fields are re-initialized to the defaults hardwired in or given in an *nurc* file. Fields that will be common to a large group of new users should be placed in an *nurc* file, and only the necessary user-specific fields defined within the batch file.

Given the following example entry in a batch file, it is easy to ascertain the meaning of all of the fields:

login=jsmith:username=Julie Smith:office=107A:extension=555:typed

Nu tries to do reasonable things while it performs its function, like:

1. Only one copy at a time may run.
2. Uses *mkpasswd* to rebuild the password and restrictions databases.
3. The *chfn* and *passwd* utilities do all of the dirty work.
4. Exits with coherent status.
5. Checks for illegal data (like ":") in interactively-input fields.
6. Interrupts are disabled during critical sections of the code.

All files in a specified skeleton directory (*/usr/skel* is the default) are copied into the user's new home directory. Sample shell source files such as *.cshrc* , *.profile*, and *.login* usually appear in this skeleton directory to provide a basis for new users on the system.

The C source for the *nu* program is shipped as part of the standard distribution of utilities, in order to provide for local customization by each site's system administrator.

FILES

/usr/skel/*	skeleton initialization files (if alternate *skeleton* not given)
/etc/nurc	nu default values file (if alternate *nurc* file not given)

/etc/group	group file
/etc/passwd	password file
/etc/pwrestrict	password restrictions file
/etc/uidcount	next user ID free to use (if *nouidfile* boolean not set)
/etc/ptmp	password lock file
/etc/rtmp	restrictions lock file
/tmp/nu.lock	*nu* lock file
/etc/shares	the share data base file
/usr/convex/nu.c	*nu* C source for local modification

SEE ALSO

chfn(1), passwd(1), pl(1), rates(1), nurc(5), passwd(5), pwrestrict(5), share(5), mkpasswd(8), lim(8), remshent(8)

NAME

on, off – enable/disable logins on a tty

SYNOPSIS

on [**–n**] [**–s**] [*ttyname ...*]
off [**–n**] [**–s**] [**–f**] [*ttyname ...*]

DESCRIPTION

on and *off* are used to enable or disable logins, respectively, on specified ttys. These programs cause three events to occur. First, */etc/ttys* is rewritten. Next, the disk containing */etc/ttys* is synchronized using *fsync*(2). Finally, the *init* program (process 1) is signaled to re-read */etc/ttys* and to appropriately set up the ttys. *on/off* attempts to synchronize operations through a lock file. If this lock file cannot be acquired, a message to this effect is displayed and the process exits. This is not a true error; it just means that another *on/off* was running at the time.

The *ttynames* should be specified as they appear in */etc/ttys*; that is, the full pathname with */dev/* stripped off.

If someone is logged on to a tty when *off* is executed, */etc/ttys* is modified, but *init* is not signaled. This protocol allows the user to finish work without interruption. The **–f** flag forces a logoff.

The **–n** flag causes *on/off* to modify */etc/ttys*, but not to signal *init*.

The **–s** flag causes *init* to be signaled, with no change to */etc/ttys*.

NOTES

off is actually a link to *on*; the function that is performed is determined by the name under which the program was invoked.

These programs manipulate */etc/ttys* and *init*; they can therefore be executed only by the superuser.

FILES

/etc/ttys	tty status
/etc/utmp	tty login status
/tmp/off.lock	synchronization lock file

EXIT STATUS

If the operation was successful and no one was logged on the tty, **0** is returned; **1** is returned if someone was logged on. If the operation was not successful, various error codes from *<sysexits.h>* are returned.

SEE ALSO

fsync(2), ttys(5), utmp(5), init(8)

NAME

op – operator interface tool for giving restricted access to privileged commands

SYNOPSIS

/etc/op *mnemonic* [*arg* ...]
/etc/op -h [**-u** *username*] [*mnemonic*]

DESCRIPTION

The *op* tool provides a flexible means for system administrators to grant to any set of trusted users permission, or access, to execute certain root operations without having to give them full superuser privileges. It is a non-interactive command interpreter that places restrictions on which users are allowed to execute which privileged commands.

The functions (or mnemonics) understood by the *op* program are listed in the configurable data file */etc/op.access*, along with the meaning of each mnemonic (an exact ConvexOS command that will accomplish the desired result) and who is allowed to execute it. The restrictions can be made as tight as each site demands, as determined by the system administrator who customizes the *op.access* file.

The format of the access database file is fully described in *op.access*(5). In summary, it contains a mapping of mnemonics, or operator functions, to the full pathnames of programs that should be invoked and the arguments that are allowed, if any. The arguments to the executed program can be a combination of literal and variable arguments, and restrictions can be placed on which values are valid substitutions for the variable arguments. The *arg*s given on the *op* command line are only necessary to specify any variable arguments the mnemonic may need. While the superuser does not have access permissions checked (root can run anything), each *arg*'s validity is verified.

The following set of attributes can also be controlled for each mnemonic by the *op* program:

- the uid to set (root by default)
- the gid to set (not changed, by default)
- the group vector to set (contains only gid, by default)
- the directory to *chdir*(2) to (not changed, by default)
- the root directory to set with *chroot*(2) (not changed, by default)
- the umask to set (022 by default)
- a list of groups allowed to execute this function (none by default)
- a list of users allowed to execute this function (none by default, except the superuser)
- the range of valid arguments for the command (any value per variable argument by default)
- any number of environment variable settings (none by default).

The following options are recognized:

–h [*mnemonic*]
: This option requests help and informs the operator of the commands he or she is allowed to run and how they are to be run. The operator is not allowed to view the access file directly for security reasons. Without an argument, this option will display a list of mnemonics which the invoking user is allowed to execute. If a specific mnemonic is given, and if that mnemonic is defined in the access file and the user has permission to execute it, then a "usage" message is output, describing the valid arguments accepted by that mnemonic and the order in which they should be given (if there are any variable arguments to that mnemonic at all).

–u *username*
: Use the specified *username* when checking for permission to execute the mnemonics listed in the */etc/op.access* file. This option may only be used by the superuser and in conjunction with the **-h** option.

If the **–h** flag (for help) is not given, the *op* program verifies that the invoking user is allowed to run a particular command, validates any variable arguments, and properly sets up any of the above-specified attributes before executing the associated program. Security measures such as clearing out the vector of group permissions and clearing all but the specified environment variables are taken before the command is executed.

Attempted executions of *op* (whether successful or not) are logged via *syslog*(3). The information logged includes when and by whom *op* was executed and with which command-line arguments. Invalid users or variables are especially noted. Many messages written to the log file contain the user's login name, which will be contained within square brackets ([]).

All *op* logging is done using *syslog*'s LOG_AUTH facility. The usage information will be logged with a *syslog* level of LOG_INFO, usage errors (such as invalid or wrong number of arguments) use the LOG_NOTICE level, an unauthorized user will be noted with the LOG_WARNING level, and a failed *execve* of the command will be logged with the LOG_ERR level. The log files to which this *syslog* information is written can be customized within the */etc/syslog.conf* file; see *syslogd*(8) for information on how to configure this file.

EXAMPLES

The following line in the *op.access* file gives the user *bruce* and any member of the *opers* group permission to run weekly dumps on any of the named file systems by simply typing **op weekly** *valid_filesys_name*, where *valid_filesys_name* is one of /, **/usr**, or **/mnt**. Because no other attributes are defined for this mnemonic, the defaults are used: the dump program runs as root, but the gid, current working directory, and root directory remain unchanged. The umask is 022, and the group and environment vectors are cleared.

```
weekly      /etc/dump 0Gun $1;      users=bruce groups=opers
                                    $1=/,/usr,/mnt
```

For more examples of the access file format and usage, see the *op.access*(5) manual page.

FILES

/etc/op.access list of operator mnemonics and restrictions enforced by *op*

SEE ALSO

op.access(5), syslog(3), "Operator Interface" chapter in the *Managing ConvexOS* doumentation set

WARNINGS

It is up to each individual system administrator to assure that the programs listed within the *op.access* file are secure. It is recommended that the access file not contain shell scripts that run as root. Also, caution should be taken in including any interactive programs in the access database.

NAME

opreq_daemon – opreq daemon

SYNOPSIS

/usr/lib/opreq/opreq_daemon [-debug] [-syslog] [-timeout *n*] [-unmount]

DESCRIPTION

opreq_daemon is the main daemon for the opreq system. It is normally started from */etc/rc.local*. It receives messages from *tpdaemon* and registers the messages for use by *opreq*. *opreq_daemon* must be started before the *tpdaemon*.

OPTIONS

-debug

Enables loggin of debug messages.

-syslog

Causes log messages to go to standard output instead of *syslog*. This option is only used for debugging purposes.

-timeout *n*

Sets the timeout interval for *Unmount* and *Info* messages to *n* seconds. When *Unmount* and *Info* messages have been in the *opreq* message system for the timeout interval, these messages automatically change status to *done*. The default is 120 seconds.

-unmount

Specifying this option causes *Unmount* requests to be deleted immediately. Without this option, the *Unmount* messages are removed either manually by the operator using *opreq* or automatically after the timeout interval.

FILES

/usr/lib/opreq/share*	Shared memory files
/usr/lib/opreq/opreq_share.lockfile	Shared memory lock file
/usr/lib/opreq/opreq_daemon.lockfile	opreq_daemon lock file

SEE ALSO

opreq(1), tpdaemon(8)

NAME

pac – printer/plotter accounting information

SYNOPSIS

/usr/etc/pac [**–p price**] [**–s**] [**–r**] [**–c**] [**–g**] [**–a**] [**–G**] [**–A**] [**–U**] [acct_file]

DESCRIPTION

pac reads the printer/plotter accounting file, accumulating the number of pages (the usual case) or feet (for raster devices) of paper consumed by each user, and printing out how much each user consumed in pages or feet and dollars. Usually, statistics are reported for every user who has used any paper. The **–g** and the **–a** options report the printer statistics by group and activity, respectively. These preceding options specify the primary field used for reporting. The **–G**, **–A**, and **–U** options include group, activity, and user information as well as information specified by the primary field. These uppercase options include group, activity, and user information in the report, respectively. For example,

/usr/etc/pac –g lpd-acct

will report printer statistics by group;

/usr/etc/pac –g –A lpd-acct

will report printer statistics by group and activity;

/usr/etc/pac –G –A lpd-acct

will report printer statistics by user, group, and activity.n

The **–p** flag causes the value *price* to be used for the cost in dollars instead of the default value of 0.02.

The **–c** flag causes the output to be sorted by cost; without this flag, the output is sorted by the first field. With no other flags, the first field defaults to "user name".

The **–r** flag reverses the sorting order.

The **–s** flag causes the accounting information to be summarized on the summary accounting file; this summarization is necessary since on a busy system, the accounting file can grow by several lines per day. Summarization updates data in the summary accounting file and causes the raw accounting file to be truncated.

FILES

/usr/adm/?acct	raw accounting files
/usr/adm/?_sum	summary accounting files

SEE ALSO

idtoname(1), lpd-acct(5), sumscripts(8),
"Accounting" chapter in the *Managing ConvexOS* doumentation set.

NOTES

pac is a shell script. If the output of *pac* is not acceptable, local modifications can be made. The script can be modified in, say, *pac.local* to produce acceptable output.

BUGS

The relationship between the computed price and reality is as yet unknown.

NAME

ping – send ICMP ECHO_REQUEST packets to network hosts

SYNOPSIS

/usr/etc/ping [**–r**] [**–v**] *host* [*packetsize*] [*count*]

DESCRIPTION

The DARPA Internet is a large and complex aggregation of network hardware, connected together by gateways. Tracking a single-point hardware or software failure can often be difficult. *Ping* utilizes the ICMP protocol's mandatory ECHO_REQUEST datagram to elicit an ICMP ECHO_RESPONSE from a host or gateway. ECHO_REQUEST datagrams ("pings") have an IP and ICMP header, followed by a **struct timeval**, and then an arbitrary number of "pad" bytes used to fill out the packet. Default datagram length is 64 bytes, but this may be changed using the command-line option. Other options are:

–r Bypass the normal routing tables and send directly to a host on an attached network. If the host is not on a directly-attached network, an error is returned. This option can be used to ping a local host through an interface that has no route through it (e.g., after the interface was dropped by *routed*(8C)).

–v Verbose output. ICMP packets other than ECHO RESPONSE that are received are listed.

When using *ping* for fault isolation, it should first be run on the local host, to verify that the local network interface is up and running. Then, hosts and gateways further and further away should be "pinged". *Ping* sends one datagram per second, and prints one line of output for every ECHO_RESPONSE returned. No output is produced if there is no response. If an optional *count* is given, only that number of requests is sent. Round-trip times and packet loss statistics are computed. When all responses have been received or the program times out (with a *count* specified), or if the program is terminated with a SIGINT, a brief summary is displayed.

This program is intended for use in network testing, measurement and management. It should be used primarily for manual fault isolation. Because of the load it could impose on the network, it is unwise to use *ping* during normal operations or from automated scripts.

SEE ALSO

netstat(1c), ifconfig(8C)

NOTES

Ping is an optional product; for more information, contact your CONVEX sales representative.

NAME

portmap – DARPA port to RPC program number mapper

SYNOPSIS

/etc/portmap

DESCRIPTION

portmap is a server that converts RPC program numbers into DARPA protocol port numbers. It must be running in order to make RPC calls.

When an RPC server is started, it will tell *portmap* what port number it is listening to, and what RPC program numbers it is prepared to serve. When a client wishes to make an RPC call to a given program number, it will first contact *portmap* on the server machine to determine the port number where RPC packets should be sent.

Normally, standard RPC servers are started by *inetd*(8c), so *portmap* must be started before *inetd* is invoked.

SEE ALSO

rpcinfo(8), inetd(8C)

BUGS

If *portmap* crashes, all servers must be restarted.

NAME

preen – run fsck in parallel over normally mounted disks

SYNOPSIS

/etc/preen [–L <concurrency limit>] [<fsck options>]

DESCRIPTION

Preen invokes *fsck*(8) to check disk partitions in a highly optimal manner, much in the same way that the **–p** option in *fsck* works. It differs from *fsck* in that it uses the base names of the disks to determine which are the same physical drive, and actually tries to keep all drives busy. *Fsck -p* merely runs groups of "passes", and with odd configurations tends to leave the arms idle a lot.

Status returned by *preen* matches (as closely as possible) that of *fsck*.

One of the biggest advantages of using *preen* is that the system administrator no longer needs to worry about configuring "pass numbers" in the file system table.

The following flag is interpreted by *preen*.

–L Uses the number specified immediately after the flag as the limit to the number of fsck's preen will try to run concurrently.

Preen also will recognize and pass on flags defined for *fsck*(8).

FILES

/etc/fstab
/etc/rc

SEE ALSO

fsck(8)

WARNINGS

Flags given to preen to be passed on to fsck will be passed on to all fscks. This means that the -b fsck option should not be used with preen.

NAME

pstat – print system facts

SYNOPSIS

/etc/pstat –aivpfstT [**–P***pid*] [**–C***name*] [**–S** [*uid*]] [*system*] [*corefile*]

DESCRIPTION

pstat interprets the contents of certain system tables. If *corefile* is given, the tables are sought there, otherwise in */dev/mem*. The required namelist is taken from */vmunix* unless *system* is specified. Options are

–a Under **–p**, describe all process slots rather than just active ones.

–i Print the inode table including the associated vnode entries with these headings:

ILOC The core location of this table entry.

IFLAG Miscellaneous inode state variables encoded thus:

- A inode access time must be corrected
- C inode has been changed
- L inode is locked
- R inode is being referenced
- T inode contains a pure text prototype
- U update time (*fs*(5)) must be corrected
- W wanted by another process (L flag is on)

IDEVICE
Major and minor device number of file system in which this inode resides

INO I-number within this device

MODE Mode bits in octal, see *chmod*(2).

NLK Number of links to this inode.

UID User ID of owner.

SIZE/DEV
Number of bytes in an ordinary file, or major and minor device of special file.

VFLAG
Miscellaneous vnode state variables encoded thus:

- R root of its file system
- S shared lock applied
- E exclusive lock applied
- O a virtual memory object is associated with this vnode
- T vnode is a pure text prototype
- Z process is waiting for a shared or exclusive lock

CNT Number of open file table entries for this inode.

SHC Reference count of shared locks on the vnode.

EXC Reference count of exclusive locks on the vnode (this may be > 1 if, for example, a file descriptor is inherited across a fork).

TYPE Vnode file type, either VCON (no type), VREG (regular), VDIR (directory), VBLK (block device), VCHR (character device), VLNK (symbolic link), VSOC (socket), or VBAD (bad).

–p Print process table for active processes with these headings:

LOC The core location of this table entry.

S Run state encoded thus:

- 0 no process
- 1 runnable
- 2 being created
- 3 being terminated
- 4 awaiting an event
- 6 stopped awaiting a signal

FLAG Miscellaneous state variables, or-ed together (hexadecimal):

00000001	process is loaded
00000002	a system process (scheduler or page-out daemon)
00000008	user area and page tables are loaded
00000010	process is being traced
00000020	used in tracing
00000040	process is locked in core
00000200	restore old sigmask after taking signal
00000400	process is exiting
00000800	doing physical i/o
00001000	process resulted from a *vfork*(2) which is not yet complete
00002000	another flag for *vfork*(2)
00004000	process has no virtual memory, as it is a parent in the context of *vfork*(2)
00008000	process was stopped while sleeping
00010000	process wants all threads to stop
00020000	process wants all threads to join
00040000	process is in a sleep which will timeout.
00100000	process used 4.1BSD compatibility mode signal primitives, no system calls will restart.
00200000	process is owed a profiling tick.
00800000	process is a login process.
02000000	process doing asynchronous i/o
04000000	waiting on async i/o completion
10000000	track lock
20000000	parent inherits signals
40000000	release process on debug close
80000000	record-locking has been done

PRI Scheduling priority, see *setpriority*(2).
SIG Signals received (signals 1-32 coded in bits 0-31),
UID Real user ID.
SLP Amount of time process has been blocked.
TIM Time resident in seconds; times over 65535 coded as 65535.
CPU Weighted integral of CPU time, for scheduler.
NI Nice level, see *setpriority*(2).
PGRP Process number of root of process group (the opener of the controlling terminal).
PID The process ID number.
PPID The process ID of parent process.
UADDR
The virtual address of the processes user structure.
RSS Resident set size – the number of physical page frames allocated to this process.
SRSS RSS on the swap device (0 if never swapped).
SIZE Virtual size of process image (data+stack) in multiples of 4096 bytes.
WCHAN
Wait channel number of a waiting process.

–v Prints information about each process's virtual memory. First the LOC, S, FLAG, PRI, PID, PPID, UADDR, RSS, SRSS, SIZE, and WCHAN fields are printed from the process table (as described above). Then information from the process's vspace structure is printed, followed by a line for each region attached to the vspace.

For each vspace, the following information is printed:

LOC The address of the vspace structure.
FLAGS Flags about the vspace structure, as follows:
- L vspace is locked
- S vspace is locked shared
- E vspace is locked exclusively
- W vspace is wanted by another
- R regions virtually present only
- N vspace is nonswappable
- P vspace is prepaged

USE The use count for the vspace.
SHARE
The number of threads holding shared locks on the vspace.
NREG The number of regions attached to the vspace.
FLOW Front hand of the page daemon clock that scans vspaces for unreferenced pages.
BLOW Back hand of the page daemon clock that scans vspaces for unreferenced pages.

For each region, the following information is printed:

LOC The address of the region structure.
FLAGS Various flags about the region structure, as follows:
- B if the region is marked BUSY
- W if the region is marked WANTED
- I if the region immediately migrates pages to the backing object

PROT The protection property of the region, as follows:
- r the pages are readable
- w the pages are writable
- x the pages are executable

SHARE
The region sharing parameters (see /usr/include/sys/mman.h), as follows:
- F MAP_FILE is set
- A MAP_ANON is set
- D MAP_DEVICE is set
- P MAP_PRIVATE is set
- I MAP_INHERIT is set
- d the region may grow downward
- u the region may grow upward
- a the region grows automatically without system call intervention
- s the region is marked as being sequentially accessed

VLOW The lowest virtual page number in the region.
VSIZE The number of virtual pages in the region.
RSS The resident set size of the region.
VLIMIT
The limit on region growth in pages.
OBJECT
The address of the paging object.
The address of the backing object.

-f Print the open file table with these headings:

LOC The core location of this table entry.

TYPE The type of object the file table entry points to.
FLG Miscellaneous state variables encoded thus:
- R open for reading
- W open for writing
- A open for appending

S shared lock present
X exclusive lock present
I signal pgrp when data ready

CNT Number of processes that know this open file.

DATA The location of the vnode table entry or socket for this file.

OFFSET
The file offset (see *lseek*(2)), or the core address of the associated socket structure.

-s Prints several pieces of information about swap space usage. The first line shows the number of kilobytes used and free as well as the number of kilobytes that have been lost from the resource map. Starting on the second line, a list of various size ranges of swap areas is shown, and the number of kilobytes of free space in each range is printed. Below that are the numbers of swap space allocations which succeeded, were truncated, and failed since ConvexOS was booted. Finally, the number of kilobytes of memory used to keep track of swap space allocations is printed.

-t Prints information about each thread of a process. This information is implementation dependent. Consult the include file <sys/thread.h> for more information.

-T prints the number of used and free slots in several system tables and is useful for checking to see how full system tables have become if the system is under a heavy load.

-P*pid* Restrict output to the process given by *pid*. Meaningful only when used with one of **-p**, **-v**, or **-t**.

-C*name*
Restrict output to those processes named *name*. Meaningful only when used with one of **-p**, **-v**, or **-t**.

-S*uid* Display information pertaining to the share scheduler. If *uid* is present then output is restricted to processes with a share uid of *uid*. The output format includes the process name, process structure address (PROC), uid (UID), share group uid (SHGUID), scheduler cpu usage (CPU), and process id (PID). This flag is only useful if the system is running the share scheduler. This flag is generally used to track down processes that are incorrectly attached to the root lnode. Processes running as the root lnode will show up with UID == SHGUID == 0.

FILES

/vmunix	kernel name list
/dev/mem	kernel data values
*/lib/kernsyms/symdata_**	kernel symbol addresses

SEE ALSO

ps(1), stat(2), fstat(8), fs(5)

BUGS

It would be very useful if the system recorded "maximum occupancy" on the tables reported by **-T;** even more useful if these tables were dynamically allocated.

NAME

putst – manipulate kernel databases for filesystem stripes

SYNOPSIS

putst *[-vnap] [stripedev ...]*

DESCRIPTION

putst initializes a stripe description table in the ConvexOS kernel for a "striped disk" partition or, with the *-p* option manually initiates the parity checking process on a stripe. Striped disk partitions are logical disk partitions which are interleaved over several physical disk partitions. Stripes are used to take advantage of performance improvements made possible by parallel operation of the mutltiple disk devices which make up the striped disk partition. ConvexOS file systems can be mounted on stripes just as with "normal" disk partitions.

The "striped filesystem" is supported by the *Virtual Volume Manager (VVM).* The *VVM* driver provides redundant and non-redundant stripe capability. Either type are handled appropriately by *putst.*

putst must be used before the striped partition can be initialized with *newfs*(8) or *mkfs*(8), or mounted as a ConvexOS file system, or otherwise accessed for reading and writing.

Normally *putst* is invoked at reboot time by the startup script */etc/rc* to load stripe tables from the stripe database file */etc/stripecap*. For each stripe name given the actual file system referenced will be */dev/rst?*. *putst* searches for an entry named *st?* in */etc/stripecap*, and uses the parameters defined by that entry to initialize the striped file system information in the ConvexOS kernel.

Switch options available:

-a Automatically loads stripe tables of the form */dev/rst?* by using all valid entries of the form **st?** in */etc/stripecap* (if they exist). No file name arguments need be given.

-v Verbose mode. Displays on *stdout* information being loaded into the stripe table. In this mode, if a single parity request is also specified (**-p** option without **-a**), then information on stripe parity data is written to syslog(3) with the LOG_DAEMON facility.

-n No changes. Performs all requested actions, but does not actually update the kernel stripe table. This option can be useful for checking the syntax of stripecap entries.

-p Perform manual parity check. With this option stripe tables are *not loaded* into the kernel, but the parity checking process is started for each stripe specified which has already been loaded. Note that without this option the parity check process is started by default for all newly loaded stripe tables.

-N No parity check. This option is used in */etc/.initrc* to suppress the initiation of parity checking while the system is operating in single user mode. The parity check is normally initiated when going to multiuser mode by a *putst -ap* command from */etc/rc.std*.

The user must have root privilege to execute this utility.

Entries in */etc/stripecap* are created with the *newst*(8) utility.

FILES

/etc/stripecap Database of the stripe descriptors.

SEE ALSO

st(4), stripecap(5), newst(8), getst(8), mvst(8), rmst(8), qst(8), convst(8)

NAME

qst – print disk device information from the stripecap file

SYNOPSIS

qst *diskdev/partition... /*

DESCRIPTION

qst retrieves stripe information from the stripecap file for each disk device given as an argument. A summary of which stripes have each listed device as a component is written to *stdout*. The disk device may be specified with or without the */dev/* pathname and may have its name preceded by *r* , but in all cases */dev/r* will be the actual disk device referenced. If a partition is specified in the device name, then only that parition will be searched for in */etc/stripecap*. If no partition is specified, then all partitions of that device will be referenced.

The user must have root privilege to execute this utility.

See *st*(4) or *newst*(8) for more information on stripes.

EXAMPLE

This utility is especially useful under conditions such as the following scenario:

I have just lost /dev/du2 that was a part of several stripes. Loosing this disk also caused the system to crash. While the system was down, I was able to replace /dev/du2 and boot single user. Of course, all the non-redundant striped filesystems that used partitions from /dev/du2 are lost. All the redundant striped filesystems must have their data reconstructed before they can be used again. Without the qst utility, I would have to search through the stripecap file for all redundant stripes with a component disk having major number = 64 and minor number between 513 and 520.

The command:

qst /dev/du2

Would return:

/dev/du2h is used in /dev/st2 (redundant)
/dev/du2g is used in /dev/st4 (non-redundant)

The user could then manually reconstruct the redundant disk, using the *mvst(8)* utility.

FILES

/etc/stripecap	Database of the stripe descriptors.

SEE ALSO

st(4), stripecap(5), newst(8), getst(8), putst(8), mvst(8), rmst(8), convst(8)

NAME

quot – summarize file system ownership

SYNOPSIS

/usr/etc/quot [**–acfhnv**] [*filesystem*]

DESCRIPTION

quot displays the number of blocks (1024 bytes) in the named *filesystem* currently owned by each user.

OPTIONS

–a Generate are report for all mounted file systems.

–c Display three columns giving file size in blocks, number of files of that size, and cumulative total of blocks in that size or smaller file.

–f Display count of number of files as well as space owned by each user.

–h Estimate the number of blocks in the file — this doesn't account for files with holes in them.

–n Run the pipeline **ncheck filesystem | sort +0n | quot –n filesystem** to produce a list of all files and their owners.

–v Display three columns containing the number of blocks not accessed in the last 30, 60, and 90 days.

FILES

/etc/mtab	mounted file systems
/etc/passwd	to get user names

SEE ALSO

ls(1), du(1), df(1), find(1)

NAME

quotacheck – check file system quota consistency

SYNOPSIS

/usr/etc/quotacheck [**–v**] [**–p**] filesystem...
/usr/etc/quotacheck [**–v**] [**–p**] **–a**

DESCRIPTION

quotacheck examines each file system, builds a table of current disk usage, and compares this table against that stored in the disk quota file for the file system. If any inconsistencies are detected, both the quota file and the current system copy of the incorrect quotas are updated (the latter only occurs if an active file system is checked).

quotacheck expects each file system to be checked to have a quota file named *quotas* in the root directory. If none is present, *quotacheck* will ignore the file system.

quotacheck is normally run at boot time from the */etc/rc.local* file, see *rc*(8), before enabling disk quotas with *quotaon*(8).

quotacheck accesses the raw device in calculating the actual disk usage for each user. Thus, the file systems checked should be quiescent while *quotacheck* is running.

OPTIONS

–v Print a list of all file and block usage counts that are updated. The list consists of lines containing the disk partition name (if **-a** or **-p** was given), a user name, the old and new file count (if different), and the old and new block count (if different). File or block counts that have not changed will not be shown.

–p In this case *quotacheck* reads the table **/etc/fstab** to determine which file systems to check. It uses the information there to inspect groups of disks in parallel taking maximum advantage of i/o overlap to check the file systems as quickly as possible.

–a check all the file systems indicated in */etc/fstab* to be read-write with disk quotas.

FILES

quotas	quota file at the file system root
/etc/mtab	mounted file systems
/etc/fstab	default file systems

SEE ALSO

quota(1), quotactl(2), quotaon(8), repquota(8), fstab(5)

NAME

quotaon, quotaoff – turn file system quotas on and off

SYNOPSIS

/usr/etc/quotaon [**–v**] *filsys...*

/usr/etc/quotaon [**–v**] **–a**

/usr/etc/quotaoff [**–v**] *filsys...*

/usr/etc/quotaoff [**–v**] **–a**

DESCRIPTION OF QUOTAON

quotaon announces to the system that disk quotas should be enabled on one or more file systems. The file systems specified must be mounted at the time. The file system quota files must be present in the root directory of the specified file system and be named *quotas*.

OPTIONS TO QUOTAON

–v display a message for each file system where quotas are turned on.

–a all file systems in */etc/fstab* marked read-write with quotas will have their quotas turned on. This is normally used at boot time to enable quotas.

DESCRIPTION OF QUOTAOFF

quotaoff announces to the system that file systems specified should have any disk quotas turned off.

OPTIONS TO QUOTAOFF

–v display a message for each file system affected.

–a force all file systems in */etc/fstab* to have their quotas disabled.

These commands update the status field of devices located in */etc/mtab* to indicate when quotas are on or off for each file system.

FILES

quotas	quota file at the file system root
/etc/mtab	mounted file systems
/etc/fstab	default file systems

SEE ALSO

quotactl(2), mtab(5), fstab(5)

NAME

rc – command script for auto-reboot and daemons

SYNOPSIS

/etc/rc
/etc/rc.local

DESCRIPTION

rc is the command script which controls the automatic reboot and *rc.local* is the script holding commands which are pertinent only to a specific site.

When an automatic reboot is in progress, *rc* is invoked with the argument *autoboot* and runs a *fsck* with option **–p** to "preen" all the disks of minor inconsistencies resulting from the last system shutdown and to check for serious inconsistencies caused by hardware or software failure. If this auto-check and repair succeeds, then the second part of *rc* is run.

The second part of *rc*, which is run after a auto-reboot succeeds and also if *rc* is invoked when a single user shell terminates (see *init*(8)), starts all the daemons on the system, preserves editor files, configures the loopback device, and performs various other functions related to system startup. *rc.local* is executed immediately before any other commands after a successful *fsck*. Normally, the first command placed in the *rc.local* file defines the machine's name, using *hostname*(1).

SEE ALSO

init(8), reboot(8)

NAME

rdump – dump file systems across the network

SYNOPSIS

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

DESCRIPTION

rdump copies to magnetic tape all files changed in the *filesystem* after a certain date. *rdump* uses the same format and keys as *dump*(8); in fact, *rdump* is simply an extension of *dump*(8). To use *rdump*, The *f* key is used with an argument of the form *user@machine:device*. user@ is optional and runs the remote daemon as the specified user. This is similar to the -l option for rlogin. If the *f* key is not specified, the default is dumphost:/dev/rmt8.

rdump creates a remote server, */etc/rmt*, on the client machine to access the tape device.

FILES

~/.rhosts
: This file exists on the server host with *rmt*(8C). It contains a list of the Internet client host names having access to the server host.

dumphost:/dev/rmt8
: The default remote tape drive to use if the *f* key is not specified.

SEE ALSO

rhosts(5), dump(8), rmt(8C), rrestore(8C)

DIAGNOSTICS

Same as *dump*(8) with a few extra diagnostics related to the network.

RESTRICTIONS

The 100ips option of *dump*(8), flag **I**, is not currently supported on remote devices.

BUGS

Since *rmt* was modified to perform write commands to tape in an asynchronous manner for performance improvements, error messages are not received by *rdump* until the subsequent write command is issued. This has a possible negative side effect if the very last write to tape fails. In this case, the error response will not be received until the close command is issued, and you will not receive the prompt to abort or try again. You will, however, see an error message indicating that something went wrong.

NOTES

Only the superuser may perform an *rdump*.

rdump is an optional product; for more information, contact your CONVEX sales representative.

NAME

reboot – ConvexOS bootstrapping procedures

SYNOPSIS

/etc/reboot [**–n**] [**–q**]

DESCRIPTION

ConvexOS is started by the Service Processor Unit (SPU). Since the system is not reenterable, it is necessary to read it in from disk each time it is to be bootstrapped.

Rebooting a running system.

When ConvexOS is running and a reboot is desired, *shutdown*(8) is normally used. If there are no users then **/etc/reboot** can be used. Reboot causes the disks to be synced, and then a multi-user reboot (as described below) is initiated. This causes a system to be booted and an automatic disk check to be performed. If all this succeeds without incident, the system is then brought up for many users.

Options to reboot are:

–n option avoids the sync. It can be used if a disk or the processor is on fire.

–q reboots quickly and ungracefully, without shutting down running processes first.

Power fail and crash recovery.

Normally, the system will reboot itself at power-up or after crashes. Provided the auto-restart is enabled in the machine software front panel and the front panel key switch is in the "secure" position, an automatic consistency check of the file systems will be performed. Unless this fails, the system will resume multi-user operations.

In an emergency, the bootstrap methods described in the paper "Installing ConvexOS" can be used to boot from a distribution tape. See the *Managing ConvexOS* doumentation set for information on bootstrapping ConvexOS.

FILES

/vmunix system code

SEE ALSO

fsck(8), init(8), rc(8), shutdown(8), halt(8), newfs(8)

NAME

renice – alter priority of running processes

SYNOPSIS

/etc/renice priority [[**-p**] pid ...] [[**-g**] pgrp ...] [[**-u**] user ...]

DESCRIPTION

renice alters the scheduling priority of one or more running processes. The *who* parameters are interpreted as process IDs, process group IDs, or user names. *renice*'ing a process group causes all processes in the process group to have their scheduling priority altered. *renice*'ing a user causes all processes owned by the user to have their scheduling priority altered. By default, the processes to be affected are specified by their process IDs. To force *who* parameters to be interpreted as process group IDs, a **-g** may be specified. To force the *who* parameters to be interpreted as user names, a **-u** may be given. Supplying **-p** will reset *who* interpretation to be (the default) process IDs. For example,

/etc/renice +1 987 -u daemon root -p 32

would change the priority of process IDs 987 and 32, and all processes owned by users *daemon* and *root*.

Users other than the superuser may only alter the priority of processes they own, and can only monotonically increase their "nice value" within the range 0 to PRIO_MIN (64). (This prevents overriding administrative fiats.) The superuser may alter the priority of any process and set the priority to any value in the range PRIO_MAX (-64) to PRIO_MIN (64). Useful priorities are: 64 (the affected processes will run only when nothing else in the system wants to), 0 (the "base" scheduling priority), anything negative (to make things go very fast). However, when giving a task a very negative priority, other processes will receive a smaller portion of the cpu which may result in degradation of overall system performance.

FILES

/etc/passwd to map user names to user IDs

SEE ALSO

getpriority(2)

BUGS

If you make the priority very negative, then the process cannot be interrupted. To regain control you make the priority greater than zero. Non-superusers cannot increase scheduling priorities of their own processes, even if they were the ones that decreased the priorities in the first place.

NAME

repquota – summarize quotas for a file system

SYNOPSIS

/usr/etc/repquota [**–v**] *filesys...*
/usr/etc/repquota [**–v**] **–a**

DESCRIPTION

repquota prints a summary of the disc usage and quotas for the specified file systems. For each user the current number files and amount of space (in kilobytes) is printed, along with any quotas created with *edquota*(8).

OPTIONS

–v report all quotas, even if there is no usage.

Only the superuser may view quotas which are not their own.

–a report on all file systems indicated in */etc/fstab* to be read-write with quotas.

FILES

quotas	quota file at the file system root
/etc/fstab	default file systems

SEE ALSO

quota(1), quotactl(2), quotacheck(8), quotaon(8), edquota(8)

NOTES

Non-superusers should use "*quota -v*" for up-to-date calculations.

NAME

restore – incremental file system restore

SYNOPSIS

/etc/restore key [name ...]

DESCRIPTION

restore reads tapes dumped with the *dump*(8) command. Its actions are controlled by the *key* argument. The *key* is a string of characters containing, at most one, function letter and possibly one or more function modifiers. Other arguments to the command are file or directory names specifying the files that are to be restored. Unless the **h** key is specified (see below), the appearance of a directory name refers to the files and recursively subdirectories of that directory.

The function portion of the key is specified by one of the following letters:

r The tape is read and loaded into the current directory. This should not be done lightly; the **r** key should only be used to restore a complete dump tape onto a clear file system or to restore an incremental dump tape after a full level zero restore. Thus:

```
/etc/newfs /dev/rda0g eagle
/etc/mount /dev/rda0g /mnt
cd /mnt
restore r
```

is a typical sequence to restore a complete dump. Another *restore* can be done to get an incremental dump in on top of this. Note that *restore* leaves a file *restoresymtab* in the root directory to pass information between incremental restore passes. This file should be removed when the last incremental tape has been restored. It should be obvious that the file system which is being restored from tape must not be larger than the disk file system into which it is being restored.

A *dump*(8), followed by a *newfs*(8), and a *restore* is used to change the size of a file system.

R *restore* requests a particular tape of a multi-volume set on which to restart a full restore (see the **r** key above). This allows *restore* to be interrupted and then restarted.

x The named files are extracted from the tape. If the named file matches a directory whose contents had been written onto the tape, and the **h** key is not specified, the directory is recursively extracted. The owner, modification time, and mode are restored if possible. If no file argument is given, then the root directory is extracted, which results in the entire content of the tape being extracted, unless the **h** key has been specified.

t The names of the specified files are listed if they occur on the tape. If no file argument is given, then the root directory is listed, which results in the entire content of the tape being listed, unless the **h** key has been specified. Note that the **t** key replaces the function of the old *dumpdir* program.

V The dump tape(s) are verified for consistency. This key performs the same operation as the **x** key, except that nothing is written to disk. This option verifies that the tape can be read without error and the dump file format is correct. It does not verify the correctness of the directories and files contained in the dump file.

i This mode allows interactive restoration of files from a dump tape. After reading in the directory information from the tape, *restore* provides a shell-like interface that allows the user to move around the directory tree selecting files to be extracted. The available commands are given below; for those commands that require an argument, the default is the current directory.

ls [arg] – List the current or specified directory. Entries that are directories are appended with a /. Entries that have been marked for extraction are prepended with a *. If

the verbose key is set the inode number of each entry is also listed.

cd arg – Change the current working directory to the specified argument.

pwd – Print the full pathname of the current working directory.

add [arg] – The current directory or specified argument is added to the list of files to be extracted. If a directory is specified, then it and all of its descendants are added to the extraction list (unless the **h** key is specified on the command line). Files that are on the extraction list are prepended with a *. when they are listed by **ls**.

delete [arg] – The current directory or specified argument is deleted from the list of files to be extracted. If a directory is specified, then it and all of its descendants are deleted from the extraction list (unless the **h** key is specified on the command line). The most expedient way to extract most of the files from a directory is to add the directory to the extraction list and then delete the files that are not needed.

extract – All of the files that are on the extraction list are extracted from the dump tape. *Restore* will ask which volume the user wishes to mount. The fastest way to extract a few files is to start with the last volume, and work towards the first volume.

verbose – The sense of the **v** key is toggled. When set, the verbose key causes the **ls** command to list the inode numbers of all entries. It also causes *restore* to print out information about each file as it is extracted.

help – List a summary of the available commands.

quit – *Restore* immediately exits, even if the extraction list is not empty.

The following characters may be used in addition to the letter that selects the function desired:

v Normally, *restore* does its work silently. The **v** (verbose) key causes *restore* to type the name of each file it treats preceded by its file type.

c *restore* converts old *(4.1-BSD)* directories to new *(4.3+BSD)* directory format.

d *restore* prints a great deal of debugging information on the controlling terminal.

f The next argument to *restore* is used as the name of the archive instead of */dev/rmt8*. If the name of the file is –, *restore* reads from standard input. Thus, *dump*(8) and *restore* can be used in a pipeline to dump and restore a file system with the command:

```
dump 0f - /usr | (cd /mnt; restore xf -)
```

y *restore* will not ask whether it should abort the restore if gets a tape error. It will always try to skip over the bad tape block(s) and continue as best it can.

s *n* *restore* skips to the *n*-th file on the tape before doing the restore. Tape files are numbered starting from **1** for the first file.

m *restore* will extract by inode numbers rather than by filename. This is useful if only a few files are being extracted, and you want to avoid regenerating the complete pathname to the file.

h *restore* extracts the actual directory, rather than the files that it references. This prevents hierarchical restoration of complete subtrees from the tape.

b The next argument to *restore* is used as the blocking factor for tape records. This number indicates how many K-bytes are in a tape record. The default is 5.

G Defaults are set for a GCR (6250 bpi) *restore*. The tape unit is set to */dev/rmt16*, the density to 6250, and the tape blocking factor to 50.

I Puts the tape drive into 100 ips streaming mode. The default is 50 ips.

It should be stressed that the files that are extracted from a dumped tape will retain the entire path name with which they were dumped. For example, create a tape by *dump*ing everything from a current working directory that contains the subdirectory *lib*, and its subdirectory *tmac*. Then perform the following steps :

```
% mkdir tmp tmp/tmac      # make directory to put files under
% cd tmp/tmac             # get there
% /etc/restore Gi         # interactive mode
restore> cd lib/tmac      # get to where the files are that we want
restore> add              # add all of them to the extraction list
restore> ls               # list and verify that all files are starred
restore> extract          # get the starred files from the tape
restore> quit             # leave restore
```

This will extract the files on the tape from *lib/tmac/** and place them **under** the current working directory, into *tmp/tmac/lib/tmac/**. The files will **not** go directly under *tmp/tmac*, because the entire path saved on the tape is restored and appended to the *end* of the path of the current working directory.

DIAGNOSTICS

Complaint about bad key characters.

Complaint if *restore* gets a read error. If **y** has been specified, or the user responds **y**, *restore* will attempt to continue the restore.

If the dump extends over more than one tape, *restore* will ask the user to change tapes. If the **x** or **i** key has been specified, *restore* will also ask which volume the user wishes to mount. The fastest way to extract a few files is to start with the last volume, and work towards the first volume.

There are numerous consistency checks that can be listed by *restore*. Most checks are self-explanatory or can "never happen". Common errors are:

Converting to new file system format
A dump tape created from the old file system has been loaded. It is automatically converted to the new file system format.

<filename>: **not found on tape**
The specified filename was listed in the tape directory, but was not found on the tape. This is caused by tape-read errors while looking for the file, and from using a dump tape created on an active file system.

expected next file <inumber> got <inumber>
A file that was not listed in the directory showed up. This can occur when using a dump tape created on an active file system.

Incremental tape too low
When doing incremental restore, a tape that was written before the previous incremental tape, or that has too low an incremental level, has been loaded.

Incremental tape too high
When doing incremental restore, a tape that does not begin its coverage where the previous incremental tape left off, or that has too high an incremental level has been loaded.

Tape read error while restoring <filename>

Tape read error while skipping over inode <inumber>

Tape read error while trying to resynchronize
A tape read error has occurred. If a filename is specified, then its contents are probably partially wrong. If an inode is being skipped or the tape is trying to resynchronize, then no extracted files have been corrupted, though files may not be found on the tape.

resync restore, skipped <num> **blocks**
After a tape read error, *restore* may have to resynchronize itself. This message lists the number of blocks that were skipped over.

FILES

/dev/rmt8	the default tape drive
/tmp/rstdir*	file containing directories on the tape.
/tmp/rstmode*	owner, mode, and time stamps for directories.
./restoresymtab	information passed between incremental restores.

NOTES

resstore understands that some files may be larger than two gigabytes in size, and is capable of restoring them, and is capable of reading dump images from the file system that are greater than two gigabytes in size.

restore is still capable of restoring dumps made before files greater than two gigabytes were possible.

SEE ALSO

mtio(4), dump(8), mkfs(8), mount(8), newfs(8), newst(8)

BUGS

restore can get confused when doing incremental restores from dump tapes that were made on active file systems.

A level zero dump must be done after a full *restore*. Because *restore* runs in user code, it has no control over inode allocation; thus, a full *restore* must be done to get a new set of directories reflecting the new inode numbering, even though the contents of the files are unchanged.

If a non-standard blocking factor is used to dump a file system, it must also be used during a *restore*. *restore* should determine the blocking factor by itself.

NAME

rexecd, in.rexecd – remote execution server

SYNOPSIS

/etc/rexecd

DESCRIPTION

rexecd is the server for the *rexec*(3X) routine. The server provides remote execution facilities with authentication based on user names and encrypted passwords. It is started automatically as needed by *inetd*(8C) and initiates the following protocol:

1) The server reads characters from the socket up to a null ("\0") byte. The resultant string is interpreted as an ASCII number, base 10.

2) If the number received in step 1 is non-zero, it is interpreted as the port number of a secondary stream to be used for the **stderr**. A second connection is then created to the specified port on the client's machine.

3) A null terminated user name of at most 16 characters is retrieved on the initial socket.

4) A null terminated, unencrypted password of at most 16 characters is retrieved on the initial socket.

5) A null terminated command to be passed to a shell is retrieved on the initial socket. The length of the command is limited by the upper bound on the size of the system's argument list.

6) *rexecd* then validates the user as is done at login time and, if the authentication was successful, changes to the user's home directory, and establishes the user and group protections of the user. If any of these steps fail the connection is aborted with a diagnostic message returned.

7) A null byte is returned on the initial socket and the command line is passed to the normal login shell of the user. The shell inherits the network connections established by *rexecd*.

DIAGNOSTICS

All diagnostic messages are returned on the connection associated with the **stderr**, after which any network connections are closed. An error is indicated by a leading byte with a value of 1 (0 is returned in step 7 above upon successful completion of all the steps prior to the command execution).

"username too long"
The name is longer than 16 characters.

"password too long"
The password is longer than 16 characters.

"command too long "
The command line passed exceeds the size of the argument list (as configured into the system).

"Login incorrect."
No password file entry for the user name existed.

"Password incorrect."
The wrong was password supplied.

"No remote directory."
The *chdir* command to the home directory failed.

"Try again."
A *fork* by the server failed.

"/bin/sh: ..."
The user's login shell could not be started. This message is returned on the connection associated with the **stderr**, and is not preceded by a flag byte.

SEE ALSO
rexec(3)

BUGS
Indicating "Login incorrect" as opposed to "Password incorrect" is a security breach which allows people to probe a system for users with null passwords.

A facility to allow all data and password exchanges to be encrypted should be present.

NAME

rivet – Rivet a symbol table to the end of a vmunix file

SYNOPSIS

/bin/nm *vmunix* > *nm.output*
/etc/rivet *vmunix nm.output*

DESCRIPTION

rivet is used to append a specially sorted symbol table to the end of a vmunix file. This symbol table is used by the SPU programs *sysboot*, *jpd*, and *jpstat* in order to allow fast symbol table lookups while minimizing the memory requirements on the SPU. Normally, *rivet* should only be invoked in the makefiles generated by *sysgen*.

SEE ALSO

sysgen(8)

NAME

rlogind, in.rlogind – remote login server

SYNOPSIS

/etc/in.rlogind [**–ln**]

DESCRIPTION

rlogind is the server for the *rlogin*(1C) program. The server provides a remote login facility with authentication based on privileged port numbers from trusted hosts.

rlogind is invoked by *inetd*(8C) when a remote login connection is established, and initiates the following protocol:

1) The server checks the client's source port. If the port is not in the range 512-1023, the server aborts the connection.

2) The server checks the client's source address and requests the corresponding host name (see *gethostbyaddr*(3N), *hosts*(5) and *named*(8)). If the host name cannot be determined, the dot-notation representation of the host address is used.

Once the source port and address have been checked, *rlogind* allocates a pseudo terminal (see *pty*(4)), and manipulates file descriptors so that the slave half of the pseudo terminal becomes the **stdin**, **stdout**, and **stderr** for a login process. The login process is an instance of the *login*(1) program, invoked with the **–r** option. The login process then proceeds with the authentication process as described in *rshd*(8C), but if automatic authentication fails, it reprompts the user to login as one finds on a standard terminal line. The *-l* option prevents any authentication based on the user's ".rhosts" file, unless the user is logging in as the superuser.

The parent of the login process manipulates the master side of the pseudo-terminal, operating as an intermediary between the login process and the client instance of the *rlogin* program. In normal operation, the packet protocol described in *pty*(4) is invoked to provide **^S/^Q** type facilities and propagate interrupt signals to the remote programs. The login process propagates the client terminal's baud rate and terminal type, as found in the environment variable, "TERM"; see *environ*(7). The screen or window size of the terminal is requested from the client, and window size changes from the client are propagated to the pseudo terminal.

Transport-level keepalive messages are enabled unless the **–n** option is present. The use of keepalive messages allows sessions to be timed out if the client crashes or becomes unreachable.

DIAGNOSTICS

All diagnostic messages are returned on the connection associated with the **stderr**, after which any network connections are closed. An error is indicated by a leading byte with a value of 1.

"Try again."
A *fork* by the server failed.

"/bin/sh: ..."
The user's login shell could not be started.

SEE ALSO

rlogin(1C), services(5), inetd(8C), ruserok(3), rshd(8)

BUGS

The authentication procedure used here assumes the integrity of each client machine and the connecting medium. This is insecure, but is useful in an "open" environment.

A facility to allow all data exchanges to be encrypted should be present.

A more extensible protocol should be used.

NOTES

rlogind is an optional product; for more information, contact your CONVEX sales representative.

NAME

rmst – remove stripe entry

SYNOPSIS

rmst *[-k] stripedev ...*
rmst *-H diskdev ...*

DESCRIPTION

rmst deletes a stripe entry from kernel memory, or with the -H option, from the Hot Spare list and updates the */etc/stripecap* file. The utility verifies that the device is not mounted before performing any updates. Any *mvst* processes or parity checking activity on the given stripes will be immediately aborted by this utility and the stripe removed, so care should be taken with its use.

Switch options available:

–H Removes the referenced disk device(s) (e.g. du1a) from the Hot Spare list only. In this mode the diskdev argument may alternatively be referenced by hot spare entry name (i.e. rmst -H hs0).

–k Removes the referenced stripe(s) from the kernel tables only. This option will abort any *mvst* processes or parity checking activity on the given stripes.

The user must have root privilege to execute this utility.

Entries in */etc/stripecap* are created with the *newst*(8) utility.

FILES

/etc/stripecap Database of the stripe descriptors.

SEE ALSO

st(4), stripecap(5), newst(8), getst(8), putst(8), mvst(8), qst(8), convst(8)

NAME

rmt – remote magtape protocol module

SYNOPSIS

/etc/rmt

DESCRIPTION

rmt is a program used by the remote dump and restore programs in manipulating a magnetic tape drive through an interprocess communication connection. *rmt* is normally started up with an *rexec*(3X) or *rcmd*(3X) call.

The *rmt* program accepts requests specific to the manipulation of magnetic tapes, performs the commands, then responds with a status indication. All responses are in ASCII and in one of two forms. Successful commands have responses of

A*number*\n

where *number* is an ASCII representation of a decimal number. Unsuccessful commands are responded to with

E*error-number*\n*error-message*\n,

where *error-number* is one of the possible error numbers described in *intro*(2) and *error-message* is the corresponding error string as printed from a call to *perror*(3). The protocol is comprised of the following commands.

O*device*\n*mode*\n
: Open the specified *device* using the indicated *mode*. *device* is a full pathname and *mode* is an ASCII representation of a decimal number suitable for passing to *open*(2). If a device had already been opened, it is closed before a new open is performed.

C*device*\n
: Close the currently open device. The *device* specified is ignored.

L*whence*\n*offset*\n
: Perform an *lseek*(2) operation using the specified parameters. The response value is that returned from the *lseek* call.

W*count*\n
: Write data onto the open device. *rmt* reads *count* bytes from the connection, aborting if a premature end-of-file is encountered. Write commands are different from the others, in that they are done in an asynchronous manner. That is, a response value is returned immediately after reading the bytes from the connection. If the previous write command was successful, then the number of bytes just read from the connection will be returned. If the previous write command failed, then an error response will be issued. The result of this asynchronous behavior is improved performance when performing remote dumps. A possible negative side effect of this system occurs if the final write to tape fails. This failure will not be reported until a close command is attempted. When running *rdump*, this error message will appear, but you will likely not be prompted to abort or continue, as you would if an earlier write command failed.

R*count*\n
: Read *count* bytes of data from the open device. If *count* exceeds the size of the data buffer (10 kilobytes), it is truncated to the data buffer size. *rmt* then performs the requested *read*(2) and responds with A*count-read*\n if the read was successful; otherwise an error in the standard format is returned. If the read was successful, the data read is then sent.

I*operation*\n*count*\n
: Perform a MTIOCOP *ioctl*(2) command using the specified parameters. The parameters are interpreted as the ASCII representations of the decimal values to place in the *mt_op* and *mt_count* fields of the structure used in the *ioctl* call.

The return value is the *count* parameter when the operation is successful.

S Return the status of the open device, as obtained with a MTIOCGET *ioctl* call. If the operation was successful, an "ack" is sent with the size of the status buffer, then the status buffer is sent (in binary).

Any other command causes *rmt* to exit.

DIAGNOSTICS

All responses are of the form described above.

SEE ALSO

rcmd(3X), rexec(3X), mtio(4), rdump(8C), rrestore(8C)

BUGS

People tempted to use this for a remote file access protocol are discouraged.

If the final write command to tape fails, the message on the *rdump* side will come after a close command has been issued, and you will not get prompted to abort or continue.

NAME

route – manually manipulate the routing tables

SYNOPSIS

/etc/route [**–f**] [**–n**] [*command args*]

DESCRIPTION

route is a program used to manually manipulate the network routing tables. It normally is not needed, as the system routing table management daemon, *routed*(8C), should tend to this task.

route accepts two commands: *add*, to add a route, and *delete*, to delete a route.

All commands have the following syntax:

/etc/route *command* [**net** | **host**] *destination gateway* [*metric*]

where *destination* is the destination host or network, *gateway* is the next-hop gateway to which packets should be addressed, and *metric* is a count indicating the number of hops to the *destination*. The metric is required for *add* commands; it must be zero if the destination is on a directly-attached network, and nonzero if the route utilizes one or more gateways. If adding a route with metric 0, the gateway given is the address of this host on the common network, indicating the interface to be used for transmission. Routes to a particular host are distinguished from those to a network by interpreting the Internet address associated with *destination*. The optional keywords **net** and **host** force the destination to be interpreted as a network or a host, respectively. Otherwise, if the *destination* has a "local address part" of INADDR_ANY, or if the *destination* is the symbolic name of a network, then the route is assumed to be to a network; otherwise, it is presumed to be a route to a host. If the route is to a destination connected via a gateway, the *metric* should be greater than 0. All symbolic names specified for a *destination* or *gateway* are looked up first as a host name using *gethostbyname*(3N). If this lookup fails, *getnetbyname*(3N) is then used to interpret the name as that of a network. If the destination of a gateway is specified as **default** that gateway is used if no other route can be found. This is not a good thing to do on large networks, because it can lead to packets being passed around unnecessarily.

route uses a raw socket and the SIOCADDRT and SIOCDELRT *ioctl*'s to do its work. As such, only the super-user may modify the routing tables.

If the **–f** option is specified, *route* will "flush" the routing tables of all gateway entries. If this is used in conjunction with one of the commands described above, the tables are flushed prior to the command's application.

The **–n** option prevents attempts to print host and network names symbolically when reporting actions.

DIAGNOSTICS

"**add [host | network] %s: gateway %s flags %x**"
The specified route is being added to the tables. The values printed are from the routing table entry supplied in the *ioctl* call. If the gateway address used was not the primary address of the gateway (the first one returned by *gethostbyname*), the gateway address is printed numerically as well as symbolically.

"**delete [host | network] %s: gateway %s flags %x**"
As above, but when deleting an entry.

"**%s %s done**"
When the **–f** flag is specified, each routing table entry deleted is indicated with a message of this form.

"**Network is unreachable**"
An attempt to add a route failed because the gateway listed was not on a directly-connected network. The next-hop gateway must be given.

"not in table"
A delete operation was attempted for an entry which wasn't present in the tables.

"routing table overflow"
An add operation was attempted, but the system was low on resources and was unable to allocate memory to create the new entry.

FILES

/vmunix	Kernel name list
/dev/mem	Kernel data values
*/lib/kernsyms/symdata_**	Kernel symbol addresses

SEE ALSO
intro(4N), routed(8C),

NOTES
route is an optional product; for more information, contact your CONVEX sales representative.

NAME

routed – network routing daemon

SYNOPSIS

/etc/routed [**-d**] [**-g**] [**-s**] [**-q**] [**-t**] [*logfile*]

DESCRIPTION

routed is invoked at boot time to manage the network routing tables. The routing daemon uses a variant of the Xerox NS Routing Information Protocol in maintaining up to date kernel routing table entries. It uses a generalized protocol capable of use with multiple address types, but is currently used only for Internet routing within a cluster of networks.

In normal operation *routed* listens on the *udp*(4p) socket for the *route* service (see *services*(5)) for routing information packets. If the host is an internetwork router, it periodically supplies copies of its routing tables to any directly connected hosts and networks.

When *routed* is started, it uses the SIOCGIFCONF *ioctl* to find those directly connected interfaces configured into the system and marked "up" (the software loopback interface is ignored). If multiple interfaces are present, it is assumed that the host will forward packets between networks. *routed* then transmits a *request* packet on each interface (using a broadcast packet if the interface supports it) and enters a loop, listening for *request* and *response* packets from other hosts.

When a *request* packet is received, *routed* formulates a reply based on the information maintained in its internal tables. The *response* packet generated contains a list of known routes, each marked with a "hop count" metric (a count of 16, or greater, is considered "infinite"). The metric associated with each route returned provides a metric *relative to the sender*.

Response packets received by *routed* are used to update the routing tables if one of the following conditions is satisfied:

(1) No routing table entry exists for the destination network or host, and the metric indicates the destination is "reachable" (i.e. the hop count is not infinite).

(2) The source host of the packet is the same as the router in the existing routing table entry. That is, updated information is being received from the very internetwork router through which packets for the destination are being routed.

(3) The existing entry in the routing table has not been updated for some time (defined to be 90 seconds) and the route is at least as cost effective as the current route.

(4) The new route describes a shorter route to the destination than the one currently stored in the routing tables; the metric of the new route is compared against the one stored in the table to decide this.

When an update is applied, *routed* records the change in its internal tables and updates the kernel routing table. The change is reflected in the next *response* packet sent.

In addition to processing incoming packets, *routed* also periodically checks the routing table entries. If an entry has not been updated for 3 minutes, the entry's metric is set to infinity and marked for deletion. Deletions are delayed an additional 60 seconds to insure the invalidation is propagated throughout the local internet.

Hosts acting as internetwork routers gratuitously supply their routing tables every 30 seconds to all directly connected hosts and networks. The response is sent to the broadcast address on nets capable of that function, to the destination address on point-to-point links, and to the router's own address on other networks. The normal routing tables are bypassed when sending gratuitous responses. The reception of responses on each network is used to determine that the network and interface are functioning correctly. If no response is received on an interface, another route may be chosen to route around the interface, or the route may be dropped if no alternative is available.

routed supports several options:

-d Enable additional debugging information to be logged, such as bad packets received.

-g This flag is used on internetwork routers to offer a route to the "default" destination. This is typically used on a gateway to the Internet, or on a gateway that uses another routing protocol whose routes are not reported to other local routers.

-s Supplying this option forces *routed* to supply routing information whether it is acting as an internetwork router or not. This is the default if multiple network interfaces are present, or if a point-to-point link is in use.

-q This is the opposite of the -s option.

-t If the -t option is specified, all packets sent or received are printed on the standard output. In addition, *routed* will not divorce itself from the controlling terminal so that interrupts from the keyboard will kill the process.

Any other argument supplied is interpreted as the name of file in which *routed*'s actions should be logged. This log contains information about any changes to the routing tables and, if not tracing all packets, a history of recent messages sent and received which are related to the changed route.

In addition to the facilities described above, *routed* supports the notion of "distant" *passive* and *active* gateways. When *routed* is started up, it reads the file */etc/gateways* to find gateways which may not be located using only information from the SIOGIFCONF *ioctl*. Gateways specified in this manner should be marked passive if they are not expected to exchange routing information, while gateways marked active should be willing to exchange routing information (i.e. they should have a *routed* process running on the machine). Routes through passive gateways are installed in the kernel's routing tables once upon startup. Such routes are not included in any routing information transmitted. Active gateways are treated equally to network interfaces. Routing information is distributed to the gateway and if no routing information is received for a period of the time, the associated route is deleted. Gateways marked *external* are also passive, but are not placed in the kernel routing table nor are they included in routing updates. The function of external entries is to inform *routed* that another routing process will install such a route, and that alternate routes to that destination should not be installed. Such entries are only required when both routers may learn of routes to the same destination.

Internetwork routers that are directly attached to the Arpanet or Milnet should use the Exterior Gateway Protocol (EGP) to gather routing information rather than using a static routing table of passive gateways. EGP is required in order to provide routes for local networks to the rest of the Internet system. Sites needing assistance with such configurations should contact the CONVEX Technical Assistance Center.

FILES

/etc/gateways for distant gateways

SEE ALSO

"Internet Transport Protocols", XSIS 028112, Xerox System Integration Standard.
udp(4), icmp(4), htable(8), gateways(5)

BUGS

The kernel's routing tables may not correspond to those of *routed* when redirects change or add routes. *routed* should note any redirects received by reading the ICMP packets received via a raw socket.

routed should incorporate other routing protocols, such as EGP. Using separate processes for each requires configuration options to avoid redundant or competing routes.

routed should listen to intelligent interfaces, such as an IMP, to gather more information. It does not always detect unidirectional failures in network interfaces (e.g., when the output side fails).

NOTES

routed is an optional product; for more information, contact your CONVEX sales representative.

NAME

rrestore – restore a file system dump across the network

SYNOPSIS

/etc/rrestore key [name ...]

DESCRIPTION

rrestore obtains from magnetic tape files saved by a previous *dump*(8) or *rdump*(8c). *rrestore* uses the same format and keys as *restore*(8); in fact, *rrestore* is simply an extension of *restore*(8). To use *rrestore*, you must use the *f* key with an argument of the form *machine:device*. If the *f* key is not specified, dumphost:/dev/rmt8 is used.

rrestore creates a remote server, */etc/rmt*, on the client machine to access the tape device.

FILES

~/.rhosts
: This file exists with *rmt*(8C) on the server host. It contains a list of the Internet client host names having access to the server host.

dumphost:/dev/rmt8
: The default remote tape drive to use if the *f* key is not specified.

SEE ALSO

rhosts(5), restore(8), rmt(8C)

DIAGNOSTICS

Same as *restore*(8) with a few extra diagnostics related to the network.

RESTRICTIONS

The 100ips option of *restore*(8), flag **I**, is not currently supported on remote devices.

NOTES

Only the superuser may perform an *rrestore*.

rrestore is an optional product; for more information, contact your CONVEX sales representative.

NAME

rshd, in.rshd – remote shell server

SYNOPSIS

/etc/in.rshd [**–d -l -n**] host.port

DESCRIPTION

rshd is the server for the *rcmd*(3X) routine and, consequently, for the *rsh*(1C) program. The server provides remote execution facilities with authentication based on privileged port numbers.

-d causes debugging information to be output and prevents rshd from forking upon startup.

-l prevents any authentication based on the user's ".rhosts" file, unless the user is logging in as the superuser.

-n disables transport-level keepalive messages. The use of keepalive messages allows sessions to be timed out if the client crashes or becomes unreachable.

rshd is invoked by *inetd*(8C) when a remote shell connection is established, and initiates the following protocol:

1) The server checks the client's source port. If the port is not in the range 0-1023, the server aborts the connection. The clients address and port number are passed as arguments to *rshd* by *inetd* in the form "host.port" with host in hex and port in decimal.

2) The server reads characters from the socket up to a null ("\0") byte. The resultant string is interpreted as an ASCII number, base 10.

3) If the number received in step 2 is non-zero, it is interpreted as the port number of a secondary stream to be used for the **stderr**. A second connection is then created to the specified port on the client's machine. The source port of this second connection is also in the range 0-1023.

4) The server checks the client's source address and requests the corresponding host name (see *gethostbyaddr*(3N), *hosts*(5) and *named*(8)). If the hostname cannot be determined, the dot-notation representation of the host address is used.

5) A null terminated user name of at most 16 characters is retrieved on the initial socket. This user name is interpreted as a user identity to use on the **client**'s machine.

6) A null terminated user name of at most 16 characters is retrieved on the initial socket. This user name is interpreted as the user identity on the **server**'s machine.

7) A null terminated command to be passed to a shell is retrieved on the initial socket. The length of the command is limited by the upper bound on the size of the system's argument list.

8) *rshd* then validates the user according to the following steps. The remote user name is looked up in the password file and a *chdir* is performed to the user's home directory. If either the lookup or *chdir* fail, the connection is terminated. If the user is not the superuser, (user ID 0), the file */etc/hosts.equiv* is consulted for a list of hosts considered "equivalent". If the client's host name is present in this file, the authentication is considered successful. If the lookup fails, or the user is the superuser, then the file *.rhosts* in the home directory of the remote user is checked for the machine name and identity of the user on the client's machine. If this lookup fails, the connection is terminated.

9) A null byte is returned on the connection associated with the **stderr** and the command line is passed to the normal login shell of the user. The shell inherits the network connections established by *rshd*.

DIAGNOSTICS

Except for the last one listed below, all diagnostic messages are returned on the connection associated with the **stderr**, after which any network connections are closed. An error is indicated by a

leading byte with a value of 1 (0 is returned in step 9 above upon successful completion of all the steps prior to the command execution).

"locuser too long"
The name of the user on the client's machine is longer than 16 characters.

"remuser too long"
The name of the user on the remote machine is longer than 16 characters.

"command too long "
The command line passed exceeds the size of the argument list (as configured into the system).

"Login incorrect."
No password file entry for the user name existed.

"No remote directory."
The *chdir* command to the home directory failed.

"Permission denied."
The authentication procedure described above failed.

"Can't make pipe."
The pipe needed for the **stderr**, wasn't created.

"Try again."
A *fork* by the server failed.

"<shellname>: ..."
The user's login shell could not be started. This message is returned on the connection associated with the **stderr**, and is not preceded by a flag byte.

"/bin/sh: ..."
The user's login shell could not be started.

SEE ALSO

rsh(1C), rcmd(3X), services(5), inetd(8C)

BUGS

The authentication procedure used here assumes the integrity of each client machine and the connecting medium. This is insecure, but is useful in an "open" environment.

A facility to allow all data exchanges to be encrypted should be present.

NOTES

rshd is an optional product; for more information, contact your CONVEX sales representative.

NAME

rwhod – system status server

SYNOPSIS

/etc/rwhod

DESCRIPTION

rwhod is the server which maintains the database used by the *rwho*(1C) and *ruptime*(1C) programs. Its operation is predicated on the ability to *broadcast* messages on a network.

rwhod operates as both a producer and consumer of status information. As a producer of information it periodically queries the state of the system and constructs status messages which are broadcast on a network. As a consumer of information, it listens for other *rwhod* servers' status messages, validating them, then recording them in a collection of files located in the directory */usr/spool/rwho*.

The server transmits and receives messages at the port indicated in the "rwho" service specification; see *services*(5). The messages sent and received, are of the form:

```
struct  outmp {
        char    out_line[8];/* tty name */
        char    out_name[8];/* user id */
        long    out_time;/* time on */
};

struct  whod {
        char    wd_vers;
        char    wd_type;
        char    wd_fill[2];
        int     wd_sendtime;
        int     wd_recvtime;
        char    wd_hostname[32];
        int     wd_loadav[3];
        int     wd_boottime;
        struct  whoent {
                struct  outmp we_utmp;
                int     we_idle;
        } wd_we[1024 / sizeof (struct whoent)];
};
```

All fields are converted to network byte order prior to transmission. The load averages are as calculated by the *w*(1) program, and represent load averages over the 5, 10, and 15 minute intervals prior to a server's transmission; they are multiplied by 100 for representation in an integer. The host name included is that returned by the *gethostname*(2) system call, with any trailing domain name omitted. The array at the end of the message contains information about the users logged in to the sending machine. This information includes the contents of the *utmp*(5) entry for each non-idle terminal line and a value indicating the time in seconds since a character was last received on the terminal line.

Messages received by the *rwho* server are discarded unless they originated at an *rwho* server's port. In addition, if the host's name, as specified in the message, contains any unprintable ASCII characters, the message is discarded. Valid messages received by *rwhod* are placed in files named *whod.hostname* in the directory */usr/spool/rwho*. These files contain only the most recent message, in the format described above.

Status messages are generated approximately once every 3 minutes. *rwhod* performs an *nlist*(3) on /vmunix every 30 minutes to guard against the possibility that this file is not the system image currently operating.

FILES

/vmunix	Kernel name list
/dev/mem	Kernel data values
*/lib/kernsyms/symdata_**	Kernel symbol addresses

SEE ALSO

rwho(1C), ruptime(1C)

BUGS

There should be a way to relay status information between networks. Status information should be sent only upon request rather than continuously. People often interpret the server dying or network communtication failures as a machine going down.

NOTES

rwhod is an optional product; for more information, contact your CONVEX sales representative.

NAME

sa, accton – system accounting

SYNOPSIS

/etc/sa [**–abcdDefFgijkKlmnprstuv**] [*acctfile*]

/etc/accton [*acctfile*]

DESCRIPTION

With an argument naming an existing *file*, *accton* causes system accounting information for every process executed to be placed at the end of the file. If no argument is given, accounting is turned off.

sa reports on, cleans up, and generally maintains accounting files.

sa is able to condense the information in the accounting file into 2 summary files. The accounting file is assumed to be */usr/adm/acct* unless an alternate accounting file is specified in the last argument.

The accounting system automatically stops placing data at the end of */usr/adm/acct* if the disk space in the */usr/adm* directory exceeds 98% capacity. When this occurs, accounting will be automatically restarted when the available space drops back to 96% of capacity. To avoid losing accounting data, allow plenty of extra disk space in the */usr/adm* directory.

/usr/adm/savacct contains a count of the number of times each command was executed, CPU time consumed, total I/O, and memory usage. The other summary file is */usr/adm/usracct*, which contains similar information for each user and also for each unique group-activity combination. The condensation of the accounting file into summary files is desirable because on a large system the accounting file can grow by 100 blocks per day. The summary files are normally read before the accounting file, so the reports include all available information.

Output fields are labeled: "CPU" for the sum of user+system time (in minutes), "re" for real time (also in minutes), "k" for CPU-time averaged core usage (in 1-Kbyte units), "avio" for average number of I/O operations per execution. Various command-line switches can cause fields to appear labeled "tio" for total I/O operations, "k*sec" for CPU storage integral (kilo-core seconds), and "u" and "s" for user and system CPU time alone (both in minutes). See *acct*(5) for more information on the CPU storage intergral.

sa recognizes the following command-line switches:

a Place all command names containing unprintable characters and those used only once under the name "***other".

b Sort output by sum of user and system time divided by number of calls. Default sort is by sum of user and system times.

c Besides total user, system, and real time for each command print percentage of total time over all commands.

d Sort by average number of disk I/O operations.

D Print and sort by total number of disk I/O operations.

e Echo records exactly as they appear in the accounting file; this option may be used to convert the accounting file to ASCII text. The fields are, in order: command name, timestamp when execution started, total CPU time in seconds, TTY name, user ID, group ID, activity ID, elapsed time in seconds, CPU storage integral (kilo-core seconds), disk I/O blocks, system CPU time in seconds, the average level of concurrency (a floating point value between one and the number of processors in the system, indicating the average number of processors dedicated to the process at each clock interval; if this value is zero, then that particular accounting record was collected prior to Version 7.0 of the operating system), and a character flag indicating in which scheduling mode the

command was executed (**f** signifies that it ran as a fixed-schedule job at least part of the time; **d** represents a dynamic-schedule job (dynamic is the default scheduling mode)). User CPU time may be computed by subtracting system CPU time from total CPU time. See *acct*(5) for more information on the accounting file.

f Force no interactive threshold compression with **–v** flag.

F*c* Append the character *c* to the summary files, thus summarizing data in files */usr/adm/savacctc* and */usr/adm/usracctc*.

g For each unique group-activity combination, print a record containing the group name, activity name, number of processes, CPU minutes, total I/O, and memory usage.

i Don't read in summary files.

j Instead of total minutes time for each category, give seconds per call.

k Sort by CPU-time average memory usage.

K Print and sort by CPU-storage integral.

l Separate system and user time; normally they are combined.

m Print number of processes, CPU minutes, total I/O, and memory usage for each user.

n Sort by number of calls.

p Preserve (do not truncate) the accounting file when creating summaries with the **–s** option.

r Reverse order of sort.

s Use the data in the accounting file to create a summary of process data in file */usr/adm/savacct*, and a summary of user and group-activity data in file */usr/adm/usracct*. After the summaries have been created, remove all data in the accounting file.

t For each command report ratio of real time to the sum of user and system times.

u Superseding all other flags except **-e**, print the user ID and command name for each command in the accounting file.

v Followed by a number *n*, type the name of each command used *n* times or fewer. Await a reply from the terminal; if it begins with "y", add the command to the category "**junk**." This is used to strip out garbage.

FILES

/usr/adm/acct	raw accounting
/usr/adm/savacct	per process summary
/usr/adm/usracct	per user and per group-activity summary

SEE ALSO

lastcomm(1), mpa(1), acct(2), acct(5), activities(5), ac(8), "Accounting Reports" chapter in *Managing ConvexOS:Operations Guide*.

NAME

stat, seestat – gather system statistics and print graphs of system statistics

SYNOPSIS

seestat [*options*]
stat [**-i** *interval*]

DESCRIPTION

seestat uses data collected by *stat* to print graphs of system use and performance. *stat* collects system data every *interval* seconds. If no interval is specified the default is thirty seconds. Currently *seestat* only supports output to a Printronix printer. It opens a pipe directly to *lpr -l*, so the environment variable **PRINTER** must be set to the name of a Printronix printer. The options of *seestat* are:

-a*numb* Specify a number which varies the smoothness of the curve through a moving average algorithm. A value of **1** will print the graph with no smoothing. The default value is 12.

-d*date* Specify the date to print a graph for, in the form **month/day[/year]**. The default is yesterday.

-m*max* Specify the maximum y-axis position. The defaults are dependent on the statistic chosen. The argument **mm** causes the highest data point to be taken as the top of the axis.

-n*numb* Specify the number of the daily graph to be printed:

1. Buffered I/O latency	2. Buffered I/O Mb/s
3. Disk Mb/s	4. Stripe Mb/s
5. Network packets in	6. Network packets out
7. Network input errors	8. Network output errors
9. Network collisions	10. CPU user time pct
11. CPU niced time pct	12. CPU system time pct
13. CPU idle time pct	14. Virtual memory activated
15. Free virtual memory	16. Page ins/sec
17. Page outs/sec	18. Device interrupts/sec
19. System calls/sec	20. Load average
21. Disk request size	22. Real memory available
23. System users	24. Page reclaims/sec
25. Page faults/sec	26. Page hits %
27. Total # processes	28. Pages scanned by clock/sec
29. Context switches/sec	30. Buffered I/O hits %

-t*dots* Specify the size of the time- (x-) axis in dots. Defaults to 600.

-w*numb* Specify weekly graph to print. Defaults to 1.

-y*dots* Specify the size of the y-axis in dots. Defaults to 190.

The ordering of the arguments to *seestat* is significant. The **n** and **w** options will actually cause the printing of graphs as the options are encountered in the command line. Therefore, each graph is drawn with only the options activated that *precede* the **n** or **w** option. Later options will affect only graphs printed with a later **n** or **w** option.

EXAMPLES

To print a graph of yesterday's load average with a moving average of 20, enter the following command:

```
seestat -a20 -n21
```

The similar command

seestat -n21 -a20

will *not* give the same results, because the **a** option *follows* the **n**. The **a** in this case has no effect.

For a weekly summary graph beginning June 21st of the current year showing the number of system users, type:

seestat -d6/21 -w24

It is also possible to print several graphs in one command. For example,

seestat -a21 -d3/11 -w21 -d3/18 -w21 -d3/12 -n24

will print three graphs, two weekly summaries of load average for the weeks beginning March 11th and March 18th of the current year, and a graph of daily system users for March 12th.

FILES

/dev/kmem, /vmunix, /etc/utmp	data sources
/usr/adm/stat/stats*	data stored by *stat*
/lib/kernsyms/symdata_*	Kernel symbol addresses

SEE ALSO

vmstat(1), uptime(1)

NAME

sendmail – send mail over the internet

SYNOPSIS

/usr/lib/sendmail [flags] [address ...]

mailq [**–v**]

DESCRIPTION

sendmail sends a message to one or more *recipients*, routing the message over whatever networks are necessary. *sendmail* does internetwork forwarding as necessary to deliver the message to the correct place.

sendmail is not intended as a user interface routine; other programs provide user-friendly front ends; *sendmail* is used only to deliver pre-formatted messages.

With no flags, *sendmail* reads its standard input up to an end-of-file or a line consisting only of a single dot and sends a copy of the message found there to all of the addresses listed. It determines the network(s) to use based on the syntax and contents of the addresses.

Local addresses are looked up in a file and aliased appropriately. Aliasing can be prevented by preceding the address with a backslash. Normally the sender is not included in any alias expansions, e.g., if 'john' sends to 'group', and 'group' includes 'john' in the expansion, then the letter will not be delivered to 'john'.

Flags are:

–ba Go into ARPANET mode. All input lines must end with a CR-LF, and all messages will be generated with a CR-LF at the end. Also, the "From:" and "Sender:" fields are examined for the name of the sender.

–bd Run as a daemon. This requires Berkeley IPC. *sendmail* will fork and run in background listening on socket 25 for incoming SMTP connections. This is normally run from */etc/rc*. When run in this mode, *sendmail* alters it's "name" (as seen by ps(1)) to show what task it is performing. The most common names are:

accepting connections
rejecting connections: load average <average>
running queue
srvrsmtp <hostname>

–bi Initialize the alias database.

–bm Deliver mail in the usual way (default).

–bp Print a listing of the queue.

–bs Use the SMTP protocol as described in RFC821 on standard input and output. This flag implies all the operations of the **–ba** flag that are compatible with SMTP.

–bt Run in address test mode. This mode reads addresses and shows the steps in parsing; it is used for debugging configuration tables.

–bv Verify names only – do not try to collect or deliver a message. Verify mode is normally used for validating users or mailing lists on the local machine.

–bz Create the configuration freeze file.

–C*file* Use alternate configuration file. *sendmail* refuses to run as root if an alternate configuration file is specified. The frozen configuration file is bypassed.

–d*X* Set debugging value to *X*.

–F*fullname* Set the full name of the sender.

–f*name* Sets the name of the "from" person (i.e., the sender of the mail). **–f** can only be used by "trusted" users (normally *root, daemon,* and *network)* or if the person you are trying to become is the same as the person you are.

–h*N* Set the hop count to *N*. The hop count is incremented every time the mail is processed. When it reaches a limit, the mail is returned with an error message, the victim of an aliasing loop. If not specified, "Received:" lines in the message are counted.

–n Don't do aliasing.

–o*x value* Set option *x* to the specified *value*. Options are described below.

–q[*time*] Processed saved messages in the queue at given intervals. If *time* is omitted, process the queue once. *Time* is given as a tagged number, with 's' being seconds, 'm' being minutes, 'h' being hours, 'd' being days, and 'w' being weeks. For example, "–q1h30m" or "–q90m" would both set the timeout to one hour thirty minutes. If *time* is specified, *sendmail* will run in background. This option can be used safely with **–bd**.

–r*name* An alternate and obsolete form of the **–f** flag.

–t Read message for recipients. To:, Cc:, and Bcc: lines will be scanned for recipient addresses. The Bcc: line will be deleted before transmission. Any addresses in the argument list will be suppressed, that is, they will *not* receive copies even if listed in the message header.

–v Go into verbose mode. Alias expansions will be announced, etc.

There are also a number of processing options that may be set. Normally these will only be used by a system administrator. Options may be set either on the command line using the **–o** flag or in the configuration file. These are described in detail in the *Sendmail Installation and Operation Guide*. The options are:

A*file* Use alternate alias file.

c On mailers that are considered "expensive" to connect to, don't initiate immediate connection. This requires queuing.

d*x* Set the delivery mode to *x*. Delivery modes are 'i' for interactive (synchronous) delivery, 'b' for background (asynchronous) delivery, and 'q' for queue only – i.e., actual delivery is done the next time the queue is run.

D Try to automatically rebuild the alias database if necessary.

e*x* Set error processing to mode *x*. Valid modes are 'm' to mail back the error message, 'w' to "write" back the error message (or mail it back if the sender is not logged in), 'p' to print the errors on the terminal (default), 'q' to throw away error messages (only exit status is returned), and 'e' to do special processing for the BerkNet. If the text of the message is not mailed back by modes 'm' or 'w' and if the sender is local to this machine, a copy of the message is appended to the file "dead.letter" in the sender's home directory.

F*mode* The mode to use when creating temporary files.

f Save UNIX-style From lines at the front of messages. (UNIX is a registered trademark of UNIX System Laboratories, Inc.)

g*N* The default group id to use when calling mailers.

H*file* The SMTP help file.

i Do not take dots on a line by themselves as a message terminator.

L*n*	The log level.
m	Send to "me" (the sender) also if I am in an alias expansion.
o	If set, this message may have old style headers. If not set, this message is guaranteed to have new style headers (i.e., commas instead of spaces between addresses). If set, an adaptive algorithm is used that will correctly determine the header format in most cases.
Q*queuedir*	Select the directory in which to queue messages.
r*timeout*	The timeout on reads; if none is set, *sendmail* will wait forever for a mailer. This option violates the word (if not the intent) of the SMTP specification, show the timeout should probably be fairly large.
S*file*	Save statistics in the named file.
s	Always instantiate the queue file, even under circumstances where it is not strictly necessary. This provides safety against system crashes during delivery.
T*time*	Set the timeout on undelivered messages in the queue to the specified time. After delivery has failed (e.g., because of a host being down) for this amount of time, failed messages will be returned to the sender. The default is three days.
t*stz,dtz*	Set the name of the time zone.
u*N*	Set the default user id for mailers.

In aliases, the first character of a name may be a vertical bar to cause interpretation of the rest of the name as a command to pipe the mail to. It may be necessary to quote the name to keep *sendmail* from suppressing the blanks from between arguments. For example, a common alias is:

msgs: "|/usr/ucb/msgs -s"

Aliases may also have the syntax ":include:*filename*" to ask *sendmail* to read the named file for a list of recipients. For example, an alias such as:

poets: ":include:/usr/local/lib/poets.list"

would read */usr/local/lib/poets.list* for the list of addresses making up the group.

sendmail returns an exit status describing what it did. The codes are defined in <*sysexits.h*>

EX_OK	Successful completion on all addresses.
EX_NOUSER	User name not recognized.
EX_UNAVAILABLE	Catchall meaning necessary resources were not available.
EX_SYNTAX	Syntax error in address.
EX_SOFTWARE	Internal software error, including bad arguments.
EX_OSERR	Temporary operating system error, such as "cannot fork".
EX_NOHOST	Host name not recognized.
EX_TEMPFAIL	Message could not be sent immediately, but was queued.

If invoked as *mailq*, *sendmail* will print the contents of the mail queue.

FILES

Except for /usr/lib/sendmail.cf, these pathnames are all specified in /usr/lib/sendmail.cf. Thus, these values are only approximations.

/usr/lib/aliases	raw data for alias names
/usr/lib/aliases.pag	
/usr/lib/aliases.dir	data base of alias names
/usr/lib/sendmail.cf	configuration file

/usr/lib/sendmail.fc	frozen configuration
/usr/lib/sendmail.hf	help file
/usr/lib/sendmail.st	collected statistics
/usr/spool/mqueue/*	temp files

SEE ALSO

binmail(1), newaliases(1), mail(1), rmail(1), syslog(3), aliases(5), sendmail.cf(5), mqueue(5), mailaddr(7), rc(8)

NAME

shutdown – close down the system at a given time

SYNOPSIS

/etc/shutdown [**–k**] [**–r**] [**–h**] [**–q**] time [warning-message ...]

DESCRIPTION

shutdown provides an automated shutdown procedure which a superuser can use to notify users nicely when the system is shutting down, saving them from system administrators, hackers, and gurus, who would otherwise not bother with niceties.

time is the time at which *shutdown* will bring the system down and may be the word **now** (indicating an immediate shutdown) or specify a future time in one of two formats: +number and hour:min. The first form brings the system down in *number* minutes and the second brings the system down at the time of day indicated (as a 24–hour clock).

At intervals which get closer together as apocalypse approaches, warning messages are displayed at the terminals of all users on the system. Five minutes before shutdown, or immediately if shutdown is in less than 5 minutes, logins are disabled by creating */etc/nologin* and writing a message there. If this file exists when a user attempts to log in, *login*(1) prints its contents and exits. The file is removed just before *shutdown* exits.

At shutdown time a message is written in the file */usr/adm/shutdownlog*, containing the time of shutdown, who ran shutdown and the reason. Then a terminate signal is sent at *init* to bring the system down to single-user state. Alternatively, if **–r**, **–h**, or **–k** was used, then *shutdown* will exec *reboot*(8), *halt*(8), or avoid shutting the system down (respectively). (If it isn't obvious, **–k** is to make people *think* the system is going down!)

The time of the shutdown and the warning message are placed in */etc/nologin* and should be used to inform the users about when the system will be back up and why it is going down (or anything else).

If a remote host has mounted an NFS partition from this host, the shutdown message is also broadcast to that remote system via *rwall*(1). The **–q** option suppresses these messages from being sent to the remote system.

FILES

/etc/nologin	tells login not to let anyone except a superuser log in.
/etc/rmtab	list of remote hosts that have mounted this host.
/usr/adm/shutdownlog	log file for successful shutdowns.

SEE ALSO

login(1), rwall(1), wall(1), reboot(8)

BUGS

Only allows you to kill the system between now and 23:59 if you use the absolute time for shutdown.

NOTES

NFS is an optional product; for more information, contact your CONVEX sales representative.

NAME

silodismount – dismount a tape from a StorageTek Automated Cartridge System

SYNOPSIS

/usr/lib/tape/silodismount [–force] volID tapedevice

DESCRIPTION

The *silodismount* command is used to tell the StorageTek Automated Cartridge System to remove a 3480 tape cartridge from a StorageTek tape drive. The volume ID of the tape to be dismounted is specified with the *volID* parameter. A volume ID is a 6 character string consisting of a combination of the digits 0 (zero) through 9 and uppercase letters A through Z.

The *tapedevice* parameter specifies the CONVEX device name of the tape device from which the robot arm will remove the tape.

The *–force* option causes *silodismount* to take the tape drive offline even if the drive is currently in use. Normally, the user should take the drive offline with the *mt offl* command before a *silodismount* is attempted.

silodismount uses Remote Procedure Calls (or RPC) to communicate with the StorageTek Sun Server software on a remote Sun workstation. The hostname of this remote Sun is obtained from the environment variable *SILOHOST*. This environment variable should be set with the hostname of the Sun server in users' *.login* or *.profile* files.

After *silodismount* is executed, the robot arm will remove a tape from the specified tape drive and store it in the Library Storage Module (or LSM). After the tape is dismounted StorageTek software automatically chooses a slot to put the tape. The slot may or may not be the same slot that from which the tape was retrieved when it was mounted in the tape drive. There is no concept of a "home" location in the StorageTek server software. It is likely, however, that the tape will remain in the same LSM as the tape drive.

SEE ALSO

siloeject(8), siloenter(8), silomount(8), siloquery(8)

FILES

/usr/lib/tape/silodrivelist
: file that is used to map CONVEX tape device names to tape names used by the StorageTek Automated Cartridge System.

DIAGNOSTICS

clnttcp_create: RPC: Program not registered
: *siloenter* could not become a client of the StorageTek Sun Server software. Either the SILOHOST environment variable is set to the wrong hostname or the StorageTek software on the Sun is not running.

BUGS

None.

NAME

siloeject – eject a tape from a StorageTek Automated Cartridge System

SYNOPSIS

/usr/lib/tape/siloeject capID volID [**volID** ...]

DESCRIPTION

The *siloeject* command is used to eject up to 21 3480 tape cartridges from a StorageTek Automated Cartridge System (or tape silo). The volume IDs of the tapes to be ejected are specified with the *volID* parameters. A volume ID is a 6 character string consisting of a combination of the digits 0 (zero) through 9 and uppercase letters A through Z.

The *capID* parameter specifies the Controlled Access Port of the Automated Cartridge System (e.g. "0,0").

siloeject uses Remote Procedure Calls (or RPC) to communicate with the StorageTek Sun Server software on a remote Sun workstation. The hostname of this remote Sun is obtained from the environment variable *SILOHOST*. This environment variable should be set with the hostname of the Sun server in users' *.login* or *.profile* files.

After *siloeject* is executed, the Controlled Access Port (or CAP) is unlocked and the "CAP eject" light above the CAP is illuminated. At this point, the user should open the CAP door and remove the tapes from the slots. All tapes should be removed from CAP door. When the user closes the door, the robot arm will verify that the first slot (top left-hand slot) is empty. The process is finished when the "CAP locked" light is illuminated. After the "CAP locked" is illuminated, the volume ID and status of each of the tapes ejected is printed on the screen.

SEE ALSO

silodismount(8), siloenter(8), silomount(8), siloquery(8)

FILES

/usr/lib/tape/silodrivelist
: file that is used to map CONVEX tape device names to tape names used by the StorageTek Automated Cartridge System.

DIAGNOSTICS

clnttcp_create: RPC: Program not registered
: *siloenter* could not become a client of the StorageTek Sun Server software. Either the SILOHOST environment variable is set to the wrong hostname or the StorageTek software on the Sun is not running.

BUGS

None.

NAME

siloenter – enter a tape into a StorageTek Automated Cartridge System

SYNOPSIS

/usr/lib/tape/siloenter capID

DESCRIPTION

The *siloenter* command is used to enter up to 21 3480 tape cartridges into a StorageTek Automated Cartridge System (or tape silo). Each 3480 tape cartridge must have a bar-code volume serial number on the outside of the tape cartridge that is readable by the robot camera. Tapes without valid bar-code labels are rejected.

The *capID* parameter specifies the Controlled Access Port of the Automated Cartridge System (e.g. "0,0").

siloenter uses Remote Procedure Calls (or RPC) to communicate with the StorageTek Sun Server software on a remote Sun workstation. The hostname of this remote Sun is obtained from the environment variable *SILOHOST*. This environment variable should be set with the hostname of the Sun server in users' *.login* or *.profile* files.

After *siloenter* is executed, the Controlled Access Port (or CAP) is unlocked and the "CAP enter" light above the CAP is illuminated. At this point, the user should open the CAP door and insert the tapes to be entered in the slots. Tapes should be inserted from left to right starting from the top row. There should not be any empty slots between the first and last slots. When the user closes the door, the robot arm will verify that each tape has a unique, readable label and begin storing them inside the LSM (or silo). Any non-unique or unreadable labels are returned to the CAP and should be removed from the CAP door after the "CAP eject" light is illuminated. The process is finished when the "CAP locked" light is illuminated. After the "CAP locked" is illuminated, the volume ID and status of each of the tapes entered is printed on the screen.

SEE ALSO

silodismount(8), siloeject(8), silomount(8), siloquery(8)

FILES

/usr/lib/tape/silodrivelist
: file that is used to map CONVEX tape device names to tape names used by the StorageTek Automated Cartridge System.

DIAGNOSTICS

clnttcp_create: RPC: Program not registered
: *siloenter* could not become a client of the StorageTek Sun Server software. Either the SILOHOST environment variable is set to the wrong hostname or the StorageTek software on the Sun is not running.

BUGS

None.

NAME

silomount – mount a tape from a StorageTek Automated Cartridge System

SYNOPSIS

/usr/lib/tape/silomount volID tapedevice

DESCRIPTION

The *silomount* command is used to tell the StorageTek Automated Cartridge System to insert a 3480 tape cartridge in a StorageTek tape drive. The volume ID of the tape to be mounted is specified with the *volID* parameter. A volume ID is a 6 character string consisting of a combination of the digits 0 (zero) through 9 and uppercase letters A through Z.

The *tapedevice* parameter specifies the CONVEX tape device name in which the robot arm will mount the tape.

silomount uses Remote Procedure Calls (or RPC) to communicate with the StorageTek Sun Server software on a remote Sun workstation. The hostname of this remote Sun is obtained from the environment variable *SILOHOST*. This environment variable should be set with the hostname of the Sun server in users' *.login* or *.profile* files.

After *silomount* is executed, the robot arm will retrieve a tape from the Library Storage Module (or LSM) and insert it into the specified tape drive. The drive must not have a tape already mounted in it for this command to succeed. If a tape drive is connected to one LSM of a multi-LSM configuration and the specified tapes resides in another LSM, the tape will be automatically moved from the current LSM to the drive's LSM via pass-thru-ports.

SEE ALSO

silodismount(8), siloeject(8), siloenter(8), siloquery(8)

FILES

/usr/lib/tape/silodrivelist
: file that is used to map CONVEX tape device names to tape names used by the StorageTek Automated Cartridge System.

DIAGNOSTICS

clnttcp_create: RPC: Program not registered
: *siloenter* could not become a client of the StorageTek Sun Server software. Either the SILOHOST environment variable is set to the wrong hostname or the StorageTek software on the Sun is not running.

BUGS

None.

NAME

siloquery – get status of a StorageTek Automated Cartridge System component

SYNOPSIS

/usr/lib/tape/siloquery acs [**acsID** ...]
/usr/lib/tape/siloquery cap [**capID** ...]
/usr/lib/tape/siloquery drive [**driveID** ...]
/usr/lib/tape/siloquery lsm [**lsmID** ...]
/usr/lib/tape/siloquery mount volID [**volID** ...]
/usr/lib/tape/siloquery port [**portID** ...]
/usr/lib/tape/siloquery request [**requestID** ...]
/usr/lib/tape/siloquery server
/usr/lib/tape/siloquery volume [**volumeID** ...]

DESCRIPTION

The *siloquery* command is used to get the status of a StorageTek Automated Cartridge System component. The components are specified by one of the following keywords: acs, cap, drive, lsm, mount, port, request, server, or volume. Unless specific identifiers are supplied after the keyword, *siloquery* will print the status of all components of the same type in the StorageTek tape library. For example,

/usr/lib/tape/siloquery volume

will print the status of all volumes (or tapes) in the library. While

/usr/lib/tape/siloquery volume 012345 ACS321

prints only the status of the tapes with labels "012345" and "ACS321".

A description of the output for each keyword follows:

siloquery acs [acsID ...]
: Get the status of each ACS (Automated Cartridge System) or cluster of tape silos controlled by the StorageTek Sun Server software. Optionally, one or more specific ACSs can be specified on the command line with the *acsID* option (e.g. "siloquery acs 0").

siloquery cap [capID ...]
: Get the status of each CAP (Controlled Access Port) controlled by the StorageTek Sun Server software. Optionally, one or more specific CAPs can be specified on the command line with the *capID* option (e.g. "siloquery cap 0,1").

siloquery drive [driveID ...]
: Get the status of each tape drive controlled by the StorageTek Sun Server software. Optionally, one or more specific tape drives can be specified on the command line with the *driveID* option (e.g. "siloquery drive /dev/rtc1").

siloquery lsm [lsmID ...]
: Get the status of each LSM (Library Storage Module) or tape silo controlled by the StorageTek Sun Server software. Optionally, one or more specific LSMs can be specified on the command line with the *lsmID* option (e.g. "siloquery lsm 0,1").

siloquery mount volumeID [volumeID ...]
: Get the status of one or more volumes controlled by the StorageTek Sun Server software and print a list of drives with their statuses. This command is essentially a combination of the "siloquery drive" and "siloquery volume" commands. Optionally, more than one volume can be specified on the command line with the *volumeID* options (e.g. "siloquery mount 993346 666999").

siloquery port [portID ...]

Get the status of each LMU (Library Management Unit) port controlled by the StorageTek Sun Server software. The LMU port is a serial communication line from the StorageTek Sun server to the LMU. Optionally, one or more specific LMU ports can be specified on the command line with the *portID* option (e.g. "siloquery port 0,0").

siloquery request [requestID ...]

Get the status of each StorageTek server request currently being executed by the StorageTek Sun Server software. Optionally, one or more requests can be specified on the command line with the *requestID* option (e.g. "siloquery request 13").

siloquery server

Get the status of the StorageTek Sun Server software. There are no optional arguments to this command.

siloquery volume [volumeID ...]

Get the status of each volume or tape controlled by the StorageTek Sun Server software. Optionally, one or more specific volumes can be specified on the command line with the *volumeID* option (e.g. "siloquery volume 998877 STM321"). Since each LSM can contain nearly 6000 tapes, without the optional arguments large quantities of output may be created.

siloquery uses Remote Procedure Calls (or RPC) to communicate with the StorageTek Sun Server software on a remote Sun workstation. The hostname of this remote Sun is obtained from the environment variable *SILOHOST*. This environment variable should be set with the hostname of the Sun server in users' *.login* or *.profile* files.

SEE ALSO

silodismount(8), siloeject(8), siloenter(8), silomount(8)

FILES

/usr/lib/tape/silodrivelist

file that is used to map CONVEX tape device names to tape names used by the StorageTek Automated Cartridge System.

DIAGNOSTICS

clnttcp_create: RPC: Program not registered

siloenter could not become a client of the StorageTek Sun Server software. Either the SILOHOST environment variable is set to the wrong hostname or the StorageTek software on the Sun is not running.

BUGS

None.

NAME

slattach – attach serial lines as network interfaces

SYOPNSIS

/etc/slattach *ttyname* [{ + | - } { **c** | **e** | **i**} *src-name dst-name* [*baudrate*]

DESCRIPTION

slattach is used to assign a tty line to a network interface, and to define the network source and destination addresses. The *ttyname* parameter is a string of the form "ttyXX," or "/dev/ttyXX." *src-name* and *dst-name* are host names as found in */etc/hosts*. The optional *baudrate* parameter is used to set the speed of the connection. If not specified, the default of 9600 is used. Only the super-user may attach a network interface.

To detach the interface, kill the *slattach* processes. Each attach creates two processes and both must be killed in order to detach from the interface.

FLAGS

The following flags are valid (+ enables, - disables):
e - Enable decompression at input, even if **c** is not set.
c - Compress TCP headers on output, decompress headers on input.
i - Drop ICMP packets

EXAMPLES

```
/etc/slattach tty08 myhost remotehost
/etc/slattach /dev/tty01 myhost remotehost 4800
```

DIAGNOSTICS

Messages indicate if the specified interface does not exist, the requested address is unknown, or the user is not privileged.

SEE ALSO

netstat(1), intro(4n), ifconfig(8), rc(8)

NAME

spu – transfer files between ConvexOS and SPU OS file systems

SYNOPSIS

spu [**–r**] [**–w**] file

DESCRIPTION

spu is used to transfer files between the SPU OS file system and the ConvexOS file system. If the argument **–r** is supplied, the SPU OS file *file* will be written to the standard output of *spu*. If the argument **–w** is used, the standard input of *spu* will be written to the SPU OS file *file*. A full pathname should be used to specify the SPU OS file *file*. Only the superuser may use the **–w** argument with *spu*.

SEE ALSO

spucmd(8)

EXAMPLES

The following are common uses of *spu*.

spu -r /mnt/errlog	cat the error log from SPU
spu -r /mnt/errlog > foo	copy the error log from SPU to foo
spu -w /ioconfig < ./ioconfig	copy new ioconfig to SPU
spu -w /mnt/os/errlog.bu < /dev/null	truncate SPU file "errlog.bu"

NAME

spucmd – execute single SPU OS command from ConvexOS

SYNOPSIS

spucmd command_string

DESCRIPTION

spucmd is used to execute a single shell command under SPU OS from a ConvexOS shell. The command string can be a single command such as "*ls*" or a single command with arguments such as "*ls -aC*" or "*ls -aC /mnt/os*". Multiple shell commands may be entered with one command if the commands are separated by semi-colons and the entire command string is enclosed in quotes. The shell filename generation operators *, ?, and [..] are not supported by *spucmd*.

A copy of the command being executed is output as each command is executed by the SPU shell. This is done by using the *-x* option to *sh*(1). After the command is executed, any output from the shell will be output from *spucmd*. If the SPU shell command has a non-zero exit code, *spucmd* will terminate via an *exit*(-1); otherwise it will terminate with an *exit*(0).

Only the superuser may use this command. There is no way for *spucmd* to send data to the command being executed, but input and output can be redirected to local SPU files. *spucmd* will allow **any** command to be executed on SPU OS. Some command under SPU OS can cause ConvexOS execution to be inhibited or stopped abruptly. Extreme caution is advised when executing commands on the SPU.

Used in conjunction with the *spu*(8) program, *spucmd* can accomplish many useful tasks for the system manager.

spucmd executes commands with a working directory of */mnt/os*.

SEE ALSO

spu(8)

EXAMPLES

The following are common uses of *spucmd*.

spucmd rm /mnt/errlog.bu	remove the old /mnt/errlog overflow file
spucmd df	determine SPU disk space available
spucmd mv /ioconfig /ioconfig.old	move /ioconfig file to /ioconfig.old
spucmd "cd /mnt/os ; ls -aC"	list file in directory /mnt/os
spucmd margin -v	list voltage/clock reading of machine
spucmd "date > /date.out"	save date in SPU file /date.out
spucmd "date 8801072010"	set SPU date to Jan 7, 1988 20:10

RESTRICTIONS

Spucmd times out after a certain about of time. So commands that wait for I/O, like *tail -f*, may not work.

NAME

sticky – file mode sticky bit

DESCRIPTION

Setting the "sticky bit," mode 01000 (see *chmod*(2)), on a directory makes it impossible for a file in that directory to be unlinked by anyone other than its owner (or the superuser), even if that directory is publicly writable.

NOTES

Since the owner-unlink feature is enforced by the local filesystem, it may be unavailable on filesystems mounted via NFS from machines from other vendors or from CONVEX machines running older versions of ConvexOS.

Setting this bit on a CONVEX executable file has no impact on the execution of that program; the old use for this bit (saving text images on the swap volume) is no longer needed due to the CONVEX virtual memory system design.

NAME

sumscripts – accounting summary awk scripts

SYNOPSIS

/etc/connecttime | awk -f **connecttime.awk**

/etc/diskuse | awk -f **diskbygrp.awk**

/etc/diskuse | awk -f **diskbyusr.awk**

/etc/diskuse | cat - *diskuse.out.old* | awk -f **diskmerge.awk**

awk -f **genbyact.awk** *general-format-logfile*

awk -f **genbygrp.awk** *general-format-logfile*

awk -f **genbygrpact.awk** *general-format-logfile*

awk -f **genbyusr.awk** *general-format-logfile*

sa -g | awk -f **sabyact.awk**

sa -g | awk -f **sabygrp.awk**

awk -f **tape.awk** */usr/adm/tp-acct*

DESCRIPTION

The directory */usr/adm/sumscripts* contains the *awk* scripts listed above. The scripts summarize accounting data that is kept in various places throughout the system. Further summarization can be accomplished by piping the output of these scripts through *sort* and *idtoname*.

There are several summary possibilities that are not explicitly provided for. For example, there is no script to summarize data in general log file format by TTY or by user ID-activity ID. The large number of summary possibilities makes it impossible to provide every possible summary script. The scripts which are provided serve as a basis for customized scripts to be created by users as needed.

Many of these scripts either produce as output or accept as input a general log file format. This general log file format allows the same summary scripts to be used for different kinds of data.

DEFINITION: Files are in general log file format if they consist of newline-separated records with space-separated fields, and if each record contains the following fields in the order they are listed:

time	A time for the beginning of an event as returned by *time*(3).
amount or *length*	A number whose meaning depends on the log file. For example, in *lpd-acct*, this field is the number of pages printed.
uid	User ID.
gid	Group ID.
aid	Activity ID.
device	A device, i.e. TTY, printer, or tape unit.
[*other*]	This optional field contains information which is specific to each log file. The scripts which operate on files that are in general log file format echo this information without doing any processing on it.

The printer log */usr/adm/lpd-acct* is in general log file format. The tape allocation log */usr/adm/tp-acct* may be converted to general log file format by using the awk script *tape.awk*. The log files */usr/adm/wtmp* and */usr/adm/bill-acct* may be combined into a single file in general log file format by running the *connecttime* utility and piping its output through the *connecttime.awk* script.

A description of each summary accounting *awk* script is given below.

connecttime.awk	Converts the output of *connecttime* to general log file format.
diskbygrp.awk	Summarizes the output of *diskuse* by group.
diskbyusr.awk	Summarizes the output of *diskuse* by user.
diskmerge.awk	Takes two versions of *diskuse* output and merges them together, creating totals for each user and group. The resulting output is in a format consistent with the output of *diskuse*, so that the same data can be run through *diskmerge.awk* repeatedly. The typical use of this script is to keep track of disk usage over time by taking old disk usage data and adding current data to it. The units produced are in kilobyte-time. For example, if this was done once a week, then the resulting data would be in units of kilobyte-weeks.
genbyact.awk	Summarizes data in general log file format by activity.
genbygrp.awk	Summarizes data in general log file format by group.
genbygrpact.awk	Summarizes data in general log file format by group-activity.
genbyusr.awk	Summarizes data in general log file format by user.
sabyact.awk	Summarizes the output of "*sa -g*" by activity.
sabygrp.awk	Summarizes the output of "*sa -g*" by group.
tape.awk	Converts from tape log format (*/usr/adm/tp-acct*) to general log file format.

EXAMPLE

```
awk -f /usr/adm/sumscripts/genbygrp.awk /usr/adm/lpd-acct | idtoname -g
```

The awk script */usr/adm/sumscripts/genbygrp.awk* will summarize the line printer usage data in */usr/adm/lpd-acct*(5) by group ID. This output is then piped through *idtoname*(1) to convert the group IDs in the first field to group names. Sample output is given below.

```
zero      60.000
hardware  29.000
test      352.000
os        2.000
```

SEE ALSO

idtoname(1), sort(1), lpd-acct(5), tp-acct(5), connecttime(8), diskuse(8), pac(8), sa(8), "Accounting" chapter in the *Managing ConvexOS* doumentation set.

NAME

swapon – specify additional device for paging and swapping

SYNOPSIS

/etc/swapon –a
/etc/swapon name ...

DESCRIPTION

Swapon is used to specify additional devices on which paging and swapping are to take place. The system begins by swapping and paging on only a single device so that only one disk is required at bootstrap time. Calls to *swapon* normally occur in the system multi-user initialization file */etc/rc* making all swap devices available, so that the paging and swapping activity is interleaved across several devices.

Normally, the **–a** argument is given, causing all devices marked as "swap" swap devices in **/etc/fstab** to be made available.

The second form gives individual block devices as given in the system swap configuration table. The call makes only this space available to the system for swap allocation.

SEE ALSO

swapon(2), init(8)

FILES

/dev/d??b normal paging devices

BUGS

There is no way to stop paging and swapping on a device. It is therefore not possible to make use of devices which may be dismounted during system operation.

NAME

sync – update the super block

SYNOPSIS

/bin/sync [**-v**]

DESCRIPTION

Sync executes the *sync* system primitive. *Sync* can be called to ensure all disk writes have been completed before the processor is halted, in a way not suitably done by *reboot*(8) or *halt*(8).

See *sync*(2) for details on the system primitive.

The *-v* switch causes output to be produced to verify the execution of sync.

SEE ALSO

sync(2), fsync(2), halt(8), reboot(8), update(8)

NAME

sysex – system exerciser that simulates stressful load conditions

SYNOPSIS

sysex

or

sysex *functional-unit count ...*

DESCRIPTION

The system exerciser provides a convenient method of generating a system load that is at least equivalent to a full complement of users running a variety of tasks that are I/O and computation intensive.

Sysex can be used in either a *default* or a *specific* mode. The *default* mode can be specified by simply entering **sysex**, causing one iteration of a default exercise to be performed.

As an alternative, individual functional units can be exercised by explicitly including them in the command. Multiple iterations are specified by **sysex** *functional-unit count*, where *count* is the number of iterations desired. This is the *specific* mode. The functional units that are currently supported are:

memory, **tty**, **os**, **cpu**, **tape1**, **tape2**, ..., **tape16**, **disk**, **disk1**, **disk2**, ...,**disk16**, **hsp**.

Note that **tape[1-16]** and **disk[1-16]** refer to the different tape and disk drives attached to the computer, while **disk** refers to the disk subsystem independent of the number of drives attached. Specify one of the **disk[1-16]** functional units to ensure that a particular drive is exercised.

You must always indicate the number of iterations when using the *specific* mode. As an example,

sysex memory 25 cpu 50 disk1 75 tape1 75

will exercise the corresponding functional units for the specified number of iterations.

The *default* mode exercises a basic configuration consisting of one tape drive and one disk drive. This configuration is defined in a script called **default**, which can be modified to exercise a particular mix of peripherals. This script should be checked before using the *default* mode to ensure that the desired default action will occur.

PREPARATION

Be sure that

1) your current working directory is the exerciser directory */sys/test/sysex*

2) you are logged in as **root**

3) the peripherals you want to exercise are present.

No other preparation is necessary unless you are going to exercise the **tape1**, **tape2**, **disk**, or **hsp** functional units. Each tape drive to be exercised should have a scratch tape (with a write-ring) mounted on it, and the drive should be on-line. The **/tmp** partition should have room for at least 7000K bytes before **disk** is used. For exercising the **hsp** functional unit, an FSE board set must be installed in the HIA. Refer to the CONVEX HIA USER'S GUIDE (081-000111-000). **Hsp** is an optional product; for more information, contact your CONVEX sales representative.

OPERATIONS and RESULTS

All execution is placed in background mode, except for the tty portion, which must be run in the foreground. There should be at least two terminals attached to the machine. The tty exercise should be performed on one terminal while another is used for the other exercises.

When an error is encountered, a description of the problem will be placed in a specified file, and the exercising of that functional unit is terminated. Otherwise, no output is generated except for a message after each iteration of a particular exercise.

EXERCISER COMPONENTS

The functionality of the CPU is exercised with a collection of computationally intensive routines performing mathematical calculations, using both scaler and vector techniques. Trigonometric and logarithmic functions are exercised over a wide range of values, and are calculated by both iteration and series expansion for comparison purposes.

Memory exercising is accomplished by a program that generates memory accesses across the entire logical address space. Blocks of memory throughout the address space are initialized to a particular value, and then accessed to make sure that the memory actually *remembered* the value.

Disk drives are exercised in two ways. One (**disk[1-16]**) involves reading random sectors from a drive, the other (**disk**) with the copying of a very large directory to another disk partition. Tape drives are exercised in a manner very similar to the disk drives. Large amounts of tape I/O will be generated, and the functions unique to tape drives will be exercised, including moving backwards and forwards between blocks, rewinding and moving to the end-of-file mark.

The ConvexOS operating system will be exercised by a set of programs that utilize system calls dealing with interprocess communication, I/O operations and file system manipulation. The tty interface will be exercised by moving large amounts of I/O through the tty subsystem.

BUGS

A copy of the system exerciser must be made for each person using it.

Only one **disk** exercise can be performed at a time, even when multiple copies of the system exerciser are created. The **disk[1-16]** exercises are not affected; that is,

sysex disk1 50 disk2 50

is correct usage.

Specifying the same functional unit more than once with the same copy of the exerciser is not handled correctly. For example,

sysex cpu 5 cpu 10

causes the two exercises to interfere with one another. This problem would still occur if they were separated into two commands.

FILES

/sys/test/sysex	the directory containing the exerciser.

NAME

sysgen – tool to aid in building customized operating system images

SYNOPSIS

sysgen [**-b**] *sysgen_file*

DESCRIPTION

Sysgen uses the specified configuration file (*sysgen_file*) and other input files to generate makefiles and include files which can be used to build tailored versions of the *vmunix*, *hsp*, *iop*, and *viop* operating system images for the specified configuration. It allows sites who have a **ConvexOS** source license to customize their systems to better meet the needs of their own site. Sites which only have a binary license can also use *sysgen* but have less freedom in the customization of their systems. *Sysgen* can be used to add user written device drivers to either source or binary systems.

In addition to the configuration file, *sysgen* also uses a set of files to determine which source files are needed to generate a system. These input files can be augmented by a configuration specific set of files that give alternate files for a specific machine (see the FILES section below).

Sysgen should be run from the **sysgen** subdirectory of the system source (usually /sys/sysgen). Typically, system managers use capitalized system names, like GOLLUM, as the names for *sysgen_files*. *Sysgen* will create the *../sysgen_file* , *../sysgen_file*_**hsp**, *../sysgen_file*_**iop**, and *../sysgen_file*_**viop** directories (if they don't already exist). These directories are used as build directories when running the *Makefiles* which *sysgen* creates.

The output of *sysgen* consists of a number of files which are placed in the directories mentioned above. These files are: *Makefiles*, files used by *make*(1) to build the system; a set of header files which contain the number of various devices that will be compiled into the system (*device_name*.h); and a set of swap configuration files which contain definitions for the disk areas to be used for the root file system and for swapping.

After running *sysgen*, it is necessary to run "make depend" in each of the directories where the new makefiles were created. After running "make depend" you should run "make" in each directory in order to actually build the operating system images.

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

As distributed with ConvexOS 8.1 and above, sysgen uses different compiler versions than previous versions of sysgen. To use the new release of sysgen under versions of ConvexOS prior to 8.1, the **-b** *flag must be specified.*

FILES

/sys/sysgen/makefile.object	generic makefile for *vmunix* image
/sys/sysgen/makefile.ccu	generic makefiles for CCU images
/sys/sysgen/files.ccu	list of CCU specific files
/sys/sysgen/files.ccu.c?	list of CPU specific CCU specific files
/sys/sysgen/devices.c?	name to major device mapping files
/sys/sysgen/pseudo_devices	list of valid pseudo-devices
/sys/sysgen/controllers	list of valid controller names and attributes
/sys/sysgen/units	list of valid unit names and attributes
/sys/sysgen/GOLLUM	system GOLLUM specification file
/sys/GOLLUM	CPU build directory
/sys/GOLLUM_iop	IOP build directory
/sys/GOLLUM_hsp	HSP build directory
/sys/GOLLUM_viop	VIOP build directory

SEE ALSO

See the *CONVEX System Generation Guide* for an example of a system build. See the *CONVEX Guide to User Written Device Drivers* for more information about adding device drivers to a system.

RESTRICTIONS

Sysgen causes the resulting version number of /vmunix to be set to zero. The *vers*(1) command should be used to set a new version number.

NOTES

In ConvexOS 9.1 and above, "make depend" in the CPU build directory will issue several warnings about non-existant **.dep* files. These warnings can be ignored.

Also in ConvexOS 9.1 and above, the */sys/sysgen/files* file has been made obsolete. To add non-standard code to vmunix (such as user written device drivers), please refer to the *ConvexOS V9.1 Release Notice.*

BUGS

The line numbers reported in error messages are usually off by one.

NAME

syslogd – log systems messages

SYNOPSIS

/usr/etc/syslogd [**-f***configfile*] [**-m***markinterval*] [**-d**]

DESCRIPTION

syslogd reads and logs messages into a set of files described by the configuration file */etc/syslog.conf*. Each message is one line. A message can contain a priority code, marked by a number in angle braces at the beginning of the line. Priorities are defined in <*sys/syslog.h*>. *syslogd* reads from the UNIX domain socket */dev/log* and from the Internet domain socket specified in */etc/services*. (UNIX is a registered trademark of UNIX System Laboratories, Inc.)

syslogd configures when it starts up and whenever it receives a hangup signal. Lines in the configuration file have a *selector*, to determine the message priorities to which the line applies, and an *action*. The *action* field is separated from the selector field by one or more tabs or spaces.

Selectors are semicolon separated lists of priority specifiers. Each priority has a *facility* describing the part of the system that generated the message, a dot, and a *level* indicating the severity of the message. Symbolic names may be used. An asterisk selects all facilities. All messages of the specified level or higher (greater severity) are selected. More than one facility may be selected using commas to separate them. For example:

*.emerg;mail,daemon.crit

selects all facilities at the *emerg* level and the *mail* and *daemon* facilities at the *crit* level.

Known facilities and levels recognized by *syslogd* are those listed in *syslog*(3) without the leading "LOG_". The additional facility "mark" has a message at priority LOG_INFO sent to it every 20 minutes (this may be changed with the **-m** flag). The "mark" facility is not enabled by a facility field containing an asterisk. The level "none" may be used to disable a particular facility. For example,

*.debug;mail.none

sends all messages *except* mail messages to the selected file.

The second part of each line describes where the message is to be logged if this line is selected. There are four forms:

- A filename (beginning with a leading slash). The file will be opened in append mode.
- A hostname preceded by an at sign ("@"). Selected messages are forwarded to the *syslogd* on the named host.
- A comma separated list of users. Selected messages are written to those users if they are logged in.
- An asterisk. Selected messages are written to all logged-in users.

Blank lines and lines beginning with "#" are ignored.

For example, the configuration file:

kern,mark.debug	/dev/console
*.notice;mail.info	/usr/spool/adm/syslog
*.crit	/usr/adm/critical
kern.err	@ucbarpa
*.emerg	*
*.alert	eric,kridle
*.alert;auth.warning	ralph

logs all kernel messages and 20 minute marks onto the system console, all notice (or higher) level messages and all mail system messages except debug messages into the file */usr/spool/adm/syslog*, and all critical messages into */usr/adm/critical;* kernel messages of error severity or higher are forwarded to ucbarpa. All users are informed of any emergency messages; the users "eric" and "kridle" are informed of any alert messages; and the user "ralph" is informed of any alert message or any warning message (or higher) from the authorization system.

The flags are:

-f Specify an alternate configuration file.

-m Select the number of minutes between mark messages.

-d Turn on debugging.

syslogd creates the file */etc/syslog.pid*, if possible, containing a single line with its process id. This can be used to kill or reconfigure *syslogd*.

To bring *syslogd* down, it should be sent a terminate signal (e.g. kill `cat /etc/syslog.pid`).

FILES

/etc/syslog.conf	the configuration file
/etc/syslog.pid	the process id
/dev/log	name of the UNIX domain datagram log socket

SEE ALSO

logger(1), syslog(3)

NAME

syspic – report system, disk, TTY, and network activity in windows

SYNOPSIS

syspic [**-p** *picture*] [**-f** *picfile*] [**-i** *interval*] [**-c** *count*] [**-a**] [**-u** *unixbin*]

DESCRIPTION

syspic is an interactive monitor that periodically reports statistics about system performance. Statistics are grouped into entities called "pictures" that consist of a few lines of information at the top of the screen and windows on the rest of the screen. Each window consists of a header and some statistics. *syspic* highlights numbers that are unusually large or small.

The command line switches, which may be given in any order, are:

-a This switch has meaning only when the picture chosen is *network*. Normally, daemons that listen for connections, such as the printer daemon, are not listed in the *network* picture. The **-a** switch causes these daemons to be listed.

-c *count* If *count* is positive, *syspic* automatically exits after *count* screen updates have been completed. If *count* is 0, *syspic* keeps updating until stopped. The default value for *count* is 0. Note that if *count* is 1, information that is computed as the change in some parameter since the previous update is not shown.

-f *picfile* Specifies a user-defined picture file. This is different from the **-p** switch in that the picture file need not reside in */usr/lib/syspic/*.syspic*, so a user may have his or her own picture files that need not clutter up */usr/lib/syspic*.

-i *interval* Specifies how many seconds to wait between each screen update. Five seconds is a good choice, because some statistics within the system are gathered every five seconds. The default is five seconds.

-p *picture* Specifies which picture to display information about. Usual values for *picture* are **disk**, **disk1**, **disk2**, **kluster**, **network**, **proc**, **proc20**, **syscalls**, **system**, **tty**, and **vm**; the default picture is *system*. Any *.syspic* file that resides in directory */usr/lib/syspic* is treated as a picture and can be specified with the **-p** switch; this allows each site to have its own set of commonly-used pictures.

-u Specifies an alternate executable file. The default is */vmunix*.

The following keys are interpreted by *syspic*:

<SPACE>

<RETURN> Force *syspic* to update the screen and restart the timer.

<CTRL-L> Replot the screen.

- Freeze the screen. When the screen is frozen, no automatic updating will take place. *syspic* still interprets keys.

The *SIGINT* signal, usually generated by **<CTRL-C>** or **<DEL>**, is used to exit *syspic*. Additionally, *syspic* can be interrupted and later resumed with the *SIGTSTP* signal, which is typically generated by **<CTRL-Z>** from *csh*.

The default set of pictures is:

disk The *disk* picture reports the activities of the first 32 currently-mounted disks (striped and non-striped), CCUs, CPU usage, and the size distribution of disk transfers. The disks are displayed in alphabetical order.

disk1 This picture displays the same information as the *disk* picture, except instead of the size distribution of disk transfers, information on sixteen additional disks is

presented (for a total of 48 disks.)

disk2 This is the same as the *disk1* picture, but displays information about disks 16 through 63. This is only useful for very large hardware configurations.

kluster This overview window reports statistics on various size distributions from disk transfers, pageins from a vnode, pageins from the swap device, and pageins from the zero fill device. Other data, such as memory usage, CPU usage, and buffered I/O are also displayed.

network When the *network* picture is first invoked, a list of connections to and from the host machine is displayed. New connections are added to the list and connections that have closed are deleted from it. If there is more than one page of connections, the number keys (0-9) may be used to select a page. The command-line switch **-a** may be used to specify that daemons listening for connections, such as the printer daemon, be included on the list of connections. A connection may be chosen for closer inspection by pressing the letter (a - o) that appears to the left of the connection's entry in the list of connections; **<ESCAPE>** returns back to the list of connections.

proc The top half of this picture contains information on processes, memory, and per-CPU usage. The *proc* picture is the only default picture that shows CPU usage broken down into a per-CPU basis; all other CPU usage windows are shown as calculated averages among all existing processors. The bottom half of this picture contains a display similar to *ps aux* (see the *ps*(1) command) for the top ten processes, sorted by percentage of CPU utilization.

proc20 The *proc20* picture displays the top twenty processes, ranked by percentage of CPU utilization.

syscalls The *syscalls* picture shows execution counts for each of the 50 or so most frequently executed system calls and shows the total number of system calls per second. This picture is good for getting a breakdown of which system calls are contributing to an unusually high number of system calls and for looking at a single application's short-term or long-term system-call usage trend.

system The *system* picture is an overview of system activity which gives information about many phases of system performance including virtual memory, CPU, tape, CCU, disk, trap, and network I/O. A system manager can use this picture to get a feel for how the system is loaded and to watch for trouble spots.

tty The *tty* picture is intended to be used as a tool for evaluating the performance of a system's terminal subsystem. The system manager can use the information displayed by the *tty* picture to make intelligent decisions about which terminal controllers are the best candidates for high traffic and high baud rate terminal lines such as UUCP ports or laser printer ports. It can also be used to find "ringing" terminal ports (ports that are slowing the system down due to spurious characters generated by noise on a terminal line.) This picture initially displays information for the terminal lines belonging to controllers *ca0* through *ca3*. However, information for other controllers may be viewed by selecting another page of data. At the bottom of the screen, there is a message stating how to get to another page.

vm The *vm* picture gives an overview of various activities related to virtual memory, including faults, paging, buffered I/O, and a virtual memory summary window. With the introduction of multi-processor machines, the virtual memory system was largely redesigned, obscuring the fact as to whether physical I/O for virtual memory activities actually took place. The lower levels of the buffer cache now maintain the page cache, and the buffer cache has no idea whether an

I/O operation is being done for paging or for read/write system calls. The best general indicator of how the cache is working is the page hit rate, shown in both the *Buffered I/O* and the *VM Summary* windows.

In addition to the default set of pictures, users may define their own pictures. The best way to create a picture file is by mimicking one of the default picture files in */usr/lib/syspic/*.syspic*. The following is a description of *.syspic* file format, **which is subject to change**:

The first line is optional; if present, it consists of the word "standard" which causes the picture to contain a standard set of information such as current time and load average (see below). This is followed by a line containing the picture name, followed by 0 or more lines consisting of a column number, row number, and window name, separated by a space. The upper left hand corner of the screen is designated as column 0, row 0; column and row are used to specify the upper left hand corner of a window. See the RESTRICTIONS section of this manual page for further information.

There are some entries in the default picture files with "Help" in their names. These entries are not really windows — they exist either for future use or to provide instructions about which keys to press.

Any window that is accessible only by selecting a network connection from the *Connections* window may not be used in a user-defined picture.

All default pictures display two standard lines at the top of the screen. These lines contain the following information:

Current Time The last time a screen update occurred.

Load Average Load averages averaged over the last 1 minute, 5 minutes, and 15 minutes. Load average is defined as the number of jobs in the run queue.

Picture Title The title of the picture as it appears in the picture file. If there is sufficient space on the screen, the host name of the machine is prepended to the picture title.

Reboot Time When the machine was last rebooted.

Up Elapsed time since the last reboot.

Users The number of users currently logged on.

In addition to the two lines at the top of the screen which are present in all default pictures, each of these pictures displays a set of windows in the lower area of the screen. The following table lists the windows that comprise each picture.

disk	*disk1*	*kluster*	*network*	*proc*
Buffered I/O	Buffered I/O	Buffered I/O	CCU Busy	Memory Mb.
CCU Busy	CCU Busy	CPU Usage	CPU Usage	Per-CPU Usage
CPU Usage	CPU Usage	DQ In	Connection	Processes
Calls	Calls	DQ Out	Connections	PS AU
Disk 0-15	Disk 0-15	Memory Mb.	Data Transfer	
DQ In	Disk 16-31	Paging	Mbuf Stats	
DQ Out	Processes	Processes	Network I/O	
Processes	Disk 32-47	Swap	Parameters	
Disk 16-31		Vnode	Recv. Sequence	
		Zfod	Retransmit	
			Send Sequence	
			TCP Stats	

TCP Timers

proc20	*syscalls*	*system*	*tty*	*vm*
PS AU 20	System Calls	Buffered I/O	ca0	Buffered I/O
		CCU Busy	ca1	CPU Usage
		CPU Usage	ca2	Faults
		Faults	ca3	Memory Mb.
		Memory Mb.		Paging
		Network I/O		Processes
		Paging		Swap
		Path. Cache		VM Summary
		Processes		Vnode
		TTY Totals		Zfod
		Tape, Mb/s		

The *disk2* picture is identical to the *disk1* picture, but displays information about disks 16 through 63.

The windows are:

Buffered I/O This window shows parameters related to buffered user I/O and the buffer cache. **Mb/s** shows the average amount of buffered I/O in megabytes per second, measured by the kernel routine *rwip*. **Latency** is the average amount of time in milliseconds that a request to the buffered I/O system (e.g., *rwip*) took and is computed as time of completion minus time started. **Page Hits** is the hit ratio (from 0-100%) on the page cache. The page cache is a least-recently-used replaced cache containing both file system data and executable images. Each page is separately encached by file system and physical block address. The page cache effectively supercedes the buffer cache of previous releases. **Buffer Hits** is the hit ratio (from 0-100%) that a file system I/O request found the desired buffer pages already attached to a buffer header. Buffer headers are required by the device drivers for doing I/O. The desired pages may be found for an I/O request in the page cache (with a resulting page hit), even though they are not attached to a buffer header (resulting in a buffer miss). Buffer hits cannot occur without a corresponding page hit. **Bufcache** is the effective size of the buffer cache in megabytes. It is calculated by counting the number of pages attached to buffer headers. **Deleted Wr.** is the amount of file system data in megabytes that was thrown away because the file(s) that contained it were deleted before they were written to the disk. Temporary files are often not written to the disk at all. **Deferred Wr.** is the amount of file system data in megabytes that had been scheduled to be written to the disk but had the write deferred to some later time because another request for access to the data was made while the I/O request was on the disk queue.

ca0, ca1, ca2, ..., caF

Each "ca" window lists the teletypes connected to a separate ACM-001 or ACM-002 terminal controller. The number of characters input and output since the last screen update are displayed for each terminal line. The last line of each window, labeled "+", indicates the total number of characters input and output summed over all lines on the controller.

TTY lines that are not hooked up to the system have their data fields labeled with "-". These TTY lines do not have terminal controllers to hook up to and

thus cannot ever be active unless another terminal controller is added to the system's configuration.

Some TTY lines contain blank data fields. This indicates that this port has done no I/O since the system was last booted and differs from a value of 0, which means that the port has done I/O in the past but has done no I/O since the last screen update. A TTY line with a "-" in its data field has no terminal controller to interface to, whereas a TTY line with a blank data field has a terminal controller to interface to, but just happens not to have done any I/O since the system was last booted.

Calls Shows the number of system calls per second.

CCU Busy Shows what percent of the time each CCU (channel control unit) on the system is busy. Here, busy is determined by sampling at the CCU's clock interrupt frequency to see if any work is currently being done. For example, in an IOP this is determined by whether or not there is a task currently executing.

Connection Active only when a network connection has been chosen from the "Connections" window. This window is not like other windows because it does not have a label and a border; it consists solely of a selected connection's entry as it appears in the list of active connections. If a connection becomes closed, this line displays "CLOSED".

Connections List of active connections. Each connection's **local address**, **foreign address**, and **state** are listed. Also shown are number of bytes buffered in the receive and send queues for this connection, under the columns **recv Q** and **send Q**. When there is more than one page of connections, the number keys (0 - 9) may be used to move to another page. Pressing the letter that appears to the left of a connection yields another page of information which includes the connection-dependent windows **Data Transfer**, **TCP Timers**, **Send Sequence**, **Recv. Sequence**, and **Retransmit**. This page also contains other windows that are not connection-dependent.

If a connection is closed, "CLOSED" appears in the **Connections** window, and no further updating is done of the connection-dependent windows. The other windows continue to update as usual because they do not rely on the connection's existence. Once a connection has been closed, it cannot be reopened.

Restriction: a specific connection may not be selected from the *Connections* window in a user-defined picture.

CPU Usage A breakdown of percentage of usage of CPU time. **User** is the time spent running normal priority user processes. **User (n)** is the time spent running low priority ("niced") processes. **System** is the time spent running system code, and **Idle** is the amount of time that the CPU has nothing to do.

Disk 0-15 Transfer rate and average latency of disk devices 0 through 15. The average latency, which has units of milliseconds, is computed as the time a request finished minus the time a request was originally queued. By definition, average latency includes the amount of time the request was in device queue. (However, since stripe devices are logical and not physical, there is no corresponding hardware device queue. Thus stripe devices don't have this additional time added in to their computation of latency.) The last line of the window, **tot**, shows the combined transfer rate of the disk devices shown in this window. Even though stripe devices have their transfer rates listed in the windows, they

are not included in each window's total. Since stripes are made up of physical drives, each byte would be counted twice if stripes were added to the total transfer rate: once for the physical disk and once for the stripe. High average latencies for all CCUs may mean that the system needs more CCUs. Uneven transfer rates or uneven average latencies may mean that there are enough CCUs, but work is being unevenly distributed between them.

Disk 16-31 Exactly the same as the *Disk 0-15* window (see above), except this window shows information about disk devices 16 through 31.

Disk 32-47 Exactly the same as the *Disk 0-15* window (see above), except this window shows information about disk devices 32 through 47.

Disk 48-63 Exactly the same as the *Disk 0-15* window (see above), except this window shows information about disk devices 48 through 63.

Data Transfer Lists the number of bytes **Sent** and **Rcvd** (received) over the lifetime of the network connection.

DQ In, DQ Out The *DQ In* and *DQ Out* windows show the size distribution of disk transfers. The number of I/O requests (processed during the sample period) that are less than 4k bytes, as well as the number of requests for each size from 4k bytes through 64k bytes, are shown. The distribution shown in *DQ In* is the requests as they are sent to the disk driver. The disk driver coalesce adjacent requests into a single, larger request, which makes I/O more efficient. The resulting request-size distribution is shown in *DQ Out*.

Faults Trap/interrupt rate averages per second over the last 5 seconds. **Interrupts** is the number of interrupts per second not including the clock. **System Calls** is the number of system calls per second. **Context Sw** is the CPU context switch rate (switches/sec) for processes (**p**) and vectors (**v**).

Mbuf Stats Displays how many message buffers are in use in the system. **In use** shows the number of regular mbufs in use; **Free** is the number of regular mbufs on the free list. **Pages** represents the number of type 1 mbuf clusters in use, and **Free Pages** is the number of type 1 mbuf clusters on the free list. A regular mbuf is a 512-byte buffer. A type 1 cluster is an 8192-byte buffer. The sizes of regular and cluster mbufs are implementation-dependent and may change in future releases of ConvexOS.

Memory Mb. Information about the usage of virtual and real memory. This information is displayed in megabytes. **Virt.** represents the total amount of virtual memory that might require paging to swap. **Real** memory is memory that is in use by processes and is actually in core (as opposed to memory that exists "virtually" on disk). **Free** memory is the amount of memory currently not in use.

Network Buffers Details how many message buffers are in use in the system. The number of buffers/pages in use and in existence is displayed. This window is no longer displayed on any of the default pictures, but similar information is available in the *Mbuf Stats* window.

Network I/O Shows activity taking place on local area networks attached to the system. Data is displayed for up to two network interfaces, two columns per interface. The numbers in the first column are accrued since the last screen update; the numbers in the second column are totals since the machine has been up. **In Packets** is the number of packets received over the network. **In Errors** is the number of receive errors. **Out Packets** is the number of packets transmitted over the network. **Out Errors** is the number of transmit errors. **Collisions** is the number of packet collisions that occur.

Paging Information about page faults, paging activity, and swaps. These are five-second averages and are displayed in units per second. **Reclaims** are pages that are freed but taken back by the process from which they were freed before being allocated anywhere else. Only pages paged to swap are included in this rate. Reclaims also include text pages that are given up by a process that exited but which are taken back by a new instance of the same program. **Page Ins** are faults that resulted in pages paged in, regardless of the number of pages or whether the pages were found in the cache or had to be read from the disk/network. **Page Outs** are the number of times the pager tried to page from a region, regardless of the number of pages paged out or whether the pages had to be written to the disk/network. **Pgin** is the rate of megabytes paged in per second as a result of "Page Ins" faults, regardless of whether the pages were found in the cache or had to be read in from the disk/network. **Pgout** is the rate of megabytes paged out per second as a result of "Page Outs" region scans, regardless of whether the pages had to be written to permanent storage. **Scan Rate** is the rate at which the pageout daemon is running through physical memory and freeing pages; also described as the number of megabytes of virtual memory scanned per second during "Page Outs" requests in order to produce "Pgout" megabytes of real memory. As the scan rate goes up, so does the number of pages freed (the number of reclaims will probably go up, too). Pages that are still free when the pageout daemon gets back to them are paged out. **Swap Ins** is the number of processes whose swap-priority-penalty has been removed. Processes are not actually swapped, but rather their priority is lowered for some time proportional to the amount of memory paged. **Swap Outs** is the number of processes partially swapped. These processes may still run, but their priority remains very low until their swap-priority-penalty is removed.

Parameters **Max seg size** is what this host believes is the maximum number of bytes a network packet can contain, not including the packet header. **Flags** contains other state information. If **Force** is set, the TCP software sends a byte even though it normally would not.

Path. Cache **Pathname Cache** shows the percentage of successful lookups ("hits") made to the pathname cache. **Calls/Hits** displays the number of calls versus hits to the pathname cache.

Per-CPU Usage Gives the same information as the *CPU Usage* window, except that here the information is shown with multiple columns, one for the total (or average) over all of the CPUs, followed by data on each individual processor in the system.

Processes **Runnable** processes are those trying to use the CPU. Processes in **Disk Wait** are awaiting the completion of disk (or other short-term I/O) operations, including paging and file requests. Processes in **Page Wait** are waiting for free memory pages, including memory for paging and file requests. **Sleeping** processes are those processes that are runnable or short sleepers (< 20 seconds). **Swapped** processes are those whose priority is lowered because they have been partially swapped.

PS AU The information in this window is very similar to output from the command *ps aux*, but it displays only the top 10 processes, ranked by "%CPU" (see *ps*(1)).

PS AU 20 This is the same as *PS AU*, except it displays the top 20 processes.

Recv. Sequence **Wind** (window) is the number of bytes of buffering actually available on the receiving side of a network connection. **Next** is the expected sequence number of the next byte. **Urg** (urgent) is the expected sequence number of the next out-of-band byte. **Adv** (advertised window size) is sent from the receiver to the

transmitter. It is the highest sequence number that the transmitter should send.

Retransmit **Rtt** (round-trip time) is the estimated round-trip time for the last packet sent on the network from this end of a connection. **Srtt** (smooth round-trip time) is computed as an average of the last few values of **Rtt** and is thus less susceptible to spikes. **Idle** is the elapsed time since the last packet was received. **Shift** controls how long to wait before retransmitting a packet. **Rtseq** is the sequence number for the packet currently being timed.

Send Sequence **Wind** (window) is what the transmitter of a network connection thinks the receiver's window size is. The sequence number of the next byte to transmit is given by **Next**. **Urg** (urgent) is the sequence number of the next out-of-band byte to transmit. **Max** is the highest sequence number the transmitter has ever sent. **Una** (unacknowledged) is the first sequence number the transmitter has sent that has not been acknowledged by the receiver. **Seq #** is the first sequence number of a segment. **Ack #** is the sequence number of the last packet which has been acknowledged.

Swap Shows the size distribution of pageins from the swap device. The sizes represented are less than 4k bytes, 4k, then increments of 4k up to 64k.

System Calls When the *syscalls* window is invoked, a sorted list is created that contains the 50 or so system calls that have been executed most often since the machine was last rebooted. Successive screen updates show two figures for each system call: how many times it has been executed since the last screen update, and how many times it has been executed since the machine was last rebooted. The totals shown are computed as the sum of the "normal" and "fast" system call totals. (Note: system call names are truncated to 10 characters.) Each screen update also shows the total number of system calls executed per second; this total takes into account all system calls, including those that do not have explicit listings in this window.

Tape, Mb/s The transfer rates of tape drives *ta0, ta1, ... ta5* are expressed in megabytes per second, along with the sum (**tot**) of these transfer rates. Fields with "-" indicate that, under the current system configuration, there is no tape drive hooked up in that slot.

TCP Stats **Bad Sum** is incremented when a received packet fails the internet checksum algorithm. **Hdr Drop** (header drop) is the number of times a packet was dropped because its header was too small. **Bad Off** is incremented when a packet is smaller than the size of a header or larger than the maximum packet size. **Dupicat** is the number of duplicate data packets received. **Retrans** is the number of data packets retransmitted.

TCP Timers When non-zero, these network connection timers are continuously decreasing. **REXMT** is the re-transmit timer. When the transmitter thinks the receiver's window size is zero, the transmitter periodically probes the receiver to make sure the window size is really zero. **PERSIST** is the amount of time remaining until the next probe. **KEEP** controls when this side of the connection sends data to the other side. This is used to determine whether or not the other machine is up. **2MSL** stands for "2 times max segment lifetime" and is used to delay final shutdown of a connection.

TTY Totals The number of characters input and output are displayed for each ACM-001 and ACM-002 terminal controller. These numbers represent the number of characters input and output since the last screen update, summed over all TTY lines hooked up to each controller. Fields with "-" indicate that, under the current system configuration, there is no TTY controller hooked up in that slot. The

final row, labeled "+", represents the sum of all ACM-001 and ACM-002 terminal controllers.

VM Summary The *VM Summary* window displays the following information per *syspic* sample period: number of page faults, number of write faults, number of PTEs copied as a result of a fork, number of pages copied as a result of a write fault, number of pages filled from a vnode, number of pages zero filled, number of pages filled from the swap device, number of pages written to a vnode, number of pages written to the swap device. The VM summary also displays the hit rate on the page cache; this rate includes both file and paging I/O.

Vnode Shows the size distribution of pageins from a vnode (i.e. regular file). Size notation is the same as for *DQ In* and *DQ Out*. The most common occurrences of these pageins are program text and data pages. The requests may have been satisfied by encached pages in which case no I/O was done, or it may have been read from the file.

Zfod Shows the size distribution of pageins from the zero-fill-on-demand device. The most common occurrence of a zero fill page fault is from the bss segment. A single fault often returns a cluster of pages, hopefully saving the time of additional faults.

FILES

/vmunix	kernel name list
/dev/mem	kernel data values
*/lib/kernsyms/symdata_**	kernel symbol addresses
/usr/lib/syspic/.syspic*	standard *syspic* display specification files
/dev/r{d|adu|,st}-ctl*	disk control devices used to obtain disk data

NOTE

The execution of *syspic* itself can affect the statistics that *syspic* measures.

RESTRICTIONS

Some information is computed as the change in some parameter since the screen was last updated. Information of this type is displayed beginning on the second screen update. As a result, if a *count* of 1 is specified using the **-c** switch, such information is blank upon exit of *syspic*.

syspic won't run on screens smaller than 24 rows by 80 columns.

Because of a restriction in the Maryland Window Library, *syspic* cannot use the 80th column on some terminals; anyone designing their own picture should make sure that their picture does not use the 80th column.

syspic uses entries in */etc/mtab* to locate mounted drives. These entries are used to build the name of the drives' control devices. Thus, the block-special devices are assumed to be in the */dev* directory and to have CONVEX-supported names.

BUGS

The way CPU time is computed causes the number of interrupts reported in the *Faults* window to be reported as "-1" occasionally.

In the *network* picture, there is no way to see the *TCP Stats* window except by looking at a specific connection, even though the *TCP Stats* window shows information that is connection-independent.

syspic makes assumptions about device names that are not always valid.

Sometimes the output gets corrupted, particularly if keys are pressed while output is being displayed on the terminal. To clean up a terminal that is in a bad state, try replotting the screen or re-entering *syspic*.

NAME

talkd, in.talkd, otalkd in.otalkd – talk daemon

SYNOPSIS

/usr/etc/in.talkd [–ttimeout]

/usr/etc/in.otalkd

DESCRIPTION

talkd is the server used by the *talk*(1) program. It is used to establish the connection between users and machines, perform the *talk* requests, and handle communications termination and errors.

talkd is invoked as needed by *inetd*(8C), and times out if inactive for two minutes. The **–t** option can be specified to use a different timeout length than the default. The timeout length is specified in minutes.

otalkd is the server used by the *otalk*(1) program. It uses an older, less reliable talk protocol.

SEE ALSO

talk(1), services(5), inetd(8C)

BUGS

The *otalk* protocol is architecture dependent, and cannot be relied upon to work between CONVEX machines and machines of other makers.

NAME

telnetd – DARPA TELNET protocol server

SYNOPSIS

/etc/telnetd

DESCRIPTION

telnetd is a server that supports the DARPA standard **TELNET** virtual terminal protocol. *telnetd* is invoked by the internet server (see *inetd*(8)), normally for requests to connect to the **TELNET** port as indicated by the */etc/services* file (see *services*(5)).

telnetd operates by allocating a pseudo-terminal device (see *pty*(4)) for a client, then creating a login process which has the slave side of the pseudo-terminal as **stdin**, **stdout**, and **stderr**. *telnetd* manipulates the master side of the pseudo-terminal, implementing the **TELNET** protocol and passing characters between the remote client and the login process.

When a **TELNET** session is started up, *telnetd* sends **TELNET** options to the client side indicating a willingness to do *remote echo* of characters, *suppress go ahead*, *remote flow control*, and to receive *terminal type*, *terminal speed*, and *window size* information from the remote client. If the remote client is willing, the remote terminal type is propagated in the environment of the created login process. The pseudo-terminal allocated to the client is configured to operate in "cooked" mode, and with XTABS and CRMOD enabled (see *tty*(4)).

telnetd is willing to *do*: *echo*, *binary*, *suppress go ahead*, and *timing mark*. *telnetd* is willing to have the remote client *do*: *binary*, *terminal type*, *terminal speed*, *window size*, *toggle flow control*, and *suppress go ahead*.

SEE ALSO

telnet(1)

BUGS

Some **TELNET** commands are only partially implemented.

Because of bugs in the original 4.2 BSD *telnet*(1), *telnetd* performs some dubious protocol exchanges to try to discover if the remote client is, in fact, a 4.2 BSD *telnet*(1).

Binary mode has no common interpretation except between similar UNIX-based operating systems. (UNIX is a registered trademark of UNIX System Laboratories, Inc.)

The terminal type name received from the remote client is converted to lower case.

telnetd never sends **TELNET** *go ahead* commands.

NAME

timed – time server daemon

SYNOPSIS

/usr/etc/timed [-t] [-M] [-n network] [-i network]

DESCRIPTION

timed is the time server daemon and is normally invoked at boot time from the *rc*(8) file. It synchronizes the host's time with the time of other machines in a local area network running *timed*(8). These time servers will slow down the clocks of some machines and speed up the clocks of others to bring them to the average network time. The average network time is computed from measurements of clock differences using the ICMP timestamp request message.

The service provided by *timed* is based on a master-slave scheme. When *timed*(8) is started on a machine, it asks the master for the network time and sets the host's clock to that time. After that, it accepts synchronization messages periodically sent by the master and calls *adjtime*(2) to perform the needed corrections on the host's clock.

It also communicates with *date*(1) in order to set the date globally, and with *timedc*(8), a timed control program. If the machine running the master crashes, then the slaves will elect a new master from among slaves running with the **–M** flag. A *timed* running without the **–M** flag will remain a slave. The **–t** flag enables *timed* to trace the messages it receives in the file /usr/adm/timed.log. Tracing can be turned on or off by the program *timedc*(8). *timed* normally checks for a master time server on each network to which it is connected, except as modified by the options described below. It will request synchronization service from the first master server located. If permitted by the **–M** flag, it will provide synchronization service on any attached networks on which no current master server was detected. Such a server propagates the time computed by the top-level master. The **–n** flag, followed by the name of a network which the host is connected to (see *networks*(5)), overrides the default choice of the network addresses made by the program. Each time the **–n** flag appears, that network name is added to a list of valid networks. All other networks are ignored. The **–i** flag, followed by the name of a network to which the host is connected (see *networks*(5)), overrides the default choice of the network addresses made by the program. Each time the **–i** flag appears, that network name is added to a list of networks to ignore. All other networks are used by the time daemon. The **–n** and **–i** flags are meaningless if used together.

FILES

/usr/adm/timed.log	tracing file for timed
/usr/adm/timed.masterlog	log file for master timed

SEE ALSO

date(1), adjtime(2), gettimeofday(2), icmp(4P), timedc(8),
TSP: The Time Synchronization Protocol for UNIX 4.3BSD, R. Gusella and S. Zatti

NAME

timedc – timed control program

SYNOPSIS

/usr/etc/timedc [command [argument ...]]

DESCRIPTION

timedc is used to control the operation of the *timed* program. It may be used to:

- measure the differences between machines' clocks,
- find the location where the master time server is running,
- enable or disable tracing of messages received by *timed*, and
- perform various debugging actions.

Without any arguments, *timedc* will prompt for commands from the standard input. If arguments are supplied, *timedc* interprets the first argument as a command and the remaining arguments as parameters to the command. The standard input may be redirected causing *timedc* to read commands from a file. Commands may be abbreviated; recognized commands are:

? [command ...]

help [command ...]
: Print a short description of each command specified in the argument list, or, if no arguments are given, a list of the recognized commands.

clockdiff host ...
: Compute the differences between the clock of the host machine and the clocks of the machines given as arguments.

trace { on | off }
: Enable or disable the tracing of incoming messages to *timed* in the file */usr/adm/timed.log*.

quit
: Exit from timedc.

Other commands are included for use in testing and debugging *timed*; the help command and the program source may be consulted for details. If the source is not available then these commands are of limited utility.

FILES

/usr/adm/timed.log tracing file for timed
/usr/adm/timed.masterloglog file for master timed

SEE ALSO

date(1), adjtime(2), icmp(4P), timed(8),
TSP: The Time Synchronization Protocol for UNIX 4.3BSD, R. Gusella and S. Zatti

DIAGNOSTICS

?Ambiguous command	abbreviation matches more than one command
?Invalid command	no match found
?Privileged command	command can be executed by root only

NAME

tpconfig – configure the tape system

SYNOPSIS

tpconfig [*command*]

DESCRIPTION

The system administrator runs this program to set up a database which keeps track of the available tape drives and their attributes. (The file */etc/tapecap* is no longer used, and the tape attributes it controlled are now controlled by *tpconfig*.)

If no arguments are given, *tpconfig* acts like a shell and prompts for commands until exited with ^D. If arguments are given, *tpconfig* interprets the arguments as a single command and exits.

The following commands are recognized, and can be abbreviated to their minimum unambiguous length:

Add ACcess_drive Group_set *group_list type:unit*
Add ACcess_drive User_set *user_list type:unit*

These commands add users or groups to a set that determines who is allowed to access a particular tape drive.

The magic character "*" represents all users or all groups. Users and groups can be specified by either their name in */etc/passwd* or */etc/group*, or by the numeric id surrounded by brackets (e.g., *[0]* is the same as *root*).

Add ALlocate_drive Group_set *group_list type:unit*
Add ALlocate_drive User_set *user_list type:unit*

These commands add users or groups to a set that determines who is allowed to allocate a particular tape drive without mounting a tape.

The magic character "*" represents all users or all groups. Users and groups can be specified by either their name in */etc/passwd* or */etc/group*, or by the numeric id surrounded by brackets (e.g., *[0]* is the same as *root*).

Add Bypass_labels Group_set *group_list type:unit*
Add Bypass_labels User_set *user_list type:unit*

These commands add users or groups to a set that determines who is allowed to bypass label processing for a particular drive.

The magic character "*" represents all users or all groups. Users and groups can be specified by either their name in */etc/passwd* or */etc/group*, or by the numeric id surrounded by brackets (e.g., *[0]* is the same as *root*).

Add DRive *type:unit* [**Nocontrol**] [**Timeout=***N*]

Adds a new tape drive to the database. The drive is identified by the *type*, which can be any string that doesn't contain white space or a colon, and by the *unit* which is an integer unit number. Examples: mt:0, exabyte:0.

If **Nocontrol** is given, then this drive is not controlled by the tape system. By default, the drive is controlled by the tape system.

The timeout specifies how long, in minutes, a drive can be idle before it is taken from the user. The default timeout is 60 minutes.

Add Label *label_type daemon*

Adds a new label type to the database. The label type is identified by the string *label_type*. *daemon* is a full path to the daemon that handles this label type. The only label types currently supported are **ansi** and **ibm**, which use **/usr/lib/tape/ansidaemon** and **/usr/lib/tape/ibmdaemon** respectivly..

Add Node *path* [**Speed**=*string*] [**Density**=*string*] [**Rewind**] [**Buffered**] *type:unit*

Adds a new node to a drive in the database. The node is identified by the complete path to the special file (e.g., */dev/rmt8*), and the drive is identified by the *type* and *unit*.

The speed is any string that describes the speed of this node (e.g., 200ips).

The density is any string that describes the density of this node (e.g., 6250).

If **Rewind** is specified, then this node is a rewind device. The default is no-rewind.

If **Buffered** is specified, then this node will use internal buffering. The default is unbuffered.

Delete ACcess_drive Group_set *group_list type:unit*
Delete ACcess_drive User_set *user_list type:unit*

These commands delete users or groups from a set that determines who is allowed to access a particular tape drive.

The magic character "*" represents all users or all groups. Users and groups can be specified by either their name in */etc/passwd* or */etc/group*, or by the numeric id surrounded by brackets (e.g., *[0]* is the same as *root*).

Delete ALlocate_drive Group_set *group_list type:unit*
Delete ALlocate_drive User_set *user_list type:unit*

These commands delete users or groups from a set that determines who is allowed to allocate a particular tape drive without mounting a tape.

The magic character "*" represents all users or all groups. Users and groups can be specified by either their name in */etc/passwd* or */etc/group*, or by the numeric id surrounded by brackets (e.g., *[0]* is the same as *root*).

Delete Bypass_labels Group_set *group_list type:unit*
Delete Bypass_labels User_set *user_list type:unit*

These commands delete users or groups from a set that determines who is allowed to bypass label processing for a particular drive.

The magic character "*" represents all users or all groups. Users and groups can be specified by either their name in */etc/passwd* or */etc/group*, or by the numeric id surrounded by brackets (e.g., *[0]* is the same as *root*).

Delete Drive *type:unit*

Deletes a drive from the database. All nodes associated with this drive are also deleted.

Delete Label *label_type*

Deletes a label type from the database.

Delete Silohost *<type:unit>*

Deletes the association of a drive to a silo server host of the StorageTek Automated Cartridge System, commonly referred to as a *tape silo*.

Delete Node *path*

Deletes a node from the database.

SEt ACCess_drive Group_set *group_list type:unit*
SEt ACCess_drive User_set *user_list type:unit*

These commands define the set of users or groups that determines who is allowed to access a particular tape drive. Users and groups can be specified by either their name in */etc/passwd* or */etc/group*, or by the numeric id surrounded by brackets (e.g., *[0]* is the same as *root*).

The magic character "*" represents all users or all groups.

SEt ALlocate_drive Group_set *group_list type:unit*
SEt ALlocate_drive User_set *user_list type:unit*

These commands define the set of users or groups that determines who is allowed to allocate a particular tape drive without mounting a tape. Users and groups can be specified by either their name in */etc/passwd* or */etc/group*, or by the numeric id surrounded by brackets (e.g., *[0]* is the same as *root*).

The magic character "*" represents all users or all groups. Users and groups can be specified by either their name in */etc/passwd* or */etc/group*, or by the numeric id surrounded by brackets (e.g., *[0]* is the same as *root*).

SEt Bypass_labels Group_set *group_list type:unit*
SEt Bypass_labels User_set *user_list type:unit*

These commands define the set of users or groups that determines who is allowed to bypass label processing for a particular drive.

The magic character "*" represents all users or all groups. Users and groups can be specified by either their name in */etc/passwd* or */etc/group*, or by the numeric id surrounded by brackets (e.g., *[0]* is the same as *root*).

SEt Control ON|OFf *type:unit*

This command puts a drive under the control of the tape system.

SEt Default Density *density*

This command defines the default tape density. Unless the user requests a different tape density with the *tpmount* command, this tape density is used.

SEt Default Drive *type*

This command defines the default drive type. Unless the user requests a different type with the *tpmount* command, this type is used.

SEt Default Flags [Rewind|Norewind] [Character|Block|Labeled]

This command sets the default flags for *tpmount* requests. **Rewind** and **Norewind** select whether the device will or will not rewind on close. **Character**, **Block**, and **Labeled** select the default device mode.

SEt Default Speed *speed*

This command defines the default drive speed. Unless the user requests a different speed with the *tpmount* command, this speed is used.

SEt No_default Density

This command undefines the default drive density.

SEt No_default Speed

This command undefines the default drive speed.

SEt Queueing Enabled|Disabled

When queueing is enabled, all tape mount and drive allocation requests are passed to *opreq*(1) for an operator to handle or fulfill. Otherwise, drive allocations are handled by *tpdaemon* and tape mounts are not allowed (ie, the users must mount their own tapes).

SEt Silohost *hostname drive_string type:unit*

This command associates this drive with a tape silo server host. *hostname* is the network name of the workstation serving the tape silo. *drive_string* is the string that defines the location of the drive within the tape silo.

SEt Timeout *N type:unit*

This command changes the timeout value for the specified drive.

SHow All

Shows everything.

SHow DEfaults

Shows all of the default values.

SHow DRive *type:unit*

Shows all information associated with the indicated drive, including the nodes.

SHow Labels

Shows all of the available label types.

SHow Node *path*

Shows all information associated with the indicated node.

SNapshot [*file*]

The command creates an ASCII snapshot of the current configuration database. If *file* is not given, the snapshot is written to standard output. The snapshot consists of a set of *tpconfig* commands that will create a database equivalent to the current database.

FILES

/usr/lib/tape/config.db tape configuration database

SEE ALSO

silomount(1), silodismount(1), siloquery(1), siloenter(1), siloeject(1), tpmount(1), tpunmount(1), tpattr(1), tplabel(1), tpunlabel(1), tpqueue(1), tpwait(1), tape(3), mt(1), ansidaemon(8), tpdaemon(8), ibmdaemon(8)

NAME

tpdaemon – tape daemon

SYNOPSIS

/usr/lib/tape/tpdaemon [**-debug**] [**-syslog**] [**-offl**]

DESCRIPTION

tpdaemon is the main daemon for the tape subsystem. It is normally started from */etc/rc.local* without any of the options selected.

Options:

-debug
Enables logging of debug messages.

-syslog
Causes log messages to go to standard output instead of to syslog. This is only used for debugging purposes.

-offl Causes *tpdaemon* to merely rewind a tape when it would normally take the tape offline. This is only used for debugging purposes.

FILES

/usr/lib/tape/config.db	tape configuration database
/usr/lib/tape/tpdaemon.lock	tape daemon lock file

SEE ALSO

tpmount(1), tpunmount(1), tpattr(1), tplabel(1), tpunlabel(1), tpqueue(1), tpwait(1), tape(3), ansidaemon(8), ibmdaemon(8), tpconfig(8), mt(1)

NAME

trpt – transliterate protocol trace

SYNOPSIS

trpt [**–a**] [**–s**] [**–t**] [**–f**] [**–j**] [**–p** hex-address] [system [core]]

DESCRIPTION

trpt interrogates the buffer of TCP trace records created when a socket is marked for "debugging" (see *setsockopt*(2)), and prints a readable description of these records. When no options are supplied, *trpt* prints all the trace records found in the system grouped according to TCP connection protocol control block (PCB). The following options may be used to alter this behavior.

–a in addition to the normal output, print the values of the source and destination addresses for each packet recorded.

–s in addition to the normal output, print a detailed description of the packet sequencing information.

–t in addition to the normal output, print the values for all timers at each point in the trace.

–f follow the trace as it occurs, waiting a short time for additional records each time the end of the log is reached. If *trpt* is killed with a *SIGTERM*, it will flush its output buffers before exiting.

–j just give a list of the protocol control block addresses for which there are trace records.

–p show only trace records associated with the protocol control block, the address of which follows.

The recommended use of *trpt* is as follows. Isolate the problem and enable debugging on the socket(s) involved in the connection. Find the address of the protocol control blocks associated with the sockets using the **–A** option to *netstat*(1). Then run *trpt* with the **–p** option, supplying the associated protocol control block addresses. The **–f** option can be used to follow the trace log once the trace is located. If there are many sockets using the debugging option, the **–j** option may be useful in checking to see if any trace records are present for the socket in question. The

If debugging is being performed on a system or core file other than the default, the last two arguments may be used to supplant the defaults.

FILES

/vmunix
/dev/kmem

SEE ALSO

setsockopt(2), netstat(1c)

DIAGNOSTICS

"no namelist" when the system image doesn't contain the proper symbols to find the trace buffer; others which should be self explanatory.

BUGS

Should also print the data for each input or output, but this is not saved in the trace record.

The output format is inscrutable and should be described here.

NOTES

trpt is an optional product; for more information, contact your CONVEX sales representative.

NAME

update – periodically update the super block

SYNOPSIS

/etc/update

DESCRIPTION

Update is a program that executes the *sync*(2) primitive every 30 seconds. This insures that the file system is fairly up to date in case of a crash. This command should not be executed directly, but should be executed out of the initialization shell command file.

SEE ALSO

sync(2), sync(8), init(8), rc(8)

BUGS

With *update* running, if the CPU is halted just as the *sync* is executed, a file system can be damaged. A fix would be to have *sync*(8) temporarily increment the system time by at least 30 seconds to trigger the execution of *update*. This would give 30 seconds grace to halt the CPU.

NAME

uucico, uucpd – transfer files queued by uucp or uux

SYNOPSIS

/usr/lib/uucp/uucico [**–d***spooldir*] [**–g***grade*] [**–r***role*] [**–R**] [**–s***system*] [**–x***debug*] [**–L**] [**–t***turnaround*]

/etc/uucpd

DESCRIPTION

uucico performs the actual work involved in transferring files between systems. *uucp*(1C) and *uux*(1C) merely queue requests for data transfer that *uucico* processes.

The following options are available.

–d*spooldir*
Use *spooldir* as the spool directory. The default is */usr/spool/uucp*.

–g*grade* Only send jobs of grade *grade* or higher on this transfer. The grade of a job is specified when the job is queued by *uucp* or *uux*.

–r*role* *role* is either 1 or 0; it indicates whether *uucico* is to start up in master or slave role, respectively. 1 is used when running *uucico* by hand or from *cron*(8). 0 is used when another system calls the local system. Slave role is the default.

–R Reverse roles. When used with the **–r1** option, this tells the remote system to begin sending its jobs first, instead of waiting for the local machine to finish.

–s*system*
Call only system *system*. If **–s** is not specified, and **–r1** is specified, *uucico* attempts to call all systems for which there is work. If **–s** is specified, a call is made even if there is no work for that system. This is useful for polling.

–x*debug* Turn on debugging at level *debug*. Level 5 is a good start when trying to find out why a call failed. Level 9 is very detailed. Level 99 is absurdly verbose. If *role* is 1 (master), output is normally written to the standard message output *stderr*. If *stderr* is unavailable, output is written to */usr/spool/uucp/AUDIT/system*. When *role* is 0 (slave), debugging output is always written to the AUDIT file.

–L Only call "local" sites. A site is considered local if the device-type field in *L.sys* is one of LOCAL, DIR or TCP.

–t*turnaround*
Use *turnaround* as the line turnaround time (in minutes) instead of the default 30. If *turnaround* is missing or 0, line turnaround is disabled. After *uucico* has been running in slave role for *turnaround* minutes, it attempts to run in master role by negotiating with the remote machine. In earlier versions of *uucico*, a transfer of many large files in one direction would hold up mail going in the other direction. With the turnaround code working, the message flow is more bidirectional in the short term. This option only works with newer *uucico*'s and is ignored by older ones.

If *uucico* receives a **SIGFPE** (see *kill*(1)), it toggles the debugging on or off.

uucpd is the server for supporting UUCP connections over networks. *uucpd* listens for service requests at the port indicated in the "uucp" service specification; see *services*(5). The server provides login name and password authentication before starting up *uucico* for the rest of the transaction.

uucico is commonly used either of two ways: as a daemon run periodically by *cron*(8) to call out to remote systems, and as a "shell" for remote systems who call in. For calling out periodically, a typical line in *crontab* would be:

```
0    *    *    *    *    /usr/lib/uucp/uucico -r1
```

This runs *uucico* every hour in master role. For each system that has transfer requests queued, *uucico* calls the system, logs in, and executes the transfers. The file *L.sys*(5) is consulted for information about how to log in, while *L-devices*(5) specifies available lines and modems for calling.

For remote systems to dial in, an entry in the *passwd*(5) file must be created, with a login "shell" of *uucico*. For example:

```
nuucp:Password:6:1::/usr/spool/uucppublic:/usr/lib/uucp/uucico
```

The UID for UUCP remote logins is not critical, as long as it differs from the UUCP Administrative login. The latter owns the UUCP files, and assigning this UID to a remote login would be an extreme security hazard.

FILES

/usr/lib/uucp/	UUCP internal files/utilities
/usr/lib/uucp/L-devices	Local device descriptions
/usr/lib/uucp/L-dialcodes	Phone numbers and prefixes
/usr/lib/uucp/L.aliases	Hostname aliases
/usr/lib/uucp/L.cmds	Remote command permissions list
/usr/lib/uucp/L.sys	Host connection specifications
/usr/lib/uucp/USERFILE	Remote directory tree permissions list
/usr/spool/uucp/	Spool directory
/usr/spool/uucp/AUDIT/*	Debugging audit trails
/usr/spool/uucp/C./	Control files directory
/usr/spool/uucp/D./	Incoming data file directory
/usr/spool/uucp/D.hostname/	Outgoing data file directory
/usr/spool/uucp/D.hostnameX/	Outgoing execution file directory
/usr/spool/uucp/CORRUPT/	Place for corrupted C. and D. files
/usr/spool/uucp/ERRLOG	UUCP internal error log
/usr/spool/uucp/LOGFILE	UUCP system activity log
/usr/spool/uucp/LCK/LCK..*	Device lock files
/usr/spool/uucp/SYSLOG	File transfer statistics log
/usr/spool/uucp/STST/*	System status files
/usr/spool/uucp/TM./	File transfer temp directory
/usr/spool/uucp/X./	Incoming execution file directory
/usr/spool/uucppublic	Public access directory

SEE ALSO

uucp(1C), uuq(1C), uux(1C), L-devices(5), L-dialcodes(5), L.aliases(5), L.cmds(5), L.sys(5), uuclean(8C), uupoll(8C), uusnap(8C), uuxqt(8C)

D. A. Nowitz and M. E. Lesk, *A Dial-Up Network of UNIX Systems.*

D. A. Nowitz, *Uucp Implementation Description.*

NAME

uuclean – uucp spool directory clean-up

SYNOPSIS

/usr/lib/uucp/uuclean [**-m**] [**-n***time*] [**-p***pre*] [**-d***subdirectory*]

DESCRIPTION

uuclean scans the spool directory for files with the specified prefix and deletes all those that are older than the specified number of hours.

The following options are available:

-p*pre* Scan for files with *pre* as the file prefix. Up to 10 **-p** arguments may be specified.

-n*time* Files whose age is more than *time* hours are deleted if the prefix test is satisfied. (Default *time* is 72 hours.)

-m Send mail to the owner of the file when it is deleted.

-d*subdirectory*
Only the specified subdirectory is cleaned.

This program will typically be run daily by *cron*(8).

FILES

/usr/spool/uucp Spool directory

SEE ALSO

uucp(1C), uux(1C), uucico(8C)

NAME

uulook – monitor inter-system communications

SYNOPSIS

uulook

DESCRIPTION

Uulook displays the end of the ***uucp*** log. It then monitors it, printing any new entries.

Uulook is a one-line shell script:

```
sniff -600 /usr/spool/uucp/LOGFILE
```

FILES

/usr/spool/uucp/LOGFILE

SEE ALSO

sniff(1), tail(1), uucp(1c)

NAME

uupoll – poll a remote UUCP site

SYNOPSIS

uupoll [**–g***grade*] [**–n**] *system*

DESCRIPTION

uupoll is used to force a poll of a remote system. It queues a null job for the remote system and then invokes *uucico*(8C).

The following options are available:

–g*grade* Only send jobs of grade *grade* or higher on this call.

–n Queue the null job, but do not invoke *uucico*.

uupoll is usually run by *cron*(5) or by a user who wants to hurry a job along. A typical entry in *crontab* could be:

```
0    0,8,16  *  *  *  /usr/bin/uupoll ihnp4
0    4,12,20 *  *  *  /usr/bin/uupoll convex
```

This polls *ihnp4* at midnight, 0800, and 1600, and *convex* at 0400, noon, and 2000.

If the local machine is already running *uucico* every hour and has a limited number of outgoing modems, a more elegant approach might be:

```
0    0,8,16  *  *  *  /usr/bin/uupoll -n ihnp4
0    4,12,20 *  *  *  /usr/bin/uupoll -n convex
5    *       *  *  *  /usr/lib/uucp/uucico -r1
```

This queues null jobs for the remote sites at the top of hour; they are processed by *uucico* when it runs five minutes later.

FILES

/usr/lib/uucp/	UUCP internal files/utilities
/usr/spool/uucp/	Spool directory

SEE ALSO

uucp(1C), uux(1C), uucico(8C)

NAME

uusnap – show snapshot of the UUCP system

SYNOPSIS

uusnap

DESCRIPTION

uusnap displays in tabular format a synopsis of the current UUCP situation. The format of each line is as follows:

site N Cmds N Data N Xqts Message

Where "site" is the name of the site with work, "N" is a count of each of the three possible types of work (command, data, or remote execute), and "Message" is the current status message for that site as found in the STST file.

Included in "Message" may be the time left before UUCP can re-try the call and the count of the number of times that UUCP has tried (unsuccessfully) to reach the site.

SEE ALSO

uucp(1C), uux(1C), uuq(1C), uucico(8C)
"UUCP Implementation Guide"

NAME

uuxqt – UUCP execution file interpreter

SYNOPSIS

/usr/lib/uucp/uuxqt [**-x** *debug*]

DESCRIPTION

uuxqt interprets *execution files* created on a remote system via *uux*(1C) and transferred to the local system via *uucico*(8C). When a user uses *uux* to request remote command execution, it is *uuxqt* that actually executes the command. Normally, *uuxqt* is forked from *uucico* to process queued execution files; for debugging, it may also be run manually by the UUCP administrator.

uuxqt runs in its own subdirectory, */usr/spool/uucp/XTMP*. It copies intermediate files to this directory when necessary.

FILES

/usr/lib/uucp/L.cmds	Remote command permissions list
/usr/lib/uucp/USERFILE	Remote directory tree permissions list
/usr/spool/uucp/LOGFILE	UUCP system activity log
/usr/spool/uucp/LCK/LCK.XQT	*uuxqt* lock file
/usr/spool/uucp/X./	Incoming execution file directory
/usr/spool/uucp/XTMP	*uuxqt* running directory

SEE ALSO

uucp(1C), uux(1C), L.cmds(5), USERFILE(5), uucico(8C)

NAME

verify – verify characteristics of system files

SYNOPSIS

verify [**–d** *directory*] [**–i** [**–y**]] [**–k**] [**–V**] database ...
verify [**–c**] [**–r**] [**–s**] [**–T**] [**–v**] [**–V**] **-m** directory

DESCRIPTION

verify checks a set of system files to ensure that they have the characteristics specified in a database file. If the *database* filename is neither an absolute pathname nor a relative path that begins with ./ or ../, then the file is assumed to be in the */usr/lib/verify* directory. The *database* file lists the files to be verified and the characteristics to verify for each file. Characteristics that may be checked are file type, owner, group, and mode. In addition, for symbolic links the "pointed to" file name may be checked.

In its normal mode of operation, *verify* produces a list of differences found. Each characteristic of a file that differs from what was specified in the database is noted in the output. With the **–i** option, the owner, group, and mode may be changed interactively to match the values given in the database.

verify is a tool to aid in the diagnosing of system problems and is intended to be used by root. Supplied with each release are a set of database files. These files are located in the directory */usr/lib/verify* and cover virtually all of the standard system files. Each database file contains entries for the files that make up an optional product. Look in */usr/lib/verify to see what database files are available.* The format of the database files is described in *verify*(5).

In the second mode of operation, *verify* produces a database of the file tree beginning at *directory*. For each file, the database includes the owner, group, mode, and type (symbolic link, directory, character special, etc). The **–c**, **–r**, **–s**, **–T** and **–v** options can be used to augment the generated information.

OPTIONS

–c Generate checksum information for each file when the **–m** option is used.

-d *directory*
Specify a base directory to prepend to all directory and file names.

–i Run in interactive mode. Every time the owner, group, or mode of a file is found to be incorrect, a prompt is issued offering to correct it. The expected response is either **y** for yes, or **n** for no. A default may also be given as part of the prompt.

–k *verify* normally complains when it discovers a symbolic link that should be a regular file. Specifying this option tells *verify* to follow the link and verify the file that it points to, rather than complain about the link and give up.

–m Create (make) a *verify* database.

–M This option is an alias. Using it is the same as using **–c**, **–m**, **–r**, **–s**, **–T** and **–v**.

–r Generate uid's and gid's for each file instead of usernames and groupnames when the **–m** option is used.

–s Generate file size information for each file when the **–m** option is used.

-T Generate timestamp information for each file when the **-m** option is used. As of ConvexOS 10.0, the timestamp used for binary files is the link timestamp generated by /bin/ld. For scripts and non-executable files, the timestamp is the value of st_mtime as defined in /usr/include/sys/stat.h.

-v Generate version information for each file when the **-m** option is used.

-V Use verbose error tracking when using **-v** and **-T**.

-y A YES answer is assumed in response to all questions posed in the interactive mode. That is, all differences noted are automatically corrected.

DIAGNOSTICS

Each time an attempt is made to change the owner, group, or mode of a file, the attempt is recorded using *syslogd*(8). Successful attempts are logged with level LOG_NOTICE and unsuccessful ones are logged with level LOG_WARNING.

verify exits with a *sysexit* code. Error messages indicate that *verify* is unable to check the current database entry. Warning messages indicate that *verify* uses the default.

FILES

/usr/lib/verify	standard database files
/usr/include/sysexits.h	sysexit codes

SEE ALSO

chgrp(1), chmod(1), chown(8), ls(1), stat(2), syslogd(8), verify(5)

NAME

versatec – versatec filters for use with lpr

SYNOPSIS

/usr/lib/vpf
/usr/lib/vpfW
/usr/lib/vplotf
/usr/lib/vpltdmp
/usr/lib/vpsf

DESCRIPTION

The programs *vpf*, *vpfW*, *vplotf*, *vpltdmp*, and *vpsf* are filters invoked by *lpd*(8) to print to a Versatec Printer/Plotter. *Vpf* and *vpfW* are filters for printing alphanumeric data; *vpfW* prints data to a wide plotter. *Vplotf* is a filter for printing *graph*(1G) output. *Vpltdmp* is a filter for printing *plot*(1G) output. *Vpsf* is a filter for printing alphanumeric data with pages appearing side-by-side, for use with a wide plotter. A typical *printcap* entry for a Versatec Plotter which is 2048 pixels wide might be:

```
ve|vers|versatec|Versatec Plotter 2048 pixels wide:\
    :lp=/dev/pb4:sd=/usr/spool/ved:af=/usr/adm/veacct:\
    :mx#2000:px#2048:py#2200:lf=/usr/spool/vers_log:\
    :if=/usr/lib/vpfW:of=/usr/lib/vpfW:\
    :gf=/usr/lib/vplotf:vt=/usr/lib/vpltdmp:
```

FILES

/usr/tmp/rasterfile containing raster image created by plot(1G)

SEE ALSO

graph(1G), lpr(1), plot(1G), plot(3X), spline(1G), vpr(1)

NAME

vipw – edit the password and password restrictions files

SYNOPSIS

vipw [editor_cmds]

DESCRIPTION

Vipw simultaneously edits the password file and the password restrictions file. To prevent concurrent updating of the files, locks are used to exclude all but one user from editing the password information. *Vipw* sets the appropriate locks, and does any necessary processing after both files are unlocked. If the files are already being edited with *vipw*, then you will be told to try again later.

To prevent separate editing of the password and password restrictions files, *vipw* combines the information into a single editing session. For each user, *vipw* constructs a single line displaying the union of the user's *pwrestrict* file entry and the user's *passwd* file entry. After editing, *vipw* will separate the information, update both files, and rebuild the password and restrictions databases with *mkpasswd*.

Vipw should be the only utility used to update the *pwrestrict* and *passwd* files.

The *vi* editor will be used unless the environment variable EDITOR indicates an alternate editor. Optionally an *editor_cmds* argument can be given to *vipw*. If supplied, the argument will be passed on to the editor as a command line argument. Assuming *vi* is used as the editor, example uses include:

vipw +/foo/ will start editing the created password file at the first line containing the text "foo".

vipw +\$ will start editing at the last line of the created password file.

Vipw checks the syntax of all password entries and tries to point out the specific errors made. The errors may be fixed by running *vipw* again. *Vipw* performs a number of consistency checks on the password entry for *root*, and will not allow a password file with a "mangled" root entry to be installed. *Vipw* further restricts root's login shell to one of the shells listed in */etc/shells*. If */etc/shells* does not exist, the C shell */bin/csh*, or the Bourne shell */bin/sh* are the only acceptable shells for root.

SEE ALSO

chfn(1), chsh(1), passwd(1), passwd(5), pwrestrict(5), mkpasswd(8), nu(8)

FILES

/etc/ptmp -- lock file containing new /etc/passwd
/etc/rtmp -- lock file containing new /etc/pwrestrict
/etc/vtmp -- lock file containing both passwd and pwrestrict fields

CAVEAT

Vipw will complain when a mangled non-root entry has been installed. As mentioned, use *vipw* to re-edit the password files and correct the mistake.

NAME

vvmdaemon – virtual volume manager daemon

SYNOPSIS

vvmdaemon

vvmdaemon –r

DESCRIPTION

Vvmdaemon is the daemon for the Virtual Volume Manager. It is normally started at boot time from */etc/rc.std*.

The first form of *vvmdaemon* does not have the *–r* option. In this form, *vvmdaemon* opens a KRPC channel and registers it with the VVM driver in the kernel. When the VVM driver encounters a disk failure, it notifies *vvmdaemon* via this channel; the daemon then takes steps to reconstruct the lost data to a hot spare disk. The actual reconstruction is done using the *mvst*(8) utility.

If the daemon fails to reconstruct the lost data, user intervention is required. If the lost data has not been reconstructed when another disk partition fails, all data in the striped partition will be lost.

The *vvmdaemon* is also used by the VVM driver to log messages via syslog whenever events in the driver occur. These events are not critical and therefore are not logged to the errlog on the SPU. Examples are parity correction messages from the parity checker in the driver.

The second form of *vvmdaemon* uses the *–r* option. In this form, *vvmdaemon* restarts any *mvst*(8) operations that were not allowed to exit gracefully (for example, those not killed with SIGTERM or those running at the time of a system crash). If a *mvst*(8) operation is not allowed to exit gracefully, the */etc/stripecap* file becomes out of sync with the kernel tables, eventually resulting in data loss. The list of operations that must be restarted is taken from */etc/streconfig*. No other options are valid with the *–r* option.

It is important that *vvmdaemon –r* be run only when initially booting the system (bringing it up from SPU level). For this reason it is run out of */etc/.initrc* and should not be run from the command line.

FILES

/etc/streconfig	Database of outstanding *mvst*(8) operations.
/tmp/vvmdaemon.*	temporary output files.

SEE ALSO

st(4), mvst(8), krpc(2)

DIAGNOSTICS

invalid argument: <offending argument>
the given command line argument is not recognized.

open /etc/streconfig: <error message>
unable to open the */etc/streconfig* file for the given reason.

krpc_open: <error message>
unable to open the krpc channel for the given reason.

open /dev/rst0: <error message>
unable to open */dev/rst0* for the given reason.

there is already another daemon registered
you may not have more than one *vvmdaemon* running at a time.

ioctl /dev/rst0: <error message>
the ioctl on */dev/rst0* failed for the given reason.

close /dev/rst0: <error message>
unable to close */dev/rst0* for the given reason.

fork: <error message>
unable to fork a process for the given reason.

aborting
previous errors were fatal.

The following diagnostics are reported using the *syslog*(3) daemon facility. The level follows in parentheses.

failure: fork: <error message>
unable to fork a process for the given reason (LOG_CRIT).

could not restart the following command:
<mvst command>
this must be done manually with mvst(8)
due to previous errors, the given command could not be restarted (LOG_CRIT).

unable to replace device /dev/d??? in /dev/st?
this must be done manually with mvst(8)
due to previous errors, the given reconstruction could not take place (LOG_CRIT).

failure: restarting the following command:
<mvst command>
the given *mvst*(8) command has begun execution (LOG_NOTICE).

failure: device /dev/d??? in stripe /dev/st? failed
the *mvst*(8) command to perform the given reconstruction has begun execution (LOG_CRIT).

failure: mvst(8) exited with an error
<mvst output>
correct the problem and restart the mvst(8) command manually
the *mvst*(8) command exited with a non-zero status. Manual intervention is necessary to complete the reconstruction process (LOG_CRIT).

failure: mvst(8) command completed successfully
the *mvst*(8) command successfully completed its task (LOG_CRIT).

krpc: <error message>
the *krpc*(2) call failed for the given reason (LOG_ERR).

parity check begun on /dev/st?
the driver has started checking the redundant data on the given device (LOG_NOTICE).

parity check complete on /dev/st?
the driver has finished checking the redundant data on the given device (LOG_NOTICE).

correcting parity block #%d in /dev/st?
the driver found a corrupt redundant data block and is correcting it (LOG_NOTICE).

unknown event: <event number>
the VVM driver sent out an unknown event on the krpc channel (LOG_WARNING).

waitpid: reaping <pid>
vvmdaemon is reaping an exiting child with pid <pid> (LOG_DEBUG).

BUGS

Mvst(8) command buffers are statically allocated at 128 characters. Command lines longer than 128 characters will not be restarted properly.

NAME

xdump – fast incremental filesystem dump

SYNOPSIS

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

DESCRIPTION

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

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

f Place the dump on the next *argument* file instead of the tape. If the name of the file is –, *xdump* writes to standard output. The default is */dev/rmt8*.

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

s The size of the dump tape is specified in feet. The number of feet is taken from the next *argument*. When the specified size is reached, *xdump* will wait for reels to be changed. The default tape size is 2300 feet.

d The density of the tape, expressed in BPI, is taken from the next *argument*. This is used in calculating the amount of tape used per reel. The default is 1600.

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

w Is like *W*, but prints only those filesystems which need to be dumped.

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

c Set size and density defaults for cartridge tape rather than 9-track.

b Take the next *argument* to be the blocking factor for tape records. The default is 10.

l Take the next *argument* as the read latency in disk sectors. This is the number of sectors the disk will rotate through between reads. The default is 17 sectors.

p Take the next *argument* as the amount of data to plan ahead on the reads in units of kilobytes. The default is 600 kilobytes.

m Take the next *argument* as the maximum number of fragments to read in a single read request. The default is 64 frags, or 128kbytes, whichever is smaller.

t Enable the printing of interesting timing statistics after each phase of the dump.

D Take the next *argument* as the minimum number of filesystem fragment sized data buffers to allocate.

G Set defaults for a GCR (6250 bpi) dump. This sets the tape unit to be */dev/rmt16*, the density to 6250, and the blocking factor to 100.

I Puts the tape drive into 100 ips streaming mode. The default is 50 ips.

E Causes *xdump* to write to tape until finished or until end-of-tape is reached. If end-of-tape is

reached, the operator is prompted to change tapes. Even with this option, *xdump* will print an estimate of how many 9-track tapes (not cartridges) will be used for the dump.

xdump requires operator intervention on these conditions: end-of-tape, end-of-dump, tape-write error, tape-open error, or disk-read error. In addition to alerting all operators implied by the **n** key, *xdump* interacts with the operator on *xdump*'s control terminal at times when *xdump* can no longer proceed, or if something is seriously wrong. All questions *xdump* poses **must** be answered by typing "yes" or "no".

xdump tells the operator what is going on at periodic intervals, including estimates of the number of blocks to be written, the number of tapes it will take, the time to completion, and the time to the tape change.

For each filesystem at your site, establish a sequence of full and incremental dumps on a weekly cycle. Because backing up and restoring a filesystem is a time-consuming process, the sequence you select has an impact on the amount of time required to perform backups and to restore files from dump tapes.

One possible backup schedule for performing dumps follows:

Day	**Dump Level**
Monday	level 0: backs up all files in the filesystem (full dump)
Tuesday	level 5: backs up all files changed since Monday's full dump
Wednesday	level 5: backs up all files changed since Monday's full dump
Thursday	level 5: backs up all files changed since Monday's full dump
Friday	level 5: backs up all files changed since Monday's full dump

With this method, restoring files requires only two sets of dump tapes: tapes for the full dump and tapes for the most recent incremental dump.

CAUTION

xdump DIFFERS IN ONE VERY IMPORTANT ASPECT FROM THE ORIGINAL. This version does NOT re-open every file just before it is dumped. If you dump an active file system using this dump, you run a slightly greater risk of dumping blocks which no longer resemble the file to which they used to belong. Guarantees of consistency are valid only for quiescent filesystems.

FILES

/dev/rmt8	default tape unit to dump to
/etc/ddate	old format dump date record (obsolete after **–J** option)
/etc/dumpdates	new format dump date record
/etc/fstab	dump table: filesystems and frequency
/etc/group	to find group *operator*

NOTES

xdump understands that file systems can be larger than two gigabytes in size, and correctly dumps these file systems. The dump tape format has not changed, so files smaller than two gigabytes may be restored on systems that do not support large files.

SEE ALSO

mtio(4), dump(5), fstab(5), restore(8)

DIAGNOSTICS

Many, and verbose. Unlike most other utilities, *xdump* returns a 1 upon normal completion.

BUGS

Sizes are based on 1600 bpi blocked tape; the raw magtape device has to be used to approach these densities.

xdump does not know about the dump sequence, does not keep track of the tapes scribbled on, does not tell the operator which tape to mount when, and does not provide assistance for the operator running *restore*.

If a different tape blocking factor is used, it must also be used with *restore*.

Index

Description	Man Page
compare two files	CMP(1)
Compose C declarations	CDECL(1)
compress and uncompress files and cat them	COMPACT(1)
compute the difference between two times.	DIFFTIME(3)
computer aided instruction about ConvexOS	LEARN(1)
condition command	TEST(1)
configurable pathname variables	PATHCONF(3)
connect — initiate a connection on a socket	CONNECT(2)
connect to a remote system	TIP(1C)
cons — console interface	CONS(4)
console interface	CONS(4)
construct Makefile dependency list	MKDEP(1)
cont — graphics interface	PLOT(3X)
contact — submit a CONVEX problem report	CONTACT(1)
contactcap — system configuration file for contact	CONTACTCAP(5)
continually watch the end of a file	SNIFF(1)
continue — command language	SH(1)
control daemons	DMON_IOCTL(2)
control device	IOCTL(2)
control maximum system resource consumption	GETRLIMIT(2)
control maximum system resource consumption	VLIMIT(3C)
control system log	SYSLOG(3)
controller crashdump utility	CDUMP(1)
controller download utility	CDOWN(1)
controller reset utility	CRESET(1)
conversion program	UNITS(1)
convert and copy a file	DD(1)
convert archives to random libraries	RANLIB(1)
convert ASCII to numbers	ATOF(3)
convert values between host and network byte order	BYTEORDER(3N)
CONVEX assembler and link editor output	A.OUT(5)
CONVEX encode/decode	CCRYPT(1)
copy	CP(1)
copy directory	CPALL(1)
copy file archives in and out	CPIO(1)
core — format of memory image file	CORE(5)
cos — trigonometric functions	SIN(3M)
cosf — trigonometric functions	SIN(3M)
cosh — hyperbolic functions	SINH(3M)
coshf — hyperbolic functions	SINH(3M)
_count — Cray-compatible bit manipulation functions that perform population count leading zero count and parity count.	BITCOUNT(3BIT)
cp — copy	CP(1)
cpall — copy directory	CPALL(1)
cpio — copy file archives in and out	CPIO(1)
cpio — POSIX compatible extended cpio archive file format	CPIO(5)
cpp — C preprocessor	CPP(1)
cpr — print "C" files	CPR(1)

Description	Man Page
_gbit — Cray-compatible bit manipulation functions that get or set the value of a particular group of bits.	BITCHANGE(3BIT)
_gbits — Cray-compatible bit manipulation functions that get or set the value of a particular group of bits.	BITCHANGE(3BIT)
gcvt — output conversion	ECVT(3)
general terminal interface	TERMIOS(4)
general terminal interface	TTY(4)
generate a fault	ABORT(3)
generate terminal pathname	CTERMID(3)
generator of lexical analysis programs	LEX(1)
get a process' activity ID	GETAID(2)
get a process' real and effective group ID	PGETREGID(2)
get a pseudo-teletype device name	GETPTY(3)
get a string from a stream	GETS(3S)
get activity file entry	GETACTENT(3)
get actwho file entry	GETACWENT(3)
get and set options on sockets	GETSOCKOPT(2)
get character or word from stream	GETC(3S)
get configurable system variables	SYSCONF(3)
get current floating point mode	GETFPMODE(3)
get current working directory pathname	GETWD(3)
get descriptor table size	GETDTABLESIZE(2)
get disk description by its name	GETDISKBYNAME(3X)
get entries from name list	NLIST(3)
get entry from password restrictions file	GETPWRESTENT(3)
get file status	CVXSTAT(2)
get file status	LSTAT(2)
get file status	STAT(2)
get file system descriptor file entry	GETFSENT(3X)
get file system descriptor file entry	GETMNTENT(3)
get file system statistics	STATFS(2)
get formatted date and time	FTIME(3C)
get group access list	GETGROUPS(2)
get group file entry	GETGRENT(3)
get group identity	GETGID(2)
get information about parallel resource utilization	CVXPRUSAGE(2)
get information about resource utilization	GETRUSAGE(2)
get information about resource utilization	VTIMES(3C)
get legal user shells	GETUSERSHELL(3)
get login name	GETLOGIN(3)
get name from uid	GETPW(3C)
get name of connected peer	GETPEERNAME(2)
get network entry	GETNETENT(3N)
get network host entry	GETHOSTBYNAME(3N)
get option letter from argv	GETOPT(3)
get password file entry	GETPWENT(3)
get process group	GETPGRP(2)
get process identification	GETPID(2)
get process times	TIMES(3C)

Description	Man Page
_mask — Cray-compatible bit manipulation functions that create a mask of contiguous "1" bits.	BITMASK(3BIT)
_maskl — Cray-compatible bit manipulation functions that create a mask of contiguous "1" bits.	BITMASK(3BIT)
_maskr — Cray-compatible bit manipulation functions that create a mask of contiguous "1" bits.	BITMASK(3BIT)
mass storage disk interface	DA(4)
mblen — multibyte and wide character functions	MBSTOWCS(3)
mbstowcs — multibyte and wide character functions	MBSTOWCS(3)
mbtowc — multibyte and wide character functions	MBSTOWCS(3)
mclear — shared memory synchronization primitives	MSET(3)
mem — main memory	MEM(4)
memchr — string search functions	STRINGSEARCH(3)
memcmp — string comparison functions	STRINGCMP(3)
memcpy — string copy functions	STRINGCPY(3)
memmove — string copy functions	STRINGCPY(3)
memory allocator	MALLOC(3)
memset — string copy functions	STRINGCPY(3)
menu driven online information system	INFO(1)
merge RCS revisions	RCSMERGE(1)
merge — three-way file merge	MERGE(1)
mesg — permit or deny messages	MESG(1)
MIL standard bit manipulation function-like macros.	BITMIL(3BIT)
mkdep — construct Makefile dependency list	MKDEP(1)
mkdir — make a directory file	MKDIR(2)
mkdir — make a directory	MKDIR(1)
mkfifo — make a FIFO special file	MKFIFO(3)
mknod — make a special (character block or fifo) file	MKNOD(2)
mkstemp — make a unique file name	MKTEMP(3)
mkstr — create an error message file by massaging C source	MKSTR(1)
mktemp — make a unique file name	MKTEMP(3)
mktime — date and time manipulation routines	CTIME(3)
mmap — map shared memory into your address space	MMAP(2)
modf — split into mantissa and exponent	FREXP(3)
modify process attributes	MPA(1)
moncontrol — prepare execution profile	MONITOR(3)
monitor — prepare execution profile	MONITOR(3)
monstartup — prepare execution profile	MONITOR(3)
more — file perusal filter for CRT viewing	MORE(1)
mount file system	MOUNT(2)
mount — mount file system	MOUNT(2)
mounted file system table	MTAB(5)
move — graphics interface	PLOT(3X)
move jobs from one line printer spooling queue to another queue	LPMV(1)
move or rename files	MV(1)
move read/write pointer	LSEEK(2)
mpa — modify process attributes	MPA(1)
mqueue — sendmail mail queue directory and queue files	MQUEUE(5)
mremap — change attributes of a memory region in a process' address space	MREMAP(2)

Description	Man Page
rcvtir — IEEE/native floating point mode conversion routines.	RCVTIR(3M)
rdiff — remote diff front end	RDIFF(1C)
rdist — remote file distribution program	RDIST(1)
read a password	GETPASS(3)
read — command language	SH(1)
read from a file	READ(2)
read or write ANSI multifile labeled tapes	ANSITAR(1)
read — read from a file	READ(2)
read scattered data	READV(2)
read value of a symbolic link	READLINK(2)
readdir — directory operations	DIRECTORY(3)
readlink — read value of a symbolic link	READLINK(2)
readnotes — a news system	NOTES(1)
readonly — command language	SH(1)
readv — read scattered data	READV(2)
realloc — memory allocator	MALLOC(3)
rearrange name list	SYMORDER(1)
reboot — reboot system or halt processor	REBOOT(2)
reboot system or halt processor	REBOOT(2)
rebuild the data base for the mail aliases file	NEWALIASES(1)
receive a message from a socket	RECV(2)
re_comp — regular expression handler	REGEX(3)
recv — receive a message from a socket	RECV(2)
recvfrom — receive a message from a socket	RECV(2)
recvmsg — receive a message from a socket	RECV(2)
re_exec — regular expression handler	REGEX(3)
regular expression handler	REGEX(3)
relational database operator	JOIN(1)
remind you when you have to leave	LEAVE(1)
reminder service	CALENDAR(1)
remote diff front end	RDIFF(1C)
remote file copy	RCP(1C)
remote file distribution program	RDIST(1)
remote host description file	REMOTE(5)
remote host phone number data base	PHONES(5)
remote login	RLOGIN(1C)
remote — remote host description file	REMOTE(5)
remote shell	RSH(1C)
remove a directory file	RMDIR(2)
remove a file or directory entry	UNLINK(2)
remove a file system	UNMOUNT(2)
remove columns from a file	COLRM(1)
remove debugger stabs or symbols and relocation bits	STRIP(1)
remove jobs from the line printer spooling queue	LPRM(1)
remove jobs from the tape mount request queue	TPRM(1)
remove jobs spooled by at	ATRM(1)
remove labels from a labeled tape	TPUNLABEL(1)
remove nroff troff tbl and eqn constructs	DEROFF(1)
remove — remove a file or directory entry	UNLINK(2)

Order Number
DSW-333

Document Number
710-015930-001