NAME

adduser − add a new user to the system

SYNOPSIS

adduser *user group home-dir*

EXAMPLES

adduser ast other /usr/ast

How user ast could be added

adduser bin operator /usr/src

How user bin could be added

DESCRIPTION

Adduser adds a new user to the system by making new entries in **/etc/passwd** and **/etc/shadow** for the new user, creating a new home directory, and copying the contents of the template home directory **/usr/ast** into it. The user-id of this new user will be the first free number not less than 10. The password is initially empty, the full name must be set, and the shell is the Bourne Shell, **/bin/sh .** Use *passwd* , *chfn* , and *chsh* to change.

SEE ALSO

login(1), **passwd**(1), **passwd**(5).

NAME

add_route − configure IP routing.

SYNOPSIS

add_route −g gateway [**−d** destination [**−n** netmask *]]* [**−i** ip device]

DESCRIPTION

Add_route is used for manual entry of routes in the IP routing table.

OPTIONS

−g *gateway* specifies the gateway IP address to use.

−d *destination* specifies the destination(s) reached via this gateway.

−n *netmask* specifies a netmask when the destination is a net.

−i *ip device* specifies the ip device.

SEE ALSO

irdp(8), **pr_routes**(8).

AUTHOR

Add_route.c was created August 7, 1991 by Philip Homburg. This manual page by A. S. Woodhull, last revised 13.02.96.

NAME

backup − backup files

SYNOPSIS

backup [**−djmnorstvz**] *dir1 dir2*

OPTIONS

−d At top level, only directories are backed up

−j Do not copy junk: **.Z, *.bak, a.out, core,* etc

−m If device full, prompt for new diskette

−n Do not backup top-level directories

−o Do not copy **.o* files

−r Restore files

−s Do not copy **.s* files

−t Preserve creation times

−v Verbose; list files being backed up

−z Compress the files on the backup medium

EXAMPLES

backup −mz . /f0 # Backup current directory compressed

backup /bin /usr/bin # Backup bin from RAM disk to hard disk

DESCRIPTION

Backup (recursively) backs up the contents of a given directory and its subdirectories to another part of the file system. It has two typical uses. First, some portion of the file system can be backed up onto 1 or more diskettes. When a diskette fills up, the user is prompted for a new one. The backups are in the form of mountable file systems. Second, a directory on RAM disk can be backed up onto hard disk. If the target directory is empty, the entire source directory is copied there, optionally compressed to save space. If the target directory is an old backup, only those files in the target directory that are older than similar names in the source directory are replaced. *Backup* uses times for this purpose, like *make*. Calling *Backup* as *Restore* is equivalent to using the -r option; this replaces newer files in the target directory with older files from the source directory, uncompressing them if necessary. The target directory contents are thus returned to some previous state.

SEE ALSO

tar(1).

NAME

badblocks − put a list of bad blocks in a file

SYNOPSIS

badblocks *block_special* [*block*] ...

EXAMPLES

badblocks /dev/hd1   # Handle bad blocks on */dev/hd1*

badblocks /dev/hd3 310 570 1680
    # Three bad blocks on */dev/hd3*

DESCRIPTION

If a device contains bad sectors, it is important to not have them allocated to important files. This program makes it possible to collect up to 7 bad blocks into a dummy file, so they will not be allocated for a 'real' file. When the program starts up, it asks for a list of bad blocks, unless they are provided as arguments. Then it creates a file whose name is of the form *.Bad_xxxxx*, where *xxxxx* is a pid.

SEE ALSO

readall(1).

NAME

boot − from power on to the login prompt

DESCRIPTION

At power on the machine reads the first sector of the boot device into memory and executes it. This bootstrap code loads **/boot**, the Minix Boot Monitor. The monitor loads the kernel binaries from **/minix**, or the newest file in **/minix** if it is a directory.

The Minix system is now running, the different tasks initialize themselves and control is transferred to the last one, **init**.

Init is the grandparent of all Minix processes, it is responsible for starting login processes on each terminal, but first it runs **/etc/rc**.

/etc/rc checks the state of the system and starts daemons. First it sets the keyboard translation to the mapping in **/etc/keymap** if present, then it reads the time zone from **/etc/timeinfo** followed by a call to **readclock**(8) to set Minix time from the hardware clock. Next the file systems are checked if necessary and the **/usr** file system is mounted.

The system is now ready for multiuser startup, **/etc/rc** starts the **update**(8) and **cron**(8) daemons, and initializes the network services. **/etc/rc** finally recovers crashed editor buffers and cleans out the **tmp** directories.

Init reads **/etc/ttytab** and starts a **getty**(8) for each enabled terminal line to allow a user to log in.

BOOT ENVIRONMENT

Many features of the drivers inside the kernel are controlled by settings in the boot environment. The values of these variables are usually colon or comma separated numbers configuring the driver. **DPETH0 = 300:10** tells the ethernet driver to use I/O address 0x300, interrupt request 10, and the default memory address (0xD0000, values may be omitted) for the first ethernet board. (Note that IRQ 2 is redirected to IRQ 9 on AT's and PS/2's, so use 9 if a device is jumpered for 2.)

Variables that are special to both the monitor and the kernel are described in **monitor**(8). This section lists extra variables or variable settings:

hd = **at** | **bios** | **esdi** | **xt**

Choose the driver that is to be used for the hard disk, in order: IBM/AT (classic AT or newer IDE), BIOS (generic driver), ESDI (some PS/2's), or IBM/XT. By default the first of these drivers that is enabled is used. Most drivers are present in the kernel as distributed, but may be taken out by modifying **/usr/include/minix/config.h**. (An XT should always use the BIOS driver, not the XT driver, because BIOS calls are cheap on an XT. The XT driver can be used on AT machines with an old XT controller.)

DPETH*n* = **on** | **off**

Turn an ethernet board on or off. The driver is by default in "sink" mode for all boards. The sink mode allows one to use the driver without an ethernet board installed. The driver will play /dev/null for that device, i.e. nothing comes in, and anything send out is dropped on the floor. If the board is turned on then the driver will use it to send out packets, if it is turned off then the driver will fail for that board.

DPETH*n* = *I/O-addr*:*irq*:*mem_addr*

Set the I/O address (hex), IRQ (decimal) and memory address (hex) of the *n*-th ethernet board and turn it on. By default they are configured as 280:3:D0000 and 300:5:CC000. The memory address is ignored for the Novell ethernet boards, but may be explicitly set to zero to indicate that the board **is** a Novell ethernet board. You do not need to specify the IRQ with modern Western Digital 8013 compatible ethernet cards, the driver asks the board what its IRQ is. (Note that the default IRQ conflicts with the second serial line, so the serial line is turned off if the ethernet board is configured for IRQ 3.)

DPETH*n***_EA** = *e0*:*e1*:*e2*:*e3*:*e4*:*e5*

Set the ethernet address of the *n*-th ethernet board. The address is normally obtained from the ethernet board, so only in exceptional circumstances is this setting ever needed. (Use the address of the main server if you want a career change.)

AHA0 = *I/O-addr*:*bus-on*:*bus-off*:*tr-speed*
Configure the Adaptec 154xA SCSI host adapter to use the given I/O address (hex), Bus-on time (decimal), Bus-off time (decimal) and transfer speed (hex). The default is 330:15:1:00. The default transfer speed is always 5.0 Mb/s (code 00) ignoring the jumper settings.

sd*n* = *target*,*lun*
Program SCSI disk **sd***n* to have the given target and logical unit number. The target and lun of a tape or other SCSI device may be changed by setting the **sd***n* variable that would be used had it been a disk. So tape device st7 can be set to target 4, lun 1 with sd35=4,1.

MCD = *I/O-addr*:*irq*
I/O address (hex) and IRQ (decimal) of the Mitsumi CD-ROM driver, by default 300:10.

TCP/IP CONFIGURATION

To use TCP/IP you have to compile a kernel with networking enabled, and unless you are running standalone you have to enable the ethernet driver. See the **DPETH***n* boot variable above. The driver supports these ethernet cards: Western Digital 8003, Western Digital 8013, SMC Elite Ultra 16, Novell NE1000, Novell NE2000. Many newer variants of the WD8013, now under the SMC brand, are also supported.

You are likely to use TCP/IP in one of three situations:

Standalone with no connection to a network.

In a small network with no support from a "big" host.

Connected to a large network with address and name servers.

In each situation you need a different set of configuration files.

Standalone

The machine is configured with a fixed IP address: **192.9.200.1**. This is one of the addresses Sun used to give to machines without a registered network address. This address is normally blocked at gateways, so it can do no damage if used in a real net by accident. You need one file, **/etc/hosts**, that should look like this (using the name "darask" as an example):

```
127.0.0.1       localhost
192.9.200.1     darask
```

Small Network

In a network where the Minix machine can't obtain its IP address and name from a different host you need specify the ethernet address to host name translation in the **/etc/ethers** file for use by the RARP daemon. Suppose you have two machines in your network then **/etc/ethers** could look like this:

```
0:0:c0:a:77:23      darask
0:0:c0:a:68:ce      burask
```

Use **hostaddr −e** to find out what the six octet ethernet address of a host is. Use the address as printed: lowercase hex digits, no leading zeros. The **/etc/hosts** file shows their IP addresses:

```
127.0.0.1       localhost
192.9.200.1     darask
192.9.200.2     burask
```

Warning! Do not add ethernet addresses of diskless workstations to your ethers file. A Sun for instance has the stupid habit of booting from the first RARP server that answers, probably your Minix machine...

Large Network

In a network with a central network administration your machine's IP address and name are given by the RARP and name services of the special servers on the network. For a new machine you need to apply for an IP address and host name with your network administrator supplying the ethernet address of your machine. You don't need any configuration files now, the **irdpd** and **nonamed** daemons automatically find a router and a name server.

Note that no knowledge of the IP address or hostname of the Minix machine itself is necessary, it all comes from the RARP and name servers. A series of Minix machines can therefore set up identically. Even if you have no RARP or name servers you can still set them up identically if you list all the Minix hosts in the hosts and ethers files.

Simpler configuration tools

The **rarpd**, **irdpd** and **nonamed** daemons are complex little programs that try to obtain information about their surroundings automatically to tell the machine what its place in the network is. It should come as no surprise that there are simpler utilities to configure a machine. On a memory starved machine it may even be wise to configure a machine statically to get rid of the daemons. The first daemon, **rarpd**, can be replaced by:

> **ifconfig −h** *host-IP-address*

to set the IP address of the machine. Note that this is only necessary if there is no external RARP service. The second daemon **irdpd** can be replaced by setting a static route:

> **add_route −g** *router-IP-address*

(if there is a router.) The last daemon, **nonamed**, can be replaced by an entry in **/etc/resolv.conf** that specifies an external name daemon:

> **nameserver** *nameserver-IP-address*

The **ifconfig** and **add_route** calls can be placed in the file **/etc/rc.net**. The calls to the daemons will have to be edited out of **/etc/rc**. Note that these changes undo all the efforts to make Minix TCP/IP autoconfigurable. Make very sure that all the IP addresses are correct, and that the IP address of your machine is unique. (Mistakenly using the address of a main server will make all other machines look at your machine, and will make all the users of all other machines look at you.)

FILES

/boot	Minix Boot Monitor.
/minix	Kernel image, or directory containing them.
/etc/rc	First of the system initialization files.
/etc/hosts	Name to IP address mapping.
/etc/ethers	Name to ethernet address mapping.

SEE ALSO

monitor(8), **init**(8), **inet**(8), **loadkeys**(8), **readclock**(8), **fsck**(1), **update**(8), **cron**(8), **ttytab**(5), **getty**(8), **hostaddr**(1), **ifconfig**(8), **irdpd**(8), **nonamed**(8), **rarpd**(8), **hosts**(5), **ethers**(5), **set_net_default**(8).

DIAGNOSTICS

Checking File Systems.

> If the system has crashed then **fsck** is called for the root and /usr file systems. It is wise to reboot if the root file system must be fixed.

Finish the name of device to mount as /usr: /dev/

> If the name of the /usr file system has not been set in /etc/fstab. You can type a device name, say **fd0**.

hostaddr: unable to fetch IP address

> TCP/IP misconfiguration. The RARP may have failed because the ethernet address of the machine is not entered in either the remote or the local ethers file. Either talk to your Network Administrator, or make an ethers and a hosts file.

1.2.3.4 login:

> If you see an IP address instead of a host name then the system failed to translate the IP address. Either talk to your Network Administrator to have the reverse address translation tables fixed, or make a hosts file.

NOTES

The names "darask" and "burask" are names of cities from the Dutch translation of the novel "The Many-Colored Land" by Julian May. The author of this text likes names of hosts to be things that contain people, like cities and ships.

BUGS

Indefinite hangs are possible if I/O addresses or IRQ's are wrong. A driver may babble about addresses and IRQ's, but that does not mean that what it says is true, it may just be configured that way. It is very difficult to find peripherals on a PC automatically, and Minix doesn't even try.

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)

NAME

checkhier − check the directory hierarchy

SYNOPSIS

checkhier

DESCRIPTION

Checkhier checks a number of files and directories that make up the top level file system hierarchy. The output of the command is a script that could be applied to fix things like bad mode, wrong owner or group, etc.

The script should never be executed without checking. I might be better to examine the differences oneself and to fix any problems by hand.

Checkhier must be run by the superuser.

SEE ALSO

chmod(1), **chown**(8), **hier**(7).

DIAGNOSTICS

The exit code is 0 if all checks out right, otherwise a script is output and the exit code is 1.

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)

NAME

chown − change owner

SYNOPSIS

chown [−R] *owner*[:*group*] *file* ...

OPTIONS

−R Change directory hierarchies

EXAMPLES

chown ast file1 file2 # Make *ast* the owner of the files

chown −R ast:other dir

Change the owner and group of all files in dir

DESCRIPTION

The owner field (and optionally group field) of the named files is changed to *owner* (i.e., login name specified) and *group* . Alternatively, a decimal uid(gid) may be specified instead of a user name. Only the superuser may execute this command.

SEE ALSO

chgrp(1), **chmod**(1), **ls**(1), **chown**(2).

NAME

cron − clock daemon

SYNOPSIS

cron

EXAMPLES

/usr/bin/cron # Use absolute path in */etc/rc*

DESCRIPTION

Cron is clock daemon. It is typically started up by including the command */usr/bin/cron* in the */etc/rc* file. Once started, *cron* puts itself in the background, so no & is needed. It runs forever, sleeping most of the time. Once a minute it wakes up and examines */usr/lib/crontab* to see if there is any work to do. If there is, the work is done. The entries of */usr/lib/crontab* contain 6 elements each. Some examples follow:

Min	**Hr**	**Dat**	**Mo**	**Day**	**Command**
*	*	*	*	*	/usr/bin/date >/dev/log #print date every minute
0	*	*	*	*	/usr/bin/date >/dev/log #print date on the hour
30	4	*	*	1-5	/bin/backup /dev/fd1 #do backup Mon-Fri at 0430
30	19	*	*	1,3,5	/etc/backup /dev/fd1 #Mon, Wed, Fri at 1930
0	9	25	12	*	/usr/bin/sing >/dev/log #Xmas morning at 0900 only

SEE ALSO

at(1).

NAME

elvprsv - Preserve the the modified version of a file after a crash.

SYNOPSIS

elvprsv ["-*why elvis died***"] /tmp/***filename***...**
elvprsv -R /tmp/*filename***...**

DESCRIPTION

elvprsv preserves your edited text after *elvis* dies. The text can be recovered later, via the *elvprsv* program.

For UNIX-like systems, you should never need to run this program from the command line. It is run automatically when *elvis* is about to die, and it should be run (via /etc/rc) when the computer is booted. THAT'S ALL!

For non-UNIX systems such as MS-DOS, you can either use *elvprsv* the same way as under UNIX systems (by running it from your AUTOEXEC.BAT file), or you can run it separately with the "-R" flag to recover the files in one step.

If you're editing a file when *elvis* dies (due to a bug, system crash, power failure, etc.) then *elvprsv* will preserve the most recent version of your text. The preserved text is stored in a special directory; it does NOT overwrite your text file automatically.

elvprsv will send mail to any user whose work it preserves, if your operating system normally supports mail.

FILES

/tmp/elv*
: The temporary file that *elvis* was using when it died.

/usr/preserve/p*
: The text that is preserved by *elvprsv*.

/usr/preserve/Index
: A text file which lists the names of all preserved files, and the names of the /usr/preserve/p* files which contain their preserved text.

BUGS

Due to the permissions on the /usr/preserve directory, on UNIX systems *elvprsv* must be run as superuser. This is accomplished by making the *elvprsv* executable be owned by "root" and turning on its "set user id" bit.

If you're editing a nameless buffer when *elvis* dies, then *elvprsv* will pretend that the file was named "foo".

AUTHOR

Steve Kirkendall
kirkenda@cs.pdx.edu

NAME

fdisk − partition a hard disk [IBM]

SYNOPSIS

fdisk [**−h***m*] [**−s***n*] [*file*]

OPTIONS

−h Number of disk heads is *m*

−s Number of sectors per track is *n*

EXAMPLES

fdisk /dev/hd0 # Examine disk partitions

fdisk −h9 /dev/hd0 # Examine disk with 9 heads

DESCRIPTION

When *fdisk* starts up, it reads in the partition table and displays it. It then presents a menu to allow the user to modify partitions, store the partition table on a file, or load it from a file. Partitions can be marked as MINIX, DOS or other, as well as active or not. Using *fdisk* is self-explanatory. However, be aware that repartitioning a disk will cause information on it to be lost. Rebooting the system *immediately* is mandatory after changing partition sizes and parameters. MINIX, XENIX, PC-IX, and MS-DOS all have different partition numbering schemes. Thus when using multiple systems on the same disk, be careful.

Note that MINIX, unlike MS-DOS , cannot access the last sector in a partition with an odd number of sectors. The reason that odd partition sizes do not cause a problem with MS-DOS is that MS-DOS allocates disk space in units of 512-byte sectors, whereas MINIX uses 1K blocks. *Fdisk* has a variety of other features that can be seen by typing *h*.

Fdisk normally knows the geometry of the device by asking the driver. You can use the **−h** and **−s** options to override the numbers found.

SEE ALSO

part(8).

NAME

fingerd, in.fingerd − remote user information server

SYNOPSIS

finger stream tcp nowait nobody /usr/sbin/in.fingerd in.fingerd
tcpd finger /usr/sbin/in.fingerd 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 or an equally narrow-minded TELNET-protocol user program can result in meaningless attempts at option negotiation being sent to the server, which will foul up the command line interpretation. **Fingerd** should be taught to filter out IAC’s and perhaps even respond negatively (IAC WON’T) to all option commands received.

NAME

ftpd, in.ftpd, setup.anonftp − DARPA Internet File Transfer Protocol server

SYNOPSIS

ftp stream tcp nowait root /usr/sbin/in.ftpd in.ftpd
tcpd ftp /usr/sbin/in.ftpd

DESCRIPTION

Ftpd is the DARPA Internet File Transfer Protototocol server process. The server uses the TCP protocol and listens at the port specified in the ‘‘ftp’’ service specification; see **services**(5).

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 (vacuously)
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 -lA’’)
MKD	make a directory
MODE	specify data transfer *mode*
NLST	give name list of files in directory (‘‘ls’’)
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
STOR	store a file
STOU	store a file with a unique name
STRU	specify data transfer *structure*
TYPE	specify data transfer *type*
USER	specify user name
XCUP	change to parent of current working directory
XCWD	change working directory
XMKD	make a directory
XPWD	print the current working directory
XRMD	remove a directory

The remaining ftp requests specified in Internet RFC 959 are recognized, but not implemented.

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.

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

Ftpd authenticates users according to three rules.

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

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

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

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

~ftp) Make the home directory owned by ''ftp'' and unwritable by anyone.

~ftp/bin)
Make this directory owned by the super-user and unwritable by anyone. The program **ls**(1) must be present to support the list commands. This program should have mode 111.

~ftp/etc) Make this directory owned by the super-user and unwritable by anyone. The files **passwd**(5) and **group**(5) must be present for the **ls** command to work properly. These files should be mode 444.

~ftp/pub)
Make this directory mode 755 and owned by the super-user. Create directories in it owned by users if those users want to manage an anonymous ftp directory.

~ftp/pub/incoming)
Optionally create this directory for anonymous uploads. Make it mode 777. The FTP daemon will create files with mode 266, so remote users can write a file, but only local users can do something with it.

The script **setup.anonftp** can be used to create or check an anonymous FTP tree.

SEE ALSO

ftp(1).

BUGS

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

NAME

getty − system login banner

SYNOPSIS

getty [*banner* ...]

DESCRIPTION

Getty displays a system identification banner, reads a user name from standard input and executes **login** with that name as argument.

Getty uses its arguments separated by spaces as a login banner. The character sequences **\n**, **\s** and **\t** are printed as newline, space and tab. Any other *x* prints that *x*. The character sequences **%s**, **%n**, etc. produce the same output as **uname −s**, **uname −n**, etc. The default banner is

%s\s\s\sRelease\s%r\sVersion\s%v\n\n%n\slogin:\s

The only other useful functionality offered by **getty** is that it can be suspended by signal SIGUSR1 and restarted by SIGUSR2. This allows a program such as **modem**(1) to temporarily claim a dialin line for dialout.

SEE ALSO

modem(1), **ttytab**(5), **init**(8).

BUGS

Getty should be taught about modems and modem lines. The signal trick is an awful hack.

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)

NAME

halt − abruptly stop the system

SYNOPSIS

halt [**−f**]

DESCRIPTION

Halt stops the system almost immediately. The users are not informed about the things to come. **Halt** is logged in **/usr/adm/wtmp** and in **/usr/adm/authlog**, if these files exist. **Halt** should only be run by the super-user, any other caller will be refused.

Halt is a rather rude program. **Shutdown**(8) is preferred for it performs a more gentle halt routine.

Halt −f is even worse, it omits the terminate signals that are normally sent first to all processes to give them a chance to die peacefully.

SEE ALSO

reboot(2), **shutdown**(8), **reboot**(8), **boot**(8).

AUTHOR

Edvard Tuinder (v892231@si.hhs.NL)

NAME

ifconfig − configure a TCP/IP device

SYNOPSIS

ifconfig [**−I** *ip-device*] [**−h** *ipaddr*] [**−n** *netmask*] [**−iv**]

DESCRIPTION

Ifconfig initializes a TCP/IP device setting the IP address and/or netmask. It will report the address and netmask set. This command may be used if the system has not been configured properly yet. It is only used at boot time to set a fixed address for a system without a physical ethernet. Normally the **inet** task will find it out by itself from the RARP server.

OPTIONS

−h The decimal TCP/IP address to set.

−n The netmask to set.

−i Don't set the IP address or netmask if already set. This way ifconfig cannot interfere if the numbers have been found out by RARP.

−v Report IP address and netmask. This is the default action if there are no other options.

SEE ALSO

hostaddr(1), **rarpd**(8), **set_net_default**(8), **boot**(8).

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)

NAME

inet − TCP/IP server

SYNOPSIS

inet

DESCRIPTION

Inet is the TCP/IP server. It is a device driver that interfaces between the file server and the low level ethernet device driver. The interface to this server is described in **ip**(4).

Inet is part of the kernel image under standard Minix. Under Minix-vmd it starts as a normal process, but by using a few low level system calls it quickly turns itself into a server.

SEE ALSO

ip(4).

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)

NAME

init − grandparent of all processes

DESCRIPTION

The first program started by Minix is **init**. The actions performed by **init** can be summarized by this pseudo shell program:

```
# Open 0, 1, 2.
exec </dev/null >/dev/log 2>&1

# Run the system initialization script.
sh /etc/rc $bootopts

>/etc/utmp
echo reboot >>/usr/adm/wtmp

while :; do
      # Wait for a process to exit, but don't always block.
      wait

      # Record logout.  (Not in this dumb way, of course.)
      if "pid is in my tables" $pid
      then
            echo "logout $pid" >/etc/utmp
            echo "logout $pid" >>/usr/adm/wtmp
      fi

      # Start a new session.
      while read line type getty init
      do
            if idle $line
            then
                  $init ... <$tty >$tty
                  $getty <$tty >$tty 2>&1 &
                  pid=$!
                  "add pid to tables" $pid
                  echo "login $line $pid" >/etc/utmp
                  echo "login $line $pid" >>/usr/adm/wtmp
            fi
      done < /dev/ttytab
done
```

The first action of **init** is to run **/etc/rc** to initialize the system as described in **boot**(8). **Init** then enters its main loop where it waits for processes to exit, and starts processes on each enabled terminal line. The file **/etc/ttytab** contains a list of terminal devices, their terminal types, the program to execute on them to allow one to login (usually **getty**(8)), and the program to execute first to initialize the line (usually **stty**(1)). These fields may be left out to indicate that a line is disabled or that initialization is not necessary. The commands are searched using the path **/sbin:/bin:/usr/sbin:/usr/bin**.

Init accepts several signals that must be sent to process id 1. (It is the first process, so natually its process id is 1.) The signals are:

SIGHUP

When receiving a hangup signal, **init** will forget about errors and rescan **ttytab** for processes to execute. **Init** normally rescans **ttytab** each time it feels the need to respawn a process, so the hangup signal is only needed if a line has been shut down, or after a terminate signal. Note that after turning a line off you will have to kill the process running on that line manually, **init** doesn't do that for you.

SIGTERM

Normally sent by programs that halt or reboot Minix. Causes **init** to stop spawning new processes.

SIGABRT

Sent by the keyboard driver when the **CTRL-ALT-DEL** key combination is typed. Causes **init** to run the **shutdown** command. A second abort signal makes **init** halt the system directly with a system call. The keyboard driver halts the system, without a sync, after the third CTRL-ALT-DEL.

Minix vs. Minix-vmd

There are a few differences between standard Minix and Minix-vmd on how **init** is run. The **/etc/rc** file is executed under standard Minix with input connected to **/dev/console**, but under Minix-vmd this is still **/dev/null**. This means that under Minix-vmd processes must be reconnected to **/dev/console** with the **intr** program if they need user interaction. Minix-vmd passes the value of the **bootopts** boot variable to /etc/rc. Standard Minix does not.

FILES

/etc/ttytab	List of terminals devices.
/etc/utmp	List of currently logged in users.
/usr/adm/wtmp	Login/logout history.

SEE ALSO

ttytab(5), **utmp**(5), **getty**(8), **stty**(1), **boot**(8).

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)

NAME

inodes − print i-node information

SYNOPSIS

inodes

EXAMPLES

inodes	# Print information about file names typed in
cd /dev; ls | inodes	# Print information about the special files

DESCRIPTION

Inodes expects a list of file names on *stdin*, one file name per line. For each file named, the file type, mode, uid, gid, checksum, length, and name is printed. The checksum is the same as used by *crc*. This program provides a way to see the sizes of the block special files in */dev*, as shown in the second example above.

NAME

installboot − make a device bootable

SYNOPSIS

installboot −i(mage) *image* [*label*:]*kernel mm fs ... init*
installboot −(e)x(tract) *image*
installboot −d(evice) *device bootblock boot* [[*label*:]*image* ...]
installboot −b(oot) *device bootblock boot* [*label*:]*image ...*
installboot −m(aster) [*fix*] *device masterboot*

DESCRIPTION

Installboot may be used to make a device bootable by constructing a kernel image and installing bootstrap code into the boot block of a Minix file system. To understand how this can be done one first has to know what happens when a PC is booted.

When the power is turned on the typical PC will try to read the first sector from the first floppy disk or from the first hard disk into memory and execute it. The code obtained from the hard disk (from the so-called master boot sector) will immediately replace itself by the code found in the first sector of the active partition. Thus the PC is now executing the bootstrap code found in the first sector of /dev/fd0, /dev/hd1, /dev/hd2, /dev/hd3, or /dev/hd4. The bootstrap will locate the operating system on the device it itself was loaded from, load it, and execute it.

To make a Minix file system **/dev/fd0** mounted on **/mnt** bootable, enter the following:

cp /usr/mdec/boot /mnt/boot

installboot −i /mnt/minix kernel mm fs init

installboot −d /dev/fd0 /usr/mdec/bootblock boot

The "boot" program in the example is named the "boot monitor". It is loaded by the bootblock code placed in the boot sector of /dev/fd0 and it will take care of loading the kernel image "minix" from the root directory of the file system. See **monitor**(8) for a description of the boot monitor. Note that **boot** is a name in the file system on **/dev/fd0** in this example, the same file as **/mnt/boot**. Making **/mnt/minix** is normally not necessary, there is usually a kernel image in the **tools** directory.

OPTIONS

−i(mage) *image* [*label*:]*kernel mm fs ... init*

The **−image** option (or the **−i** shorthand) combines the executable files needed to run Minix in one file. Only the names and a few zero bytes are inserted into the image. The name is for identification and the zeros are used to pad separate pieces to sector boundaries for fast loading.

An executable may be prefixed by a label. The monitor may be instructed to load processes by label. So more than one kernel process may be included in the image, each with a different winchester driver for instance. So if you have compiled two different kernels with an AT or XT driver then

installboot −i *image AT:at_kernel XT:xt_kernel mm fs init*

will make an image with two different labeled kernels and one unlabeled set of the other binaries.

−(e)x(tract) *image*

Extract the binaries from *image* under the names stored in the image. (The name includes the optional label.)

−d(evice) *device bootblock boot* [[*label*:]*image* ...]

Installs *bootblock* in the boot sector of *device* together with the disk addresses to *boot*. These disk addresses are needed to load *boot* from the file system at boot time. The argument *boot* is first searched in the file system on *device*. If it is not found then it is read as a normal file and added at the end of the file system. The file system should be smaller than the device it is on to allow this. Any extra images are also added to the end as described under **−boot**. (Make sure you understand all this.)

The device need not be mounted when **installboot** is run, nor does it matter if it is.

Installboot needs to be run again if *boot* is rewritten, because it will then occupy a new place on the disk.

Old boot parameters are kept if there are no images added.

−b(oot) *device bootblock boot* [*label*:]*image ...*

This option fills a blank floppy in *device* with boot code and kernel images. This "boot disk" does not have a root file system, only the boot monitor and Minix kernels. The boot parameters sector is filled with code that enables menu options for selecting an image. After loading an image, the monitor will ask you to insert a root file system diskette before starting Minix.

The labels used on the images should match those on the executables used inside the image. You can put a comma separated list of labels on an image for each label used within the image. For the image created earlier one would create a boot floppy like this:

installboot −b /dev/fd0 bootblock boot *AT,XT:image*

If a label-list is omitted on an image, then that image will be selected by default. (Like in the normal one image, no labels case.)

Note that **−device** and **−boot** together allow you to make a boot floppy with or without a root file system. With the boot code in the file system, attached to the end of it, or after the boot block. And with one or more kernel images in the file system or at the end of the device. Somewhat confusing.

−m(aster) [*fix*] *device masterboot*

This option installs the *masterboot* program into the boot sector of the given device. If another device is given instead of *masterboot* then its bootstrap code is copied to *device*. The master bootstrap on a hard disk boots the active partition on that disk at boot time. The MS-DOS fdisk command normally puts a master bootstrap on the hard disk. Minix has two bootstraps that can be used as a master bootstrap. A fairly normal one named **masterboot** that works as follows:

If the ALT key is held down while booting then '/dev/hd?' appears and you are expected to type a number key (0 − 9) to select the device to boot.

If *fix* (a small number) is given then the bootstrap is locked into booting the **/dev/hd***fix* disk or primary partition. This is needed if 'boot *hd*N*' is used from the monitor to boot an O.S. that needs the active flag set.

If installed on a Minix floppy then it will try to boot the next floppy or the first hard disk. Ideal for floppies with just data on it, they will no longer obstruct the boot process if left in the drive. Also a very useful trick to boot from floppy drive 1.

If installed on a hard disk then the active partition is selected and booted as usual, unless none of the partitions is marked active, then it will boot the next disk. The latter is useful if you want to boot an operating system from the second disk by default.

The second bootstrap is named **extboot**. It has only one function, to boot the logical partition named by *fix*. *Fix* is not optional for **extboot** and must be a number-letter pair, like **2c** for **/dev/hd2c**.

Extboot or **masterboot** with a fix key need not be installed in the hard disk master bootstrap per se if you don't want to mess with the DOS master bootstrap, or if you want keep the active flag functioning. An extended partition or a non-root Minix partition are better candidates. It seems logical to put **extboot** in the extended partition boot block.

A backup copy of the current master bootstrap (including the partition table) can be made with:

dd if=*device* of=*backup-file* count=1

A simple 'cat *backup-file* > *device*' will put it back. You can also use **fdisk /mbr** under MS-

DOS 5.0 (or newer) to restore the master bootstrap.

FILES

/usr/mdec/bootblock	Minix bootstrap for the Minix root device. To be placed in the boot sector.
/usr/mdec/boot	Minix Boot Monitor. Can usually be found in the root directory of a bootable device.
/usr/mdec/masterboot	Master bootstrap. Can be placed in the first sector of a disk to select the active partition. In a Minix primary partition it selects the active subpartition.
/usr/mdec/extboot	Extended partition bootstrap.

SEE ALSO

part(8), **monitor**(8).

DIAGNOSTICS

Boot doesn't fit on *device*

If there is no space on the device to add the boot code. This usually means that there is no boot code in the file system you use **installboot −device** on.

Image doesn't fit on *device*

If the device is too small for all the images you try to put on it.

BUGS

It has four more options than the SunOS installboot program it is modeled after.

The bootblock code has been crunched to such ugliness that you can use it to scare little kids out of your garden.

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)

NAME

irdpd − internet router discovery protocol daemon

SYNOPSIS

irdpd [**−bsd**] [**−U** *udp-device*] [**−I** *ip-device*] [**−o** *priority-offset*]

DESCRIPTION

Irdpd looks for routers. This should be a simple task, but many routers are hard to find because they do not implement the router discovery protocol. This daemon collects information that routers do send out and makes it available.

At startup **irdpd** sends out several router solicitation broadcasts. A good router should respond to this with a router advertisement.

If a router advertisement arrives then no more solicitations are sent. The TCP/IP server has filled its routing table with the info from the advertisement, so it now has at least one router. If the advertisement is sent by a genuine router (the sender is in the table) then the **irdpd** daemon goes dormant for the time the advert is valid. Routers send new adverts periodically, keeping the daemon silent.

Otherwise **irdpd** will listen for RIP (Router Information Protocol) packets. These packets are sent between routers to exchange routing information. **Irdpd** uses this information to build a routing table.

Every now and then a router advertisement is sent to the local host to give it router information build from the RIP packets.

Lastly, if a router solicitation arrives and there is no router around that sends advertisements, then **irdpd** sends an advertisement to the requestor. Note that this is a direct violation of RFC1256, as no host is supposed to sent those adverts. But alas the world is not always perfect, and those adverts make booting hosts find routers quickly with this help from their brothers. (Of course, they will lose the router soon if they don't have an **irdpd** daemon themselves.)

OPTIONS

−b Broadcast advertisements instead of sending them to the local host only. This may be used to keep (non-Minix) hosts alive on a net without adverts.

−s Be silent, do not send advertisements to hosts that ask for them.

−d Debug mode, tell where info is coming from and where it is sent. Debugging can also be turned on at runtime by sending signal **SIGUSR1** or turned off with **SIGUSR2**.

−o *priority-offset*
Offset used to make the gateway's preferences collected from RIP packets look worse than those found in genuine router adverts. By default **-1024**.

SEE ALSO

set_net_default(8), **boot**(8), **inetd**(8), **nonamed**(8), **rarpd**(8).

BUGS

Under standard Minix this daemon can't listen to two both IRDP and RIP at the same time, so it starts out with IRDP. It switches over to RIP if it can't find a router, or if it threatens to lose its router. It does not switch back.

Irdpd may help a host that should not be helped, i.e. if it doesn't have an **irdpd** daemon with RIP collecting trickery. It will make System Administrators pull out their remaining hair trying to find out why a host can access outside networks for a some time after boot, but goes blind afterwards.

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)

NAME

MAKEDEV, DESCRIBE − make/describe device files

SYNOPSIS

MAKEDEV [**−n**] *key* ...
DESCRIBE [*device*] ...

DESCRIPTION

MAKEDEV may be used to create the device files normally found in the **/dev** directory. The *key* arguments are simply the names of the devices you want. **MAKEDEV** knows about all supported devices and will create them in the current directory with the proper owner and mode. For many devices **MAKEDEV** will not only create the device you want, but also the devices related to it that you will probably want too. Naming one floppy device will create all floppy devices for the same drive for instance.

Call **MAKEDEV** without arguments to see a list of keys that it understands. Then use the **−n** flag to make the script echo the commands it will execute the next time when you call it without that flag.

The special key **std** must be given alone to **MAKEDEV**. This key will create all standard devices.

The command **DESCRIBE** will give you a one-line description of a given device. It will by default list all devices in **/dev**.

SEE ALSO

mknod(8).

BUGS

MAKEDEV's eagerness to create devices may cause many "File exists" errors from **mknod**.

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)

NAME

mkdist − make a Minix distribution

SYNOPSIS

mkdist

DESCRIPTION

Mkdist makes a Minix distribution on floppies. Run the command as **root** and follow the instructions. It will make one or two bootable installation floppies and a compressed tar file of **/usr** on several floppies using **vol**(1).

The result can be installed on another system as described in **usage**(8), except that all of **/usr** is saved on one set of floppies instead of being nicely split in binary and source packages.

SEE ALSO

tar(1), **compress**(1), **vol**(1), **usage**(8).

NOTES

Also very useful for making backups.

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)

NAME

mknod − create a special file

SYNOPSIS

mknod *file* [**b**] [**c**] *major minor*
mknod *file* **p**

EXAMPLES

mknod /dev/plotter c 7 0
Create special file for a plotter

mknod /dev/fd3 b 2 3 # Create a device for diskette drive 3

mknod /tmp/stream p # Create a named pipe

DESCRIPTION

Mknod creates a special file named *file* , with the indicated major and minor device numbers. The second argument specifies a block special, a character special, or a named pipe. Named pipes do not have device numbers so they are omitted.

SEE ALSO

mkfifo(1), **mknod**(2).

NAME

monitor, edparams − load and start Minix, modify boot parameters

SYNOPSIS

edparams *device* [**command** ...]
boot.com *virdisk*

DESCRIPTION

This text describes the Boot Monitor, a boot time interactive program designed not only to load and start Minix, its most important task, but to also provide an easy to use interface to configure Minix and to boot other operating systems.

The monitor is controlled with an environment that is modeled after the Bourne shell. This environment is filled at startup with default values that depend on the machine the monitor is running on and the environment settings saved into the boot parameters sector (the second sector on a device). When the environment is loaded, the monitor executes the function named **main**, which by default starts a simple menu.

The environment can be manipulated at boot time from the monitor prompt, but may also be edited using **edparams** on a given device. **Edparams** simulates the monitor as much as it can, echoing commands it can't execute between brackets. It can also be used in Makefiles and scripts by giving it commands as arguments.

The MS-DOS version of the monitor, usually named **boot.com** under DOS, boots Minix from a "DOS virtual disk". (See below.)

COMMANDS

The monitor is best described by the commands you can type to the '>' prompt. This is known as the "monitor mode". You can enter this mode by hitting the Escape key. These are the monitor commands:

name = [**device**] *value*

Set environment variable.
Changes the value of *name* to *value*. The optional word **device** marks *name* as being subject to device translation. (See the section on devices.) These (name, value) pairs are passed to the kernel who uses them to configure itself. These variables are passed by default:

rootdev
This is the device used as your root device. It is by default set to **ram,** which means that the device specified by **ramimagedev** will be loaded into the RAM disk and used as root. If you change this variable then a physical device will be used as root, and the RAM disk will be uninitialized and have the size specified by **ramsize**.

ramimagedev
Describes the device to use to initialize the RAM disk if **rootdev** is set to **ram**. It's by default set to **bootdev**, a special name for the device the monitor booted from.

ramsize
The size of the RAM disk. If the RAM disk is used for the root file system then the root file system is stretched out to **ramsize** if possible.

processor
Set by default to **86**, **186**, **286**, **386**, **486**, ... depending on the hardware you have. You can set it to a smaller value to test your kernel in a more limited environment.

bus
The type of system bus, either **xt**, **at** or **mca**. This answers basic questions like: "How many interrupt controllers and how to initialize?" Or: "Does the keyboard have LEDs?"

memsize
Kilobytes of conventional memory. This is the amount of RAM within the first megabyte.

emssize
Kilobytes of extended memory.

video
Describes capabilities of the VDU: **mda**, **cga**, **ega** or **vga**.

chrome
Either **color** or **mono**.

console
If set to a hexadecimal value makes the monitor set the BIOS video mode to this value. This allows the use of video modes with more rows or colums than the standard 80x25 mode. The kernel must of course be able to handle a nonstandard mode. More parameters may follow the mode number. **Warning:** Not all monitors can handle all of the modes, some may generate frequencies that can damage your monitor. Read the manual of card and monitor for details.

Two variables are only used by the monitor, even though they are passed to the kernel too:

image
The name of the file containing the kernel image, by default **minix**. If it refers to a directory however then the newest file inside the directory is chosen to be the kernel image. The names inside **/minix/** are best set to the Minix version you are using, which looks good when the monitor prints its name. Rules for pretty printing image names:

A '/' or '_' is changed to a space.

The first letter is changed from lowercase to uppercase.

An 'r' if followed by a digit changes to " revision ".

label
If set then only processes marked with this label or without a label are loaded from the image.

Installboot −boot will create functions to select images and labels. These functions will set label and image and echo what you selected. The two numbers separated by a colon used as an image name tell the starting sector and sector count of the image on disk.

name() { ... }
Define function.
Functions may be used to bundle a set of commands, so that you can easily boot Minix with a different set of parameters then normal. E.g.

ram() { rootdev=ram; boot }

will allow you to run Minix with the root device on RAM for a change, if you normally use a real device as root. The only pre-set function is **main** with default value **menu**, which is the default command executed by the monitor. You can use newlines after the ')' token, the monitor will then use a '+' prompt and ask for the rest.

name(*key*) { ... }
Define kernel selecting function.
The menu command uses functions like these to add menu entries to select a different kernel from a boot disk. **Installboot −boot** produces these functions when the images are labeled. The label **AT** would give:

AT(a) {label=AT;image=42:626;echo AT kernel selected;menu}

With the menu option:

a Select AT kernel

Typing **a** will then execute the **AT** function above.

name(*key*,*text*) { ... }

User defined menu option.
This variant may be used to make any menu entry you like:

dos(d,Boot MS-DOS) { boot hd1 }

Text may be anything, even parentheses if they match.

name

Call function.
If *name* is a user defined function then its value is expanded and executed in place of *name*. Try a recursive one like 'rec() {rec;xx}' one day. You can see the monitor run out of space with nice messages about using **chmem**(1) to increase it's heap.

boot [−*opts*]
boot *device*

Boot Minix or another O.S.
Without an argument, **boot** will load and execute the Minix image named by the **image** variable. With options the variable **bootopts** is first set to −*opts* before Minix is started, and unset when Minix returns. With a *device* argument, **boot** loads the boot sector of *device* into memory and jumps to it, starting another operating system. You would normally use partitions on the first hard disk for this command (hd[1−4]), using hd0 will also work (choosing the active partition). One can also boot devices on the second hard disk (hd[5−9]) if the bootstrap writer did not hardwire the disk number to disk 0.
Some Operating Systems can only be booted from the active partition, if you use a '*', e.g. **boot *hd3**, then partition 3 is first made active. You'll then need to use **installboot −master** with a fix key to forcefully boot the Minix partition at startup.

delay [*msec*]

Delay (500 msec default).
Fast booting speed was one of the objectives when this program was created, so a hard disk boot usually takes only a fraction of a second. If you need some time (to hit Escape, or stare at the numbers) you can use **delay** to make the monitor pause for a specified amount of time. To specify a delay just before Minix is started, you can set the variable **delay** to a number of milliseconds. Example:

main() {delay 250; delay=500; boot}

Look at this carefully, 'delay 250' means: "wait 1/4 sec now!", while 'delay=500' means: "wait 1/2 sec after loading Minix".

If you use **delay=swap** then the monitor will wait until you have inserted a root diskette and typed RETURN.

echo *word* ...

Print these words.
Used to tell you that you just selected image X.

ls [*directory*]

List contents of a directory.
Useful when looking for kernel images.

menu

Menu driven startup.
This command allows you to execute functions defined with a *key*. If no menu functions have been defined then **menu** will use this one hidden built-in function:

*(=,Start Minix) { boot }

Kernel selecting functions only add new options to this set, but if you define a two argument function yourself then the above one is no longer shown, allowing you to customize the menu completely. Your first function definition should therefore be one that starts Minix.

Menu entries are shown in the same order as **set** shows them. If you don't like the order then you have to unset the functions and retype them in the proper order.

If you type a key then a scheduled trap is killed and the appropriate menu function is executed. If you need more time to choose then hit the spacebar. A key not on the menu also kills a trap, but does nothing more.

save

Save environment.
This will save all the environment variables and functions with nondefault values to the parameter sector (the second sector on the boot device), so they are automatically set the next time you boot the monitor.

set

Show environment.
Show the current values of the environment variables and functions. Default values are shown between parentheses to distinguish them from values that were explicitly set.

trap *msec command*

Schedule command.
Schedules a command to be executed after *msec* milliseconds. Only the monitor mode cannot be interrupted, a scheduled trap is killed when the prompt is printed. Example:

```
main() {trap 10000 boot; menu}
```

This gives you 10 seconds to choose a menu option before Minix is booted.

unset *name ...*

Unset environment variables.
Removes the named variables and functions from the environment, and sets special variables back to their default values. This is also the only way to remove the "device name translation" property from a variable.

exit

Exit the monitor
Reboot the machine, exit to Minix or exit to DOS as appropriate.

DEVICES

The Minix kernel can't do anything with device names, so they have to be translated to device numbers before they are passed to the kernel. This number is found under the st_rdev field (see **stat**(2)) of the file on the boot file system. The monitor will look for the device file with the working directory set to '/dev'. If it can't find the device name then it will translate names like 'ram', 'fd1', 'hd6', 'hd3a', and 'sd2' to what it itself thinks the numbers should be.

The special name **bootdev** is translated to the name of the device booted from, like 'fd0', or 'hd3', and then searched for in /dev. **Bootdev** can't be translated to a device other then the fd or hd devices, so SCSI devices for instance must be named explicitly.

EXTENSIONS

A few extensions have been made to this program for kernel hackers. They may be triggered by setting bits in the flags word in the kernel startup code (the mpx file.) The flag bits are:

0x0001 Call kernel in 386 mode.

0x0002 Do not make space for the bss areas of processes other then the kernel.

0x0004 Use the stack size set by **chmem**(1).

0x0008 Load MM, FS, etc. into extended memory.

0x0010 No need to patch process sizes into the kernel.

0x0020 The kernel can return to the monitor on halt or reboot.

MS-DOS MONITOR

Minix-vmd has a version of the monitor that runs under MS-DOS to boot a "DOS virtual disk". It is a simple COM program that interprets an MS-DOS file as a disk, loads a Minix kernel from the active partition in the same way as the BIOS based monitor, and executes it to start Minix. All the monitor commands function in the same way, except for the **boot** command, it can only load Minix. The memory that MS-DOS has in use is copied out of the way when Minix takes control, and is put back in place when Minix exits. This memory shuffling also happens when the BIOS disk driver makes BIOS calls, slowing things to a crawl. It is better to use a Minix driver. The MS-DOS monitor does not work if there is a memory manager active that runs in 386 protected mode, like EMM386.

SEE ALSO

chmem(1), **stat**(2), **installboot**(8), **usage**(8), **boot**(8).

BUGS

The **delay** command will hang forever on the original IBM PC (not the XT!). Not that it matters, as everything takes forever on that box.

Reading the first sector to boot a floppy (e.g. **boot fd1**), is done using whatever floppy parameters boot currently has available. This will probably always work.

The two forms of **delay** are a crock.

The word **emssize** comes from EMS, that has to do with expanded memory, not extended memory.

ACKNOWLEDGMENTS

Guy Helmer, for the floppy sensing code that somehow disappeared into the boot block.

Earl Chew, for the inspiration his ShoeLace package provided, unless he wants to file a "look and feel" suit against me, then I will say I modeled it after the SunOS ROM boot monitor, which is also true.

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)

NAME

ncheck − i-node to name converter

SYNOPSIS

ncheck [**−i** *numbers*] [**−a**] [**−s**] *file_system*

OPTIONS

−a List all files, even . and ..

−i Followed by a list of i-nodes

−s List only special and setuid files

EXAMPLES

ncheck − /dev/fd0 # List everything on */dev/fd0*

ncheck −i 10,15 /dev/fd0
List i-nodes 10 and 15 on */dev/fd0*

DESCRIPTION

Sometimes one knows about an i-node number and wants to find the file name that goes with it. This program makes that mapping. The default is to list everything on the device, but the **−i** flag restricts the list to specified i-nodes and the **−s** flag restricts it to special files and setuid files (to look for possible security violations).

SEE ALSO

fsck(1).

NAME

nonamed − not a name daemon, but acts like one

SYNOPSIS

nonamed [**−d**[*level*]] [**−p** *port*] [**−n** *address*[/*port*]]

DESCRIPTION

Nonamed is not an Internet name daemon. When started it immediately tries to find a real name daemon on the local network. If it finds one then it will relay any DNS queries to that name daemon. If it can't find one then it will do its best to answer simple queries using the **/etc/hosts** table for name to address translation.

On startup **nonamed** broadcasts up to five" simple UDP queries on the local network. This is a somewhat nasty thing to do, because it makes all name servers on the net spring into action to answer the request. The first name server to answer is used by **nonamed** to answer the queries of its clients. The broadcast search is repeated after an hour to give another name server a chance to be abused. (It is quite normal for a host to use a remote name server. The broadcasts are a bit unfriendly, but they only happen once an hour.)

If a real name server is found then **nonamed** enters "relay mode", passing all queries on. In relay mode the daemon cannot be distinguished from a real DNS name daemon.

Nonamed accepts both UDP and TCP queries under Minix-vmd. Under standard Minix only UDP queries are accepted. 256" relayed UDP queries can be outstanding before it forgets where the first one came from.

In answer mode **nonamed** can answer simple DNS queries to translate a host name to an IP address, or an IP address to a host name. Suppose **/etc/hosts** looks like this:

```
127.0.0.1      localhost
192.9.200.1    darask.home.cs.vu.nl
192.9.200.2    burask.home.cs.vu.nl
```

Then queries for the host names listed can be answered with the IP addresses to the left of them. If the name can't be found then it is retried with the domain name stripped of, this is because "localhost" is looked up as "localhost.home.cs.vu.nl" in the above example. Reverse lookups are answered with the first hostname on the line with the IP address.

UDP queries are immediately answered on startup if the answer can be found in the hosts file. Failure answers, and answers to TCP queries are delayed until it decides that there are no real name servers out there.

Nonamed employs several timeouts for efficient operation:

If five" broadcasts, two seconds" apart fail to produce a name server on startup then it decides that there are no real name servers around.

After 1 hour" it will again look for a name server. (We don't want to hog the same one indefinitely.)

If no UDP reply is seen in four seconds" than a simple probe is sent the name server. If that doesn't provoke an answer in two seconds" then it will look for a new name server. A failing TCP connection will also invoke a search, the TCP connection is then made to the new name server. A client using UDP will retry eventually, a client using TCP will notice nothing but a short delay. The daemon drops back into answer mode if the search fails and it has a hosts file with a "localhost" entry.

It waits five minutes" for any action on a TCP stream before the connection is aborted.

In answer mode it will generate data with a TTL (time to live) of 1 hour".

OPTIONS

The options are only useful when debugging **nonamed**, although it can be very instructive to watch DNS queries being done.

−d[*level*]

Set debugging level to *level* (a single digit, by default **1**.) Debug mode 1 makes **nonamed** decode and display the DNS queries and replies that it receives, sends and relays. The decoding only handles the common cases. In debug mode 2 it prints tracing information about the internal jobs it executes. In debug mode 3 it core dumps when an error causes it to exit. The debugging level may also be increased by 1 at runtime by sending signal **SIGUSR1** or turned off (set to 0) with **SIGUSR2**.

[**−p** *port*]
Port to use instead of the normal **domain** port.

[**−n** *address*[/*port*]]
Use this IP address and port to find a real name server. It can be used to make **nonamed** relay to another **nonamed**. (They do not listen to broadcasts, of course.) You can run two daemons on one machine if you choose a new port number for one of them. You can force **nonamed** out of relay mode by sending it a hangup signal.

FILES

/etc/hosts Hosts to address translation table when in answer mode.

SEE ALSO

gethostbyname(3), **resolver**(3), **hosts**(5), **set_net_default**(8), **boot**(8), **inetd**(8), **irdpd**(8), **rarpd**(8).

NOTES

You can specify a remote name server in **/etc/resolv.conf** to circumvent **nonamed**. But then you lose its talent for automatically finding new name servers when the remote name server becomes unreachable.

Don’t add a "localhost" entry to the hosts file if there are remote name servers. It makes **nonamed** drop back in answer mode on a flaky network.

BUGS

If you can get a remote **nonamed** to listen to your name server then you can make it believe anything you want to. You need access to a machine on the same subnet of course.

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)

NAME

part − partition table editor

SYNOPSIS

part [*device*] ...

DESCRIPTION

Part is a screen oriented partition table editor.

While editing you will see six lines of numbers, the first line shows the device name and its geometry (number of cylinders, heads and sectors), the second shows the start and end of the drive or partition you are working on, the last four lines show the different partitions or subpartitions. All numbers except those on the second line can be edited. Question marks are showed instead of numbers if the partition table is not loaded yet. You have to select a device and type ’r’.

Editing is a simple matter of moving around with the arrow keys and changing the values with + and − (or PgUp and PgDn), or by typing the desired value. The ’?’ key will give a small list of commands, the ’!’ key gives advice on how to make a new entry.

The spacebar toggles between showing the size of the partition and the last sector on the partition. Useful to check if a partition is adjacent to the next.

The ’m’ key is "magical", it lets you cycle through a set of interesting values for the base or size of a partition. These values are: Aligned to a cylinder, taped to other partitions (inside or outside), or filling out holes. **Use this key**!

Minix subpartition tables or extended partitions may be edited after hitting the ’>’ key. The number of this partition will be shown after the device name on the second row, e.g. **/dev/hd0:2**. Minix subpartition tables are shown as is, but extended partition bases are translated to absolute offsets on the screen to hide the gory details of their implementation from the innocent user. (Hit ’p’ if you dare.) The ’<’ key will bring you back to the enclosing partition table.

With arguments, **part** will use the given devices or files. Without arguments, **part** will use all interesting block devices in **/dev** sorted by device number and starting with **/dev/hd0**.

Values that are out of range, overlapping, or otherwise strange are shown in reverse video. Values that may possibly be a problem for operating systems other then Minix are shown in bold characters.

The name of the device is highlighted when it has not been read yet.

Head or sector numbers are highlighted if the partition does not start or end at a cylinder boundary.

The base and/or size field is highlighted if they fall outside the device, if they are inside some other partition, if the base equals the device’s base (no room for the boot sector), or if the size is zero.

Part complies with the good old UNIX tradition of trusting the user. It will write any table, no matter how bad. You have been warned.

By the way, as far as Minix is concerned there is absolutely no reason to make partitions start precisely on a cylinder or track nor does it have to be an exact number of cylinders long. Minix only looks at the base and size of a partition, the geometry of the drive doesn’t have to be correct. Other Operating systems can be very picky about partitions that are not aligned. Some partition editors may refuse to edit a table, others may even make a mess of the table. The only exception is the first partition, it traditionally starts on the first track, not the first cylinder. All editors must understand this. (Subpartition tables are Minix specific, so there is no reason at all for any alignment.)

Extended Partitions

Extended partitions are a mess that is only made slightly better by **part** by translating the base offsets to absolute numbers. It is better to use DOS **fdisk** to create them, but if you insist on using **part** then this is what they should look like:

> The extended partition entry in the primary partition table must cover the whole logical partition space within it.

The area thus created is split in segments, each segment contains a partition table in sector 0 and one (just one) logical partition.

The first entry of a segment's partition table describes this logical partition: it's partition ID, base and size.

The second entry is an extended partition that describes base and size of the next segment (partition table and logical partition). The last segment's partition table is empty, or contains one logical partition.

SEE ALSO

fd(4), **hd**(4).

BUGS

You can have a table read, messed up, and written in no time, be careful.

You can't type head or sector numbers directly.

Sectors are counted from 0 for consistency, but the partition table counts from 1 like DOS addresses them. Most confusing.

You can't write a backup copy to a file, that's what **dd**(1) with count=1 is for.

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)

NAME

partition − make a partition table

SYNOPSIS

partition [**−mf**] *device* [*type*:]*size*[**+***] ...

DESCRIPTION

Partition makes a partition table on *device* using the types and sizes given. It may be used in combination with **repartition**(8) for automatic installation of Minix.

You may give up to four *type*:*size*[**+***] specifications for the partitions. You may also specify holes before, between, and after the partitions. A hole differs from a partition specification by not having a type.

The first hole is by default 1 sector to make space for the primary bootstrap and the partition table. The other holes are 0.

The *type* field is the type of the partitition in hexadecimal. The *size* field is the partition's size in sectors. The **+** or ***** may optionally be added to indicate that the partition must be expanded to contain any leftover space on the device or to mark the partition active.

Partitions are padded out to cylinder boundaries, except for the first one, it starts on track 1. Some operating systems care about this. Minix and MS-DOS do not.

OPTIONS

−m Minix only, no need to pad partitions. This is the default for subpartition tables.

−f Force making a partition table even if the device is too small.

EXAMPLE

partition /dev/hd0 01:16384 81:40000 81:2880* 06:20000+

Partitions disk 0 into an 8 Mb DOS partition, 20 Mb Minix /usr, 1.44 Mb Minix / (active), and a DOS partition of at least 10 Mb at the end of the disk. (06:0+ would have been ok too, it's just a sanity check.)

SEE ALSO

hd(4), **part**(8), **repartition**(8).

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)

NAME

printroot − print the name of the root device on standard output

SYNOPSIS

printroot

OPTIONS

−r Print only the root device, not a full mtab line

EXAMPLES

printroot # Print the name of the root device

DESCRIPTION

Printroot is useful for initializing the */etc/mtab* when the system is booted. It figures out what the root device is by searching */dev* until it finds a block special file with the right major/minor device numbers.

SEE ALSO

fstab(5), **boot**(8).

NAME
pr_routes − show IP routing.

SYNOPSIS
pr_routes [**−i** ip device]

DESCRIPTION
Pr_routes displays the IP routing table.

OPTIONS
−i *ip device* specifies the ip device.

SEE ALSO
add_routes(8), **irdp**(8).

AUTHOR
Pr_routes.c was written by Philip Homburg. This manual page by A. S. Woodhull, last revised 13.02.96.

NAME

pwdauth − password authentication program

SYNOPSIS

/usr/lib/pwdauth

DESCRIPTION

Pwdauth is a program that is used by the **crypt**(3) function to do the hard work. It is a setuid root utility so that it is able to read the shadow password file.

Pwdauth expects on standard input two null terminated strings, the password typed by the user, and the salt. That is, the two arguments of the **crypt** function. The input read in a single read call must be 1024 characters or less including the nulls. **Pwdauth** takes one of two actions depending on the salt.

If the salt has the form "##*user*" then the *user* is used to index the shadow password file to obtain the encrypted password. The input password is encrypted with the one-way encryption function contained within **pwdauth** and compared to the encrypted password from the shadow password file. If equal then **pwdauth** returns the string "##*user*" with exit code 0, otherwise exit code 2 to signal failure. The string "##*user*" is also returned if both the shadow password and the input password are null strings to allow a password-less login.

If the salt is not of the form "##*user*" then the password is encrypted and the result of the encryption is returned. If salt and password are null strings then a null string is returned.

The return value is written to standard output as a null terminated string of 1024 characters or less including the null.

The exit code is 1 on any error.

SEE ALSO

crypt(3), **passwd**(5).

NOTES

A password must be checked like in this example:

```
pw_ok = (strcmp(crypt(key, pw->pw_passwd), pw->pw_passwd) == 0);
```

The second argument of crypt must be the entire encrypted password and not just the two character salt.

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)

NAME

rarpd − reverse address resolution protocol daemon

SYNOPSIS

rarpd [**−d**]

DESCRIPTION

Rarpd listens on the ethernet for broadcast packets asking for reverse address resolution. These packets are sent by hosts at boot time to find out their IP address. **Rarpd** looks up the six octet ethernet number in the **/etc/ethers** file finding a host name. This name is translated to the IP address of the host by a DNS lookup. The IP address is then sent to the host.

Before **rarpd** can start its service it first finds out what the IP addresses of the ethernets are. It will look through **/etc/ethers** to map the ethernet addresses to host names. It then uses **/etc/hosts** to map the host names to IP addresses. If this lookup fails then several RARP requests are broadcasted in the hope that some RARP server knows the addresses. The IP addresses are eventually set in the same way as **ifconfig**(8). (The address is not changed if already set with **ifconfig**.) Note that the host names in the ethers and hosts files must match exactly. The DNS can not be used yet, so a simple name can't be translated to a fully qualified name.

Rarpd exits after startup if there are no active ethernets, or if there is no ethers file.

Warning! Sun diskless workstations assume that the first RARP server that answers is the host they are to boot from. For this to work all other Sun RARP servers delay their answer if they are not also the requestors boot server. The Minix **rarpd** does not have this kludge so it will happily engage the Sun boot server to see who can answer the client first. Unless your Minix host can actually serve a Sun diskless client, it is better not to list any more hosts in the ethers file than necessary.

OPTIONS

−d Turns on debugging messages. Debugging can also be turned on at runtime by sending signal **SIGUSR1** or turned off with **SIGUSR2**.

SEE ALSO

ifconfig(8), **ethers**(5), **hosts**(5), **set_net_default**(8), **boot**(8), **inetd**(8), **irdpd**(8), **nonamed**(8).

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)

NAME

readclock − read the real time clock [IBM PC/AT and AMIGA]

SYNOPSIS

readclock

EXAMPLES

date `/usr/bin/readclock` </dev/tty

Useful in */etc/rc*

DESCRIPTION

Readclock reads the AT’s real time clock and prints the result in a form useful to date, namely, MMDDYYhhmmss. If the clock does not exist (e.g., on a PC), it outputs ’−q’ to query the user for the time. The example given above can be put in */etc/rc* to load the real time when the system is booted.

SEE ALSO

date(1).

NAME

reboot − reboot the system immediately

SYNOPSIS

reboot [**−f**]

DESCRIPTION

Reboot can be used to reboot the system after installing a new kernel. It does not inform the users, but does log it's actions in **/usr/adm/wtmp** and **/usr/adm/authlog**. The system is then rebooted with the **reboot**(2) systemcall.

If the **−f** flag is not given then all processes are sent terminate signals to give them a chance to die peacefully before the **reboot()** call.

If the wtmp file exists, **reboot** logs itself as if it were a shutdown. This is done to prevent **last**(1) from talking about system-crashes. **Reboot** is registered as is in the authlog file.

Reboot can only be executed by the super-user. Any other caller will be refused, either by **reboot**(8) or by **reboot**(2).

SEE ALSO

reboot(2), **shutdown**(8), **halt(8), boot**(8).

BUGS

The error message's given by **reboot** are not always useful. There are several routines that can fail, but which are not fatal for the program.

AUTHOR

Edvard Tuinder (v892231@si.hhs.NL)

NAME

repartition − load a partition table

SYNOPSIS

repartition *device* [*partition-file*]

DESCRIPTION

Repartition uploads a new partition table for the partitions of *device*. The table is obtained from the first sector of *partition-file* if given, *device* otherwise. *Device* may refer to the whole drive or a primary partition, depending on whether you want to upload a partition or a subpartition table. The partitions will be truncated to fit within the enclosing device like the disk driver does, unless the numbers are coming from *partition-file*.

EXAMPLES

repartition /dev/hd0
repartition /dev/hd4 /etc/hd4.table

Reload the partition table of drive 0 setting /dev/hd[1−4], and the subpartition table of /dev/hd4 setting /dev/hd4[a−d] using a file. The latter may be useful if you need more than the 4 subpartitions a single Minix partition gives you.

DIAGNOSTICS

The new table is printed on standard output.

FILES

/dev/hd[0-9]

SEE ALSO

hd(4), **part**(8).

BUGS

The disk must be in use for the changes to stick. The partition table of an idle disk will be reloaded on the first open.

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)

NAME

rlogind, in.rld − remote login server

SYNOPSIS

login stream tcp nowait root /usr/sbin/in.rld in.rld
tcpd login /usr/sbin/in.rld

DESCRIPTION

Rlogind is the server for the **rlogin**(1) program. The server provides a remote login facility with authentication based on privileged port numbers from trusted hosts.

Rlogind listens for service requests at the port indicated in the ‘‘login’’ service specification; see **services**(5). When a service request is received the following protocol is initiated:

1) The server checks the client’s source port. If the port is not in the range 0-1023, the server aborts the connection.

2) The server checks the client’s source address and requests the corresponding host name (see **gethostbyaddr**(3), **hosts**(5) and **named**(8)). If the hostname 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 **tty**(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**(8), but if automatic authentication fails, it reprompts the user to login as one finds on a standard terminal line.

The parent of the login process manipulates the master side of the pseduo terminal, operating as an intermediary between the login process and the client instance of the **rlogin** program. In normal operation, the packet protocol described in **tty**(4) is invoked to provide ˆS/ˆQ 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.

SEE ALSO

rlogin(1).

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.

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.

NAME

rshd − remote shell server

SYNOPSIS

shell stream tcp nowait root /usr/sbin/in.rshd in.rshd
tcpd shell /usr/sbin/in.rshd

DESCRIPTION

Rshd is the server for the **rcmd**(3) routine and, consequently, for the **rsh**(1) program. The server provides remote execution facilities with authentication based on privileged port numbers from trusted hosts.

Rshd listens for service requests at the port indicated in the ‘‘cmd’’ service specification; see **services**(5). When a service request is received the following protocol is initiated:

1) The server checks the client’s source port. If the port is not in the range 0-1023, the server aborts the connection.

2) The server reads characters from the socket up to a null (‘\0’) byte. The resultant string is interpreted as an ASCII number, base 10.

3) If the number received in step 1 is non-zero, it is interpreted as the port number of a secondary stream to be used for the **stderr**. A second connection is then created to the specified port on the client’s machine. The source port of this second connection is also in the range 0-1023.

4) The server checks the client’s source address 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 the user identity 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 a user identity to use 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 local (server-end) user name is looked up in the password file and a **chdir** is performed to the user’s home directory. If either the lookup or **chdir** fail, the connection is terminated. If the user is not the super-user, (user id 0), the file **/etc/hosts.equiv** is consulted for a list of hosts considered ‘‘equivalent’’. If the client’s host name is present in this file, the authentication is considered successful. If the lookup fails, or the user is the super-user, then the file **.rhosts** in the home directory of the remote user is checked for the machine name and identity of the user on the client’s machine. If this lookup fails, the connection is terminated.

9) A null byte is returned on the initial socket 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 initial socket, 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 execution of the login shell).

‘‘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.

SEE ALSO

rsh(1), **rcmd**(3).

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.

NAME

screendump − write current console screen to standard output

SYNOPSIS

screendump

DESCRIPTION

Screendump prints the contents of the console screen to standard output. It does this by reading the screen memory, skipping over attribute bytes, omitting trailing blanks and inserting a newline character at the end of each line.

NOTES

The most common use of **screendump** is with output redirected to a file. This allows screen displays (including output of F-keys) to be captured for inclusion in other documents.

This version is for IBM-PC architecture only.

BUGS

A network user captures an image of the main console, not his or her own screen. The output will usually not be what you expect if the display is in hardware scrolling mode, since in that mode the order of the lines in screen memory may not be the same as what appears on the screen. If you expect to use **screendump** to send notes to someone about problems that occur while using **MINIX** you must remember to toggle to software scrolling before you make the dump. It will also be necessary to use **su**, because **/dev/mem** is normally not world readable.

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)

NAME

shutdown − graciously close the system down

SYNOPSIS

shutdown [**−hrRmk**] [**−x** *code*] [*time-specification* [*message*]]

DESCRIPTION

Shutdown is a program which allows a system operator to close down the system in an nice way. **Shutdown** informs the users why and when the system is going down. This warning is issued 10 minutes before shutdown time and every minute in the last 5 minutes. At this time (5 minutes), **shutdown** creates a file **/etc/nologin** to prevent new users from logging in.

Shutdown keeps a logfile of shutdowns. Every shutdown is registered in **/usr/adm/wtmp**, if this file exists. After these actions, a call is done to **reboot**(2) which actually brings the system down.

Time-specification may be something like **15:00**, **15.00**, **+15**, or **now** for a shutdown at 3pm (twice), 15 minutes from now, or immediately.

The message may be used to describe why the system is going down, it may also be typed on standard input with the **−m** option.

OPTIONS

−h This flag prevents the system from rebooting after the shutdown. The system can now be powered off. This is the default.

−r This flag indicates that the system should reboot after shutting down.

−R Reboot the system by resetting it. Normally the kernel will try to return to the Boot Monitor. With **−R** the system will receive a hardware reset.

−x *code* Halt the system and let the Monitor execute the given code as if typed at the monitor prompt. You can for instance use **−x ’boot hd0’** as a very fast way to reboot "from the top."

−m Allows the operator to type a shutdown message on standard input, that will be added to the messages displayed on all terminals.

−k This option gives the possibility of terminating an already started shutdown. This is only possible if shutdown time has not yet arrived.

−C Check if the system crashed. This option is not used at shutdown time, but at reboot time. It tells if the file systems should be checked by testing if the last entry in the wtmp file is a shutdown entry. (A crude replacement for a file system clean flag.)

FILES

/usr/adm/wtmp, /etc/nologin, /usr/adm/authlog

SEE ALSO

reboot(2), **wall**(1), **halt**(8), **boot**(8).

AUTHOR

Edvard Tuinder (v892231@si.hhs.NL)

NAME

srccrc − compute CRC checksums of the entire source tree

SYNOPSIS

srccrc

OPTIONS

EXAMPLES

DESCRIPTION

Srccrc traverses the entire /usr/include and /usr/src tree to run the crc command on all files found. The result is a long list of filenames with their checksums. The filenames are relative to /usr.

The command makes an effort to remove most junk files such as .o, .bak and files in bin directories. It cannot find single binaries however, so you will have to run **make clean** in /usr/src to be able to make a crc list that contains only source files.

Two crc files can be compared easily with the *diff* command. A crc list of the original source tree can be found in /usr/src/crclist.

SEE ALSO

crc(1).

NAME
sync − flush the cache to disk

SYNOPSIS
sync

OPTIONS
(none)

EXAMPLES
sync # Write out all modified cache blocks

DESCRIPTION
MINIX maintains a cache of recently used disk blocks. The *sync* command writes any modified cache blocks back to the disk. This is essential before stopping the system, and should be done before running any *a.out* program that might crash.

SEE ALSO
sync(2), **shutdown**(8).

NAME

s_n_d, set_net_default − select the default TCP/IP network

SYNOPSIS

set_net_default: not found

DESCRIPTION

This manual page is here because so many others refer to it. The **set_net_default** command does not exist under standard Minix, because the TCP/IP server only supports one network. The server under Minix-vmd supports four different networks, that can each be chosen as the default network.

The TCP/IP library and commands do have a notion about a "default network" however, because the code is shared between the Minix versions. So if you want to venture into network programming then you should know about the devices and environment variables mentioned below to make your program compatible. (Especially since there is a plan to upgrade the TCP/IP server for standard Minix.)

Options and environment to change the default

Many TCP/IP programs implement the following options and environment variables to change the default devices set by **set_net_default**. The options are only implemented for low level programs where it makes sense to name a network device. The environment variables are used in all code. You can run a process and all its children connected to a different network by setting four environment variables.

ETH_DEVICE=*device*
-E *device*
Device to use as raw ethernet device instead of the default /dev/eth.

PSIP_DEVICE=*device*
-P *device*
Pseudo IP device to use instead of /dev/psip.

IP_DEVICE=*device*
-I *device*
IP device to use instead of /dev/ip.

TCP_DEVICE=*device*
-T *device*
TCP device to use.

UDP_DEVICE=*device*
-U *device*
UDP device to use.

FILES

/dev/eth[01]	First and second raw ethernet.
/dev/psip[01]	First and second Pseudo IP network.
/dev/ip[0123]	IP devices for two ethernets and two Pseudo IP networks.
/dev/tcp[0123]	TCP devices for same four networks.
/dev/udp[0123]	UDP devices.

/dev/eth, /dev/psip, /dev/ip, /dev/tcp, /dev/udp
Devices for the default network, links to the devices above. **Eth** is only present if ethernet is the default, **psip** only for pseudo IP.

SEE ALSO

ip(4), **boot**(8).

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)

NAME

update − periodically write the buffer cache to disk

SYNOPSIS

update

EXAMPLES

update & # Start a process that flushes the cache

DESCRIPTION

When the system is booted, *update* is started up in the background from */etc/rc* to issue a SYNC system call every 30 sec.

SEE ALSO

boot(8).

NAME

usage − installing and using Minix

DESCRIPTION

This manual page describes the installation and use of Minix from a System Administrators point of view. It contains an installation guide, instructions on how to do the initial configuration and some other info. Please read this document entirely before attempting to install Minix. The installation steps are in the proper order, but not all the information you may need is presented at the right moment. Other detailed information that may be useful can be found in **boot**(8) and **hier**(7).

1. REQUIREMENTS

The minimum system Minix can be installed on comfortably is an IBM PC/AT or PS/2 with a 286 processor, 2 Mb memory, a 720 kb diskette drive, and 30 Mb free space on an AT, ESDI, or SCSI hard disk (the latter controlled by an Adaptec 1540.) Minix for the 386 (Minix-386 for short) can be installed on a machine with at least a 386sx processor and 3 Mb memory.

2. MINIX INSTALLATION BACKGROUND

The objective of the installation is to create a partition on your disk and to put Minix into it. Minix really requires two partitions however, so the single "primary" partition is split into two subpartitions. The **a** subpartition will contain the root file system, and the **c** subpartition will contain the **/usr** file system. What MS-DOS calls "drives", i.e C:, D:, E:, Minix calls "file systems". Minix does not use drive letters, but requires that one file system is made a part of another file system by "mounting" one on the other. The "root" file system is always present and starts with the directory "/", the root of the directory tree. The root file system contains a few programs in **/bin**, device files in **/dev**, and configuration files in **/etc**. This is just enough to get the system started. Minix will soon extend its directory tree by mounting a file system on the **/usr** directory. What is henceforth known as the /usr file system contains all Minix programs in **/usr/bin**, file system sources in **/usr/src**, etc, etc. The ROOT image contains the complete Minix root file system, but USR contains just a small subset of the /usr file system, with just enough utilities to install Minix. The complete /usr file system is split up into the USR.TAZ, SYS.TAZ and CMD.TAZ archives that are installed later to fill /usr.

Let’s suppose your first hard disk, which has device name **/dev/hd0**, has MS-DOS already present in the first primary partition (**/dev/hd1**), and some free space left after that. After Minix is installed in that free space the disk will look like this:

```
/dev/hd0      Whole hard disk #0
  /dev/hd1      MS-DOS C: drive
  /dev/hd2      Minix primary partition
    /dev/hd2a     Minix root partition
    /dev/hd2c     Minix /usr partition
```

/dev/hd0 is the sum of a partition table, /dev/hd1 and /dev/hd2. Likewise is /dev/hd2 the sum of a subpartition table, /dev/hd2a and /dev/hd2c. Read the "DEVICES" sections for more information on Minix devices.

3. INSTALLATION

You can install Minix automatically or manually as described the sections below. The end result is the same, but manual installation allows one to deviate from the preconfigured choices. You may wish to read the manual pages of the programs used below before you start. You may especially want to read **boot**(8) if your machine is different from what the majority buys, because you may need to set a few boot parameters to configure drivers. To do this type **ESC** to get to the Boot Monitor prompt, set the appropriate variables, use **save** to store the settings and **menu** to continue where you left off.

To install the system you need two diskettes: a bootable root diskette and a diskette full of binaries to use as **/usr**. These diskettes are named **ROOT** and **USR**. These two diskettes may also be combined on a single high density diskette. In that case the USR part is on the **c** partition.

Insert the ROOT diskette, boot the machine and type ’=’ to the menu. The Minix kernel is loaded and takes control when you see the copyright banner. After loading the root diskette into the RAM disk you will be asked to finish the name of the device to mount on **/usr**. Type **fd0c** for a diskette that contains both ROOT and USR, otherwise replace ROOT by USR and type **fd0**. Login as **root**.

4. AUTOMATIC INSTALLATION

Type **setup** to start the installation script. First it offers to install a national keyboard map. The names should be clear, except for **us-swap**, which swaps the CTRL and CAPS LOCK keys of a standard US style keyboard for people who believe that the natural place of CTRL is next to A. The default suggested between [and] is the US standard keyboard.

The next thing to do is to make a partition, for this you are placed in a partition table editor named **part**. This partition table editor is very easy to use (in the author's opinion), but you will probably hate it. You can move all over the place with the arrow keys, change values, and make a mess of your partition table real quick. So if you get into trouble, type 'q' to quit, 'n' to not write the table, and RETURN to start over. Use the '?' key to get help.

With the '+' and '−' keys you can select the disk device to install on, probably **/dev/hd0**, the first hard disk. Type 'r' to load the partition table of the selected disk. Either create one new partition by modifying a partition marked "None", or reuse an existing partition by changing its type to "MINIX" (hex code 81). The FIPS program can be used under MS-DOS to shrink an MS-DOS partition. FIPS splits the DOS partition in two, so one of the two can be used for Minix. You have to be absolutely sure which one. When in doubt, first use the FDISK program under DOS to delete the extra partition, and let Minix part create a new one. DO NOT use part to shrink an existing partition! Minix needs a partition of at least 20 Mb, but not larger than 128 Mb (Minix-86) or 1 Gb (Minix-386). The system needs 30 Mb in compiled state.

The script then wants to know the name of the partition you've created, this name is probably still visible on the screen (hd2, hd6, something like that.) The new partition table is reloaded into the disk driver, and the new Minix partition is carved up into two subpartitions, a 1440 kb root and the rest for /usr.

After making /usr, it is immediately put to use to replace the installation /usr file system so that you can remove the USR diskette and insert the ROOT diskette (unless they are one and the same). The root file system is filled with the contents of the ROOT diskette and slightly patched up to work on the hard disk (/etc/fstab.)

To compute the size of the so-called "second level block cache" you are asked to specify the RAM size of your machine. If you have plenty, i.e 4 Mb or more then simply hit RETURN, otherwise enter the size of your system RAM in kilobytes.

You can now skip the next section and move to "TESTING", but it may be instructive to read it anyway.

5. MANUAL INSTALLATION

The instructions that follow are at a very low level and require you to be very careful. The big advantage is that you know precisely what tools have been used and how everything works. The disadvantage is that you may easily make a mistake that either forces you to start over if you are lucky, or wipes out the contents of your hard disk if you are not. Only if you really want to do something different should you use a manual installation. Slavishly following the steps shown below will only make you end up with the same result as an automatic installation.

Run **part** to make partitions to load the system into. The best thing to do is to make one large primary partition of type "MINIX" and to carve this partition up into three subpartitions for root and /usr. The assumption is that you will use the second partition on the first hard disk, **/dev/hd2**, and that **hd2a** is the root subpartition and **hd2c** is /usr. If you want to use the first partition on the second hard disk for instance, then substitute hd6 and hd6[ac] for the above. On a SCSI disk it will be /dev/sd2 for the second partition on the disk at target 0. See the section on devices below, and the manual pages of **part**(8), **hd**(4), and **sd**(4). Start **part** and select the whole hard disk device (the "multiple of 5" device) that you want to install Minix onto. In our example it will be **/dev/hd0**.

Use **part** to make a single partition in the primary partition table of type "MINIX", then hit '>' on this new partition to make a subpartition table.

For the root subpartition you are advised to use 1440 kb exactly. You can make it larger if you want to, but it is advisable never to let the contents outgrow a floppy. (The ROOT diskette is a copy of a root file system, and will be used to fill your root subpartition.)

The second subpartition is either empty or a "scratch" partition. Minix no longer uses the **b** subpartition for anything useful anymore, but it has become customary to have root on **a** and /usr on **c**. (You are free to ignore this convention, of course.)

Use the rest of the partition for the /usr **c** subpartition.

When you are done check that /dev/hd2a is active (the * after the partition number) so you can boot from it later.

If your disk has bad blocks then don't put the root or scratch subpartition on top of them. Make sure the inode tables in the other partitions don't have bad blocks either. You can put the subpartitions out of order on the disk if that helps. Subpartition tables, other than the main partition table, are not sorted by the driver.

After making the partitions you do not have to reboot. The disk driver reloads the partition tables on the next access if the disk is not in use. (Open or mounted.)

To be able to boot from /dev/hd2a you must place a master bootstrap in /dev/hd2. It has been placed there by **part** if it told you that it was creating a new partition table, but

installboot −m /dev/hd2 /usr/mdec/masterboot

will put it there for sure.

You will start by making a file system for /usr and filling it partially. This may seem to be out of order, but you can't insert the ROOT floppy right now.

mkfs /dev/hd2c
readall −b /dev/hd2c | sh
mount /dev/hd2c /mnt
cpdir −v /usr /mnt

This will create a file system on /dev/hd2c, mount it on /mnt, and copy the contents of the USR floppy onto it. The call to **readall** marks bad blocks on the file system as unusable, you can omit this on a drive known to be spotless (IDE or SCSI.)

You can now use the new /usr in place of the USR floppy:

umount /dev/hd2c
umount /dev/fd0   # fd0c if combined
mount /dev/hd2c /usr

This little dance has freed up your floppy drive, so please remove the USR diskette and replace it by the ROOT diskette. Make a file system for the root with at least 512 inodes (files), and fill it from the floppy:

mkfs −i 512 /dev/hd2a
mount /dev/fd0 /fd0
mount /dev/hd2a /mnt
cpdir −v /fd0 /mnt
umount /dev/fd0

Remove **/mnt/etc/issue** to get rid of the "use setup" message that greets you when you boot, and edit the file **/mnt/etc/fstab** to name the devices Minix has been installed on. In our example it should look like this:

root=/dev/hd2a
usr=/dev/hd2c

Unmount the new root:

umount /dev/hd2a

Make it bootable:

installboot −d /dev/hd2a /usr/mdec/bootblock boot

The automatic script would now set the **rootdev** and **ramimagedev** boot variables. You can do this now using the **edparams** command, but it is easier to postpone it until the testing phase. The settings should be:

rootdev=hd2a
ramimagedev=hd2a

6. TESTING

By now a new Minix system is present on your hard disk. Time to see if it works. Leave the ROOT diskette in the drive and type **halt**. You are now going to use the power of the Boot Monitor on the diskette to boot the Minix partition on the hard disk. Use the monitor command **boot hd2** to boot the primary partition Minix has been installed in. (It is "hd2" in our example.) For a SCSI disk you will have to use a 'hd' name too. The monitor uses the BIOS, so you will have to treat it as a "normal" disk at this point.

The hard disk bootstrap is now showing the menu again. You can type '=' to start Minix, but you probably want to change the boot parameters. Hit **ESC** once more to get to the command prompt. The command **set** shows what the current parameters are. Here is an example that shows how to make a menu to either start Minix or boot MS-DOS:

minix(=,Minix) {boot}
dos(d,MS-DOS) {boot hd1}
save

MS-DOS is assumed to be in the first partition in the example above (hd1). When finished type **menu** to see if the menu looks right. If so hit '=' to start Minix.

7. ADDING PROGRAMS AND SOURCES TO /usr

The **setup** command can also be used to add files from floppy sets to the system. The **USR.TAZ** (programs and stuff), **SYS.TAZ** (system sources), and **CMD.TAZ** (commands sources) are all installed relative to the **/usr** directory, so the command to use three times is

setup /usr

Setup will ask for the size of data on the floppies, which is by default simply the entire floppy. Don't worry if you see a few "File exists" errors while extracting, as some directories already exist. You need the **USR.TAZ** set if you want a working Minix system, **SYS.TAZ** if you want recompile the system or study it, and **CMD.TAZ** if you also want the sources of the simple commands. On a disk space starved machine you could opt to do without the commands sources, as they are not absolutely necessary to understand Minix.

If your machine does not have enough memory to run **setup /usr** then type these commands manually:

cd /usr
vol /dev/fd0 | uncompress | tar xvfp −

8. NAMES

A standalone machine will have to be given a name. As **root** type

echo *name* **>/etc/hostname.file**

to change the host name of your machine to *name*.

9. ACTIVE ON BOOT

You may want to make the Minix partition active so that it is automatically booted. With DOS **fdisk** or Minix **part**, mark the primary partition that contains Minix active. Using the menu you made earlier you can boot either Minix or DOS at a keypress. You can even set timeouts. To boot Minix automatically after 5 seconds:

main() {trap 5000 minix; menu}

See **monitor**(8) for all the details on the monitor.

If you don’t trust this then you can rig up a diskette that boots the Minix partition when left in the drive:

installboot −m 2 /dev/fd0 /usr/mdec/masterboot

The number 2 indicates the hard disk partition that must be booted, you can use the numbers 1 to 9 for hd1 to hd9.

10. DEVICES

A crash course on the Minix devices in **/dev**: The two hard disks are named **hd0** and **hd5**. These "multiple of five" devices address the entire hard disk, from the first to the last byte. Each disk has four partitions, for disk 0 they are **hd1**, **hd2**, **hd3**, and **hd4**. And for disk 1 they are named **hd6**, **hd7**, **hd8**, and **hd9**. These partitions may contain file systems, **hd1** often contains the MS-DOS "C:" file system. Minix can use these partitions for file systems too, but you can also partition one of these "primary partitions" into four so-called "subpartitions". The subpartitions of **hd1** are named **hd1a**, **hd1b**, **hd1c**, and **hd1d**. The other partitions may have four subpartitions that are named in the same way by adding a letter from **a** to **d**. So one disk may have four partitions, and 16 subpartititions total. SCSI disks are named in the same way, from **sd0** to **sd39d** for all possible devices for all eight SCSI targets. The two floppy disks are **fd0** and **fd1**. Each may have four partitions named **fd0a**, **fd0b**, ... **fd1d**. The command **MAKEDEV** knows how to make devices, and **DESCRIBE** can tell you what an unknown device may be, or even what all devices in **/dev** may be if called without arguments. Devices are described fully in **dev**(4), and in the device specific manual pages like **fd**(4) and **hd**(4).

11. EDITORS

The editors available are **elvis** (a **vi** clone), **elle** (a simple **emacs** clone), and the old Minix **mined** editor. Of these editors only elvis can recover your file after a system crash. Only **mined** is available at installation time. (All you need to know about mined right now is that CTRL-X gets you out of it.)

12. INSTALLING ON A SCSI DISK

Using a disk other than a **hd** disk complicates things a bit. The Boot Monitor uses the BIOS, so it names all disks with **hd** names. So it is **boot hd1** to boot partition 1, and **ramimagedev=sd2a** to tell Minix its root partition. If you have both a normal and a SCSI disk then the disks may be **hd0** and **hd5** to the Monitor, and **hd0** and **sd0** to Minix.

13. NATIONAL KEYBOARDS

The directory **/usr/lib/keymaps** contains keymap tables for several national keyboards. If you have a German keyboard for instance, then

loadkeys /usr/lib/keymaps/german.map

will load the German key translation table into the keyboard driver. Copy the map to **/etc/keymap** once Minix is installed on the hard disk, because having to type a key sequence like one of these:

loadkezs −usr−lib−kezmaps−german.map
loqdkeys =usr=lib=key,qps=french.,qp

on a reboot gets a bit annoying after a while. Send corrections and new keymaps to the person named below. (Do not send a Dutch keymap, buy yourself a real keyboard instead.)

SUGGESTIONS

Below are a few useful suggestions. Some of the information can be of use in other situations than described here.

14. VIRTUAL CONSOLES

Hold down the ALT key and press the left or right arrow key, F1, or F2. This switches the console between two login sessions. (Unless you have an old mono adapter, because virtual consoles sit in video memory, and a mono adapter only has memory for one.)

Note that kernel messages, including function key output, only appear on the first console. This may be confusing, but it keeps the other consoles clean.

15. LOW ON MEMORY

The normal installation requires that you have enough memory for a large RAM disk. You can still install Minix normally if you either have a high density diskette drive for a combined root+usr floppy, or you have two floppy drives of at least 720 kb. Before booting you have to set the variable **rootdev** to the same value as **ramimagedev**. This is slower then a RAM disk, but saves a lot of memory.

The automatic installation script knows how to handle this new situation. If you install manually then you have to use

> **cpdir −vx / /mnt**

to copy the root device to disk. When it is time to fill /usr and you only have one floppy drive then hit DEL to get out of the installation script and reboot as described in "TESTING". You can then finish the installation manually.

16. LOW ON MEMORY AND ONLY ONE 720 KB FLOPPY DRIVE

If you only have one 720 kb floppy drive and your system is low on memory then you can use the TINYROOT boot image. This image contains a small kernel with only the BIOS disk driver, and a small root file system. You can use this disk to boot your machine. Use the normal ROOT to install the root file system. Keep booting your machine with TINYROOT until you have compiled a small kernel for your system. Use the **rootdev** boot variable to select the hard disk root file system. Do **not** use TINYROOT for anything other than booting, always use ROOT when mentioned.

17. FLOPPY DRIVE 1 IS A HIGH DENSITY DRIVE

If you would like to install from floppy drive 1 then you need to copy at least one sector from the USR image onto a diskette for drive 0. The USR bootstrap has been rigged to boot the other drive.

18. INSTALLING ON A SECOND HARD DISK

Minix doesn't care if it is installed on the second disk of a system with two disks. The only problem is to get it booted. You can either rig up a diskette to boot Minix as shown earlier, or you can use the same trick on the first disk. The command

> **installboot −m 5 /dev/hd0 /usr/mdec/masterboot**

will lock the first disk into booting the second disk. Note that this command modifies the disk outside a Minix partition, overwriting a bit of code that has likely been put there by DOS fdisk. First verify that the Boot Monitor can boot a DOS partition, because then the Minix master bootstrap can do it too.

19. LOTS OF MEMORY ON A 286

You will have a hard time making Minix run out of 3 Mb memory. Memory you can spare can be used for a "second level block cache" on the RAM disk. The File System uses the second level cache to store copies of disk blocks that are pushed out of the normal (primary) block cache. The size of the primary cache is compiled into the FS server, but the size of the second level cache can be set with the **ramsize** boot variable. Set it to a number between 0 and 512. 512 kilobytes is enough to keep most of the compiler cached.

20. LOTS OF MEMORY ON A 386+

Processes can be as big as you'd like on a 386, but in practice 4 Mb is nice. The installation script sets up a second level cache for Minix-386 of up to 1024 kilobytes. This is because the default file system cache is only 80 kb. Your first point of call is to get rid of the poorly performing second level cache and to assign the memory used by it to the normal block cache by enlarging the appropriate **NR_BUFS** and **NR_BUF_HASH** constants in <minix/config.h> with as much as you can spare. (1024 for NR_BUFS is the minimum to keep **cc −c** cached. 2048 is then a nice value for NR_BUF_HASH.) Disable the second level cache, compile a new kernel, reboot and set **ramsize** to 0.

21. LOTS OF DISK SPACE

The maximum file system size is 1 Gb for Minix-386 and 128 Mb for Minix-86. (Minix-86 can handle larger file systems, but **fsck** can't check them.) Note that a Minix file system can only contain 65535 inodes (files), so the average file should be 16 kb to completely fill it. It may be better to make two smaller file systems. Besides, fsck takes forever on a large file system.

SYSTEM ADMINISTRATION

The system has been set up with the idea that working as root is a bad thing to do. As root you are in no way protected from doing stupid things. So don't do development as root, but work as **bin**! Only in exceptional cases do you want to become root. Being root is fun for wannabe hackers; administrators know better.

To make life easier for bin, some programs like **su**(1), **install**(1) and **shutdown**(8) treat bin and other members of the operator group as special and allow them the privileges of root. (One is an operator if one's group id is zero.) Operators should share the shadow password of root by having **##root** in their password field. This way they all have one face (password) to the outside world, forming no greater security risk than root alone.

The home directory of bin contains one important Makefile. You can use it to recompile all the commands and libraries of the system. Type **make** to see the usage message. If you want to compile just one command then you can simply type **make** to do so. To put it in its proper place you have to type **make install**. Read the Makefiles in the **commands** and **lib** subdirectories to understand how everything is put together. If you are tight on memory then **make** may fail to traverse down the source tree and also compile things. You will have to type **make** in each subdirectory. You can run make in /usr/src at the end to see if you've missed something or not.

The login shell of bin is **ash**, the BSD shell. It has been modified to offer simple line editing using the **editline**(3) library. **Ash** is rather big, so you may have to change bin's shell back to **/bin/sh** with **chsh**(1) if you are low on memory. Do not change root's shell to ash, and do not replace /bin/sh by ash. It may run out of memory at the wrong moment.

The kernel is not compiled from the master Makefile. To make a new kernel you have to step into the **tools** directory. There you can run four different make commands:

make This makes all the different kernel parts and combines them in the file named **image**.

make fdboot
: As above and then makes a boot floppy that you can use to restart your system with. You are prompted for the floppy device name.

make hdboot
: First makes the image file and then copies it into the directory **/minix**. If there are already two images in that directory then the newest image will be removed to make space for this newer image. It is assumed that the oldest image is the most stable system image, one that always works, and that the newest image is experimental. Check beforehand what **/minix** contains before you run **make hdboot**. Remove the oldest image if you want another image to become the stable image. The Boot Monitor chooses the newest image in **/minix** to boot. You can use the monitor command **ls minix** to view the images present, and set the **image** variable to the full name of the image you want to use instead if the newest doesn't work. The images in **/minix** are named using the Minix release and version numbers with an extra revision number added to distinguish the images.

The first new kernel you would like to make is one configured for your system. The kernel you are running now contains several hard disk drivers you don't need, and it does not have a TCP/IP server that you may want to have. In <minix/config.h> you can find a number of **ENABLE_***XXX* variables that can be set to **0** to exclude, or **1** to include a particular driver. Another driver related variable is **DMA_SECTORS**. This variable sets the size of a buffer used by DMA based disk drivers (all but the floppy, AT/IDE, and Adaptec drivers). Raise its value to greatly improve throughput, especially writing. A value of 16 shows good results. (The BIOS driver benefits most, because it is a long way to the BIOS from protected mode, especially from 286 protected mode.) You can increase **NR_CONS** if you want to have more virtual consoles. Having more consoles costs little memory, because all the consoles are kept in video memory. Scrolling speed of the console will go down if more virtual consoles share the available memory. CGA cards have space for 4 consoles, EGA and VGA can have 8 consoles. The **NR_PTYS** variable sets the number of pseudo-ttys. You need pseudo-ttys to be able to login remotely over a network with the **rlogin** command. Each remote login session needs one pseudo-tty. If you fear that the system will now run out of processes then increase **NR_PROCS**. Configuring a new kernel is sometimes not enough to enable new devices, you sometimes need to use the **MAKEDEV** command to make new device files in **/dev**. For pseudo-ttys you also have to check if **/etc/ttytab** mentiones the new devices.

New additions to the system can be made in the **/usr/local** tree. An empty directory tree has been set up for you and binaries and manual pages are already in the search paths. You can make a new user entry with the **adduser** command.

The **TZ** variable in **/etc/profile** tells the time zone offset from the wall clock time to GMT. You have to change it for your time zone. (See **TZ**(5).)

The function keys produce debug dumps, showing various interesting data about the system. F1 lists processes and F5 shows ethernet stats, which may be of use now. Read **console**(4) to know all the details of the screen and keyboard.

22. SYSTEM SHUTDOWN

You can't just turn a Minix system off. Minix must be told to flush the modified data in the file system cache first. The following commands/keystrokes can be used to exit Minix properly:

shutdown
: First alert all users and then all processes of the impending shutdown then halt or reboot the system in one of various ways. See **shutdown**(8).

reboot / halt
: Alert all processes of the system shutdown then reboot or halt.

CTRL−ALT−DEL
: Halt the system by running **shutdown −h now**.

Minix halts by returning to the Boot Monitor, Minix reboots by instructing the monitor to reboot Minix. (Minix is just a subprocess to the monitor.) Either halt Minix and use monitor commands to escape Minix, or use **shutdown −R** to reset the system.

FILES

/usr/ast Honorary home directory of Andew S. Tanenbaum. Doubles as the place where the default setup for a new user is found.

SEE ALSO

monitor(8), **boot**(8), **part**(8), **mkfs**(1), **mount**(8), **M**(8), **fstab**(5), **hier**(7), **console**(4), **dev**(4), **adduser**(8), **TZ**(5), **mkdist**(8), **shutdown**(8).
"Operating Systems − Design and Implementation" by Andrew S. Tanenbaum.

NOTES

The notation **<***file***.h>** refers to a C language include file in /usr/include.

Root and **bin** do not have the current directory in their program search path to avoid executing programs left around by malicious people. This means that to run **foo** from the current directory, **./foo** must be typed.

Some of the commands have changed since earlier Minix versions. For instance **mkfs** doesn't need a size argument anymore, and **vol** automagically determines if it needs to read or write. Keep this in mind if you use an older Minix version to examine the newer system.

BUGS

There are many PS/2 models, all different. Some will run Minix, some won't, some crippled if you lie to Minix by setting **processor** to **86**. Almost no PS/2 has a standard disk, so setting **hd** to **esdi** or **bios** will be necessary.

While testing a full library rebuild of this distribution it sometimes happened that some things were not put back into the library. This seems to be fixed, but we do not understand why the fix fixed the problem. So if you see strange "undefined" errors when compiling a program after a library rebuild then run **make install** again in **/usr/src/lib/** to try and add the missing pieces.

Except for the floppy driver none of the DMA based drivers know about DMA being limited to a 24 bits address, i.e. the first 16 Mb. So under Minix-386 you run a slight risk that a **tar** or **dd** command may use a buffer above 16 Mb for reading or writing to a character device. This only happens if the low 16 Mb is taken by some huge processes, and you have more than 16 Mb, of course.

AUTHOR

Kees J. Bot (kjb@cs.vu.nl)