REFERENCE MANUAL FOR THE MINIX 1.5 DEMONSTRATION DISK

ANDREW S. TANENBAUM

Prentice Hall, Inc

1

INTRODUCTION

Every computer needs an operating system to manage its memory, control its I/O devices, implement its file system and provide an interface to its users. Many operating systems exist, such as MS-DOS, OS/2, and UNIX. This manual provides a very brief introduction to another operating system, MINIX. It is intended to accompany the MINIX demonstration diskette.

Although MINIX was inspired by the well-known AT&T UNIX operating system, its design and implementation are completely new. It does not contain even a single line of AT&T code: not in the operating system, not in the C compiler, and not in any of the nearly 200 utility programs supplied with MINIX. For this reason, it is possible to include not only all the binary programs, but, virtually all the source code of the operating system and utilities as well. In this way, people can study MINIX in detail to learn how a modern operating system is constructed, and can also modify it to suit their own tastes if need be.

Before getting started, we would like to point out that this manual and the accompanying demonstration diskette only deal with a tiny fraction of MINIX, just to give the flavor of the system. If your favorite feature (e.g., the Berkeley *vi* editor) is not present here, that does not mean that it is also absent from the full system. The standard MINIX distribution for the IBM PC, for example, is 17 diskettes, whereas the demonstration program is only 1 diskette. Similarly, the manual that comes with MINIX is 680 pages, including a cross-referenced listing of the operating system source code itself, in C.

Some of the differences between the demonstration system and full MINIX are given in the table below.

Item	**Demonstration disk**	**Full MINIX**
Complete operating system provided?	Yes	Yes
Complete shell provided?	Yes	Yes
Printer supported?	Yes	Yes
RAM disk supported?	Yes	Yes
Floppy disk supported?	Yes	Yes
Hard disk supported?	Yes	Yes
RS-232 serial lines supported?	Yes	Yes
Real mode supported?	Yes	Yes
Protected mode supported?	No	Yes
Ethernet supported?	No	Yes
Version 7 system calls supported?	Yes	Yes
Operating system source provided?	No	Yes
Utility program source provided?	No	Yes
Multiprogramming supported?	Yes	Yes
C compiler provided?	No	Yes
Mountable file systems supported?	Yes	Yes
Remote login supported?	No	Yes
Network file transfer supported?	No	Yes
Number of Editors provided:	1	5
Communication software provided:	No	Kermit, zmodem
Maximum number of simultaneous users:	1	3
Approximate number of utilities:	25	190

In addition to the IBM version (8088, 8086, 80286, 80386), MINIX is also available for the Atari, Amiga, and Macintosh. A version for the SPARC is in the works and will be available in 1991.

1.1. BOOTING MINIX

The steps below will tell you how to boot MINIX on your computer. One word of warning, however, concerning running MINIX on IBM clones. Although it will work perfectly with all clones that are compatible with the IBM *hardware*, there are a small number of clones that deviate from the IBM hardware and mask these differences in the BIOS. Since MINIX does not generally use the BIOS, it may not

run on these machines. If the demonstration disk works on your system, it is very likely that full MINIX will work too, and vice versa.

The problem you are most likely to encounter with the demonstration diskette is nonstandard video cards. If the screen goes blank every 25 lines (which will happen with some EGA cards that are not IBM compatible), hit the F3 key once to enable software scrolling.

Since there are no clones for the Atari, Amiga, and Macintosh, these problems do not arise there, although occasional problems do arise with peripherals, such as hard disks, that are not compatible with the vendor's.

This said, we can now boot MINIX in the following steps.

1. Turn off your computer.
2. Insert the demonstration disk in drive 0 (also called drive A).
3. Turn the computer on.
4. Wait for the menu, then hit the = key (i.e., equal sign) [IBM only].
5. When it asks for the date, please enter a 12-digit number in the form MMDDYYhhmmss, followed by a carriage return. For example, 3:35 p.m. on July 4, 1976 was 070476153500.
6. When the *login:* prompt appears, type:

   ```
   root
   ```

   Please note that in this manual, lines printed in the Helvetica typeface, as above, are lines that either you are to type in (literally) or lines that the computer will display. It will always be clear from the context which is which. Always type a carriage return (ENTER key) after each line.
7. The computer (actually the shell) will now display:

   ```
   #
   ```

   to show that it is ready to accept a command.
8. To see what programs are available, type:

   ```
   ls –l /bin
   ```

   A list of file names of executable programs will be displayed. To keep the list from scrolling out of view, type CTRL-S to stop the scroll and CTRL-Q to restart it. (CTRL-S means depress the CTRL key, and while holding it down, depress the S key.) You can also use:

   ```
   ls –C /bin
   ```

   to get a short listing.

9. At this point you can run programs, make tests and so on. Some examples are given below. If you are familar with UNIX you can try the various programs in */bin* as most will be familiar.

10. When you are done using MINIX, type CTRL-D to log out. *login:* prompt will appear, and you can now turn off the computer or boot a different operating system.

1.2. MAKING AND MOUNTING FILE SYSTEMS

As distributed, the demonstration disk is so full that there is not much room for user files. Thus it is necessary to make a file system on disk and copy some files there. You may use either a diskette or a hard disk.

The first step is to decide how large a file system you want to make. The size of the file system is in units of 1K blocks. Thus a 360K 5.25-inch diskette will hold 360 blocks, a 1.2M 5.25-inch diskette will hold 1200 blocks, a 720K 3.5-inch diskette will hold 720 blocks, and a 1.44M 3.5-inch diskette will hold 1440 blocks. If you wish to use a hard disk, pick an unused partition and determine its size in 512-byte sectors using your present operating system's software (e.g., *fdisk*). Then divide the number of sectors by 2 to get the number of blocks, rounding downward to an integer if there are an odd number of sectors. Finally, subtract 1 from this number if and only if you are using partition 1, to account for the boot block. The number of blocks should be between 360 and 32767.

If you are using a diskette, insert it into drive 0 (drive A). To make a file system, please use the *mkfs* command. Below are several examples. Do not type the sharp sign or the text following it, as this is simply commentary.

```
mkfs  /dev/fd0   360          # For a 360K 5.25-inch diskette
mkfs  /dev/at0  1200          # For a 1.2M 5.25-inch diskette
mkfs  /dev/ps0   720          # For a 720K 3.5-inch diskette
mkfs  /dev/PS0  1440          # For a 1.44M 3.5-inch diskette
mkfs  /dev/hd1  6800          # For a 6800-block hard disk partition 1
mkfs  /dev/hd2 13600          # For a 13600-block hard disk partition 2
mkfs  /dev/hd3 12041          # For a 12041-block hard disk partition 3
```

You can also use drive 1 (drive B) by replacing the 0 at the end of the device name by a 1 (e.g., */dev/fd1*).

Having made an empty MINIX file system, you can now mount it, make directories on it, and copy files to it, and remove files from the RAM disk. To perform this work, type the following commands (again, without typing the comments):

```
/etc/mount  /dev/fd0  /usr                  # Mount the new file system
mkdir  /usr/bin                             # Create a directory for binaries
cp  /bin/*  /usr/bin                        # Copy all the binaries
rm  /bin/*                                  # Free up space on the RAM disk
cp  /usr/bin/mined  /usr/bin/sh /bin        # Restore 2 programs
df                                          # See how much space you have
```

You can now begin using MINIX. However, if you stop using it and wish to reboot it later, you need not go through this entire procedure again. After rebooting, mount the diskette or partition and remove the files from */bin* that you do not need using *rm*. It is best to restore at least *mined and sh* to */bin*, however, for speed.

1.3. USING THE MINED EDITOR

MINIX comes with five editors as follows:

1. Mined - A simple full screen editor
2. Elle - An editor based on EMACS
3. Vi - A clone of the famous Berkeley *vi* editor
4. Ex - A clone of the famous Berkeley *ex* editor
5. Ed - A simple line-oriented editor

The demonstration disk comes only with *mined* for two reasons. First, *mined* is very easy to learn. In 15 minutes you will be an expert. Second, it is quite small; editors like *vi* and *elle* would take up too much space on the demonstration disk.

In this section we will give an introduction to *mined*. See the manual page in Chap. 2 for more details. Since the keyboard layouts for each of the machines differs somewhat, we will describe the IBM keystrokes here, but the 68000 version will be given in parentheses. To start *mined* with a sample file type:

```
cp  /etc/text  mytext
mined  mytext
```

The first line copies the file */etc/text* to *mytext* so we can modify it without affecting the original. The second line starts the editor. Within a second your screen will contain 24 lines of a well-known poem by Lewis Carroll, the author of *Alice in Wonderland.* The cursor will be positioned at the upper left-hand corner of the screen, The bottom line will tell you the name of the file, its length in lines, and its length in characters, displayed in reverse video. The bottom line will vanish as soon as the first key is struck, but can be recalled by hitting the middle key in the numeric keypad (i.e., the key above the ‘‘2’’ and below the ‘‘8’’ (F5 on 68000).

You can move the cursor around the screen using the arrow keys on the numeric keypad. The HOME key (on the 68000, also HOME) puts it back at the top of the file. The END key (on the 68000, F6) moves it to the end of the file. The PGDN key (on the 68000, F3) scrolls the screen down by 24 lines. The PGUP key (on the 68000, F4) scrolls it up by 24 lines. Using these keys you can position the cursor anywhere in the file. Try them now.

You can enter new text by just typing it. The new text will appear under the cursor, and the cursor will be advanced one position. There is no automatic wraparound, so you must terminate each line using a carriage return (ENTER key). Although *mined* can handle lines greater than 80 characters, to keep things simple, restrict your input lines to less than 80 characters for the moment.

If you make a typing error, use the BACKSPACE key to erase the character to the left of the cursor. By moving the cursor around the screen using the arrows, you can erase any character by just positioning the cursor to the right of the character to be erased. The DEL key erases the character under the cursor.

With only the information given so far, you can produce any document you need. However, other commands exist to speed up editing. For example, you can move the cursor to the start or end of the current line by typing CTRL-A or CTRL-Z, respectively. You can go forward or backward one word using CTRL-F or CTRL-B, respectively. You can scroll the screen down or up one line using CTRL-D or CTRL-U, respectively. Try these.

In addition to the BACKSPACE and DEL keys, other methods are provided for erasing text. CTRL-N deletes the next word (i.e., the one to the right of the cursor). CTRL-P deletes the previous word. CTRL-T deletes the tail of the current line (i.e., all text from the cursor to the end of the line). To delete a block of text, first position the cursor at the start of the block and type CTRL-@ to mark the start. Then move the cursor one character after the end of the block and type CTRL-K to kill it. The text will vanish from the screen, but be saved in a hidden buffer. You can now move the cursor anywhere you want and type CTRL-Y to yank the contents of the hidden buffer out and insert it at the cursor. Yanking does not change the hidden buffer, so you can insert the same text in multiple places. Try these commands to see how they work.

You can search forward for a given piece of text by hitting the large plus sign (+) on the numeric keypad (on the 68000, F1). You will be prompted for a string. After entering the string, type a carriage return. The numeric minus sign (−) searches backwards (on the 68000, F2). If you are familiar with the magic characters allowed by the standard *ed* editor in searches, you can use those here too.

Although there are some more commands, with just these you can work quite efficiently. Practice a little bit now. When you have finished editing your file, type CTRL-W to write the file back to the (RAM) disk. Then exit using CTRL-X. If you type CTRL-X without first giving CTRL-W, you will be asked if you want to save it. Answer with ‘‘y’’ for yes and ‘‘n’’ for no. After exiting the editor, you will be back in the shell, indicated by the sharp sign.

You can create a new file, say *foobar,* by typing:

```
mined foobar
```

After entering the text and typing CTRL-W and CTRL-X, the file will be present on the disk. Please note that only a small amount of free RAM disk space is available initially, which limits the number and size of the files you can create. You can remove programs that you do not need with *rm*. This will give you more space.

1.4. USING THE SHELL

The MINIX shell is functionally equivalent to the standard Version 7 (Bourne) shell. In this section we will give some examples of how it is used. For more information, consult one of the many books on UNIX. MINIX supports a hierarchical directory system. Every directory has an absolute path name, starting at the root directory /. To change to the root directory, type:

```
cd /
```

You can find out what files and directories are located here by typing:

```
ls –l
```

You can create a new directory *mydir* by typing:

```
mkdir mydir
```

You can now change to that directory to make it your working directory with:

```
cd mydir
```

Try doing this, and then use *mined* to create a small file called *file1* here. You can examine the contents of this directory by typing:

```
ls –l
```

Now create a new directory inside *mydir* by typing:

```
mkdir newdir
```

Change to *newdir* using *cd* and create a file *file2* there. You can find out where you are by typing:

```
pwd
```

Now change back to the root directory and examine your handiwork by typing:

```
cd /
ls –l mydir
ls –l mydir/newdir
```

Note the use of the slash character to indicate subdirectories. This choice conforms to UNIX usage, but is different than MS-DOS, which uses a backslash.

To see how much space you have left, type:

```
df
```

To get rid of all the files and directories you have just created, type:

```
rm –rf /mydir
```

Now try *df* again to see how much space you have recovered.

Some of the MINIX programs provided on the demonstration disk are **filters**. That is, they read an input file, called **standard input**, process it in some way, and write the results on **standard output**. To see an example of this, type:

```
head –15 /etc/text
```

which will extract the first 15 lines of */etc/text* and write them on standard output, which, by default, is the screen. To redirect the output to a file, *x*, type:

```
head –15 /etc/text >x
```

You can examine *x* using *mined* or by copying it to the screen using:

```
cp x /dev/tty
```

In a similar way, try:

```
tail –5 /etc/text
```

to see the last 5 lines of */etc/text*.

MINIX allows filters to be combined in a simple way. If you type:

```
head –15 /etc/text | tail –5
```

head will extract the first 15 lines of the file and pass them to *tail*. In other words, the input to *tail* will be the output from *head*. Thus *tail* will get lines 1 through 15 as its input, and extract the last 5 lines as its output. This will result in lines 11 through 15 of the original file appearing on the screen. When two (or more) programs are connected this way, the construction is called a **pipeline**.

A still more complicated example is:

```
head –15 /etc/text | tail –5 | sort >y
```

which first extracts 15 lines from the file, then takes the last 5 of these and passes them to *sort*, which sorts them alphabetically (using the ASCII collating sequence), and writes the result on the file *x*. Try this command, and make up your own pipelines using these programs and also *rev*, which reverses the characters in a line. The first character becomes the last one and the last one becomes the first one (e.g., HORSE becomes ESROH).

You can compare two files to see if they are identical using *cmp*. Another

useful program is *comm*, which expects two input files, each containing a sorted list of items. The output of *comm* tells which items occur in only the first file (left-hand column), occur in only the second file (middle column), or in both files (right-hand column). The output of *comm* can be displayed, used as the input of another filter, or redirected to an output file.

It is possible to see the output and save it at the same time using *tee* as follows:

```
head /etc/text | sort | tee x
```

This pipeline writes the sorted output onto the file *x*, but also writes it on standard output (the screen).

You can time a command using *time*:

```
time sleep 5
```

The *sleep* command simply waits 5 seconds before returning, and *time* reports on the real time, user time, and system time used to carry out the command. For *sleep*, the user and system times will usually be 0.0.

MINIX supports **multiprogramming** (sometimes called **multitasking**). Multiprogramming allows multiple processes to run simultaneously. A process can be put in the background by appending its command with an ampersand (&) like this:

```
sleep 15 &
```

Notice that after typing a command, the shell will respond with a number, the process id of the background process just started. If you change your mind, you can terminate a background process using the *kill* command. For example, if the previous command caused the shell to output ‘‘28’’ as the process id, the command

```
kill 28
```

would terminate process 28. Process id 0 can be used to kill all background processes.

You can see how many background processes you have by hitting the F1 key on the PC or CTRL-ALT-F1 on the Atari. As an example of multiple processes, try typing:

```
time sleep 60 & time sleep 50 & time sleep 40 &
```

This command will start up six background processes: three running *time* and three running *sleep*. Try killing them off one at a time using *kill* and hit F1 on the IBM PC (or CTRL-ALT-F1 on the Atari) each time to see how many are left. Needless to say, you can start any process off in the background, not just *sleep*.

While a complete tutorial on shell programming is beyond the scope of this manual, we will just point out that the shell supplied with the demonstration disk is a full Bourne shell. Consult any book on UNIX to find out more about using it. As a simple example, create the following file, *compare* using *mined*:

```
case $# in
    0)  echo Compare: You have not supplied any arguments. At least 2 needed.
        exit 1
        ;;
    1)  echo Compare: You have only supplied one argument. At least 2 needed.
        exit 1
        ;;
esac

file=$1
shift
for i
do
     if cmp –s $i $file 2>/dev/null
       then echo $i is the same as $file
       else echo $i does not exist or is different from $file
     fi
done
```

To test this shell script, create a file *x* along with several other files. Some of these should be copies of *x*; the rest should be different. For example, type:

```
cp  /etc/rc  x
cp  /etc/text  a
cp  x  b
cp  /etc/passwd  c
```

To run your new shell script, type:

```
sh  compare  x  a  b  c
```

The shell script will compare the second, third, fourth (and subsequent, if present) files with the first one, and tell which are the same and which are different. Also try it with no arguments and with one argument. In full MINIX, you could make *compare* executable with *chmod*. See Chap. 2 for more programs and try them all.

1.5. PRINTING

You can print files using *lpr*. For example:

```
lpr  /etc/text  &
```

will print */etc/text* using the centronics printer port. Note that *lpr* is not a spooler, so you cannot start up the next *lpr* until the first one is finished, but by starting it up in the background, as shown, you can do other things while it is printing.

2

MANUAL PAGES

This chapter contains the manual ''pages'' for those MINIX programs included on the demonstration disk. For each command, its name, syntax, and flags are given, as well as at least one example of its usage, and finally a description of what it does. Although most commands are available in all versions of MINIX A few are not. These commands have square brackets in their **Command** lines listing the versions in which they are present.

The following conventions are used in the **Syntax** lines below. Items printed in **bold typeface** are to be typed exactly as they appear here. Items printed in *italics typeface* are to be replaced by user-chosen directory names, file names, integers, and so on. Punctuation marks are printed in the roman typeface. The sharp sign (#) is used below to indicate the start of a comment. The text following the sharp sign is not part of the command. It is provided to help explain the command.

2.1. MANUAL PAGES

Command: **cmp – compare two files**
Syntax: **cmp** [**–ls**] *file1 file2*
Flags: **–l** Loud mode. Print bytes that differ (in octal)
–s Silent mode. Print nothing, just return exit status

Examples: cmp file1 file2 # Tell whether the files are the same
cmp –l file1 file2 # Print all corresponding bytes that differ

Two files are compared. If they are identical, exit status 0 is returned. If they differ, exit status 1 is returned. If the files cannot be opened, exit status 2 is returned. If *file1* is – , *stdin* is compared to *file2*.

Command: comm – print lines common to two sorted files
Syntax: **comm** [**–123**] *file1 file2*
Flags: **–1** Suppress column 1 (lines present only in *file1*)
–2 Suppress column 2 (lines present only in *file2*)
–3 Suppress column 3 (lines present in both files)
Examples: comm file1 file2 # Print all three columns
comm –12 file1 file2 # Print only lines common to both files

Two sorted files are read and compared. A three column listing is produced. Files only in *file1* are in column 1; files only in *file2* are in column 2; files common to both files are in column 3. The file name – means *stdin*.

Command: cp – copy file
Syntax: **cp** *file1 file2*
cp *file ... directory*
Flags: (none)
Examples: cp oldfile newfile # Copy *oldfile* to *newfile*
cp file1 file2 /usr/ast # Copy two files to a directory

Cp copies one file to another, or copies one or more files to a directory. A file cannot be copied to itself. *Cp* is much faster than *cat* or *dd* and should be used for copying when it is applicable.

Command: date – print or set the date and time
Syntax: **date** [**–q** [[*MMDDYY*]*hhmm*[*ss*]]
Flags: **–q** Read the date from *stdin*
Examples: date # Print the date and time
date 0221921610 # Set date to Feb 21, 1992 at 4:10 p.m.

Without an argument, *date* prints the current date and time. With an argument, it sets the date and time. *MMDDYY* refers to the month, day, and year; *hhmmss* refers to the hour, minute and second. Each of the six fields must be exactly two digits, no more and no less.

Command: **df – report on free disk space and i-nodes**
Syntax: **df** *special ...*
Flags: (none)
Examples:
```
df /dev/ram              # Report on free RAM disk space
df /dev/fd0 /dev/fd1     # Report on diskette space
df                       # Report on all mounted devices
```
The amount of disk space and number of i-nodes, both free and used is reported. If no argument is given, *df* reports on the root device and all mounted file systems.

Command: **echo – print the arguments**
Syntax: **echo** [**–n**] *argument ...*
Flags: **–n** No line feed is output when done
Examples:
```
echo Start Phase 1       # ''Start Phase 1'' is printed
echo –n Hello            # ''Hello'' is printed without a line feed
```
Echo writes its arguments to standard output. They are separated by blanks and terminated with a line feed unless **–n** is present. This command is used mostly in shell scripts.

Command: **head – print the first few lines of a file**
Syntax: **head** [*–n*] [*file*] ...
Flags: *–n* How many lines to print
Examples:
```
head –6                  # Print first 6 lines of stdin
head –1 file1 file2      # Print first line of two files
```
The first few lines of one or more files are printed. The default count is 10 lines. The default file is *stdin*.

Command: **kill – send a signal to a process**
Syntax: **kill** [*–n*] *process*
Flags: *–n* Signal number to send
Examples:
```
kill 35                  # Send signal 15 to process 35
kill –9 40               # Send signal 9 to process 40
kill –2 0                # Send signal 2 to whole process group
```
A signal is sent to a given process. By default signal 15 (SIGTERM) is sent. Process 0 means all the processes in the sender's process group.

Command: **login – log into the computer**
Syntax: **login** [*user*]
Flags: (none)
Example: login ast # Login as ast

Login allows a logged in user to login as someone else without first logging out. If a password is needed, *login* will prompt for it.

Command: **lpr – copy a file to the line printer**
Syntax: **lpr** [*file*] ...
Flags: (none)
Examples: lpr file & # Print *file* on the line printer
pr file | lpr & # Print *stdin* (*pr*'s output)

Each argument is interpreted as a file to be printed. *Lpr* copies each file to */dev/lp*, without spooling. It inserts carriage returns and expands tabs. Only one *lpr* at a time may be running.

Command: **ls – list the contents of a directory**
Syntax: **ls** [**–ACFRadfgilrstu**] [*name*] ...
Flags: **–A** All entries are listed, except . and ..
–C Multicolumn listing
–F Put / after directory names
–R Recursively list subdirectories
–a All entries are listed, even . and ..
–d Do not list contents of directories
–f List argument as unsorted directory
–g Group id given instead of user id
–i I-node number printed in first column
–l Long listing: mode, links, owner, size and time
–r Reverse the sort order
–s Give size in blocks (including indirect blocks)
–t Sort by time, latest first
–u Use last usage time instead of modification time
Examples: ls –l # List files in working directory
ls –lis # List with i-nodes and sizes

For each file argument, list it. For each directory argument, list its contents, unless **–d** is present. When no argument is present, the working directory is listed.

Command: **mined – MINIX editor**
Syntax: **mined** [*file*]
Flags: (none)
Examples:

```
mined /user/ast/book.3      # Edit an existing file
mined                       # Call editor to create a new file
ls –l | mined               # Use mined as a pager to inspect listing
```

Mined (pronounced min-ed) is a simple full-screen editor. When editing a file, it holds the file in memory, thus speeding up editing, but limiting the editor to files of up to about 35K. Larger files must first be cut into pieces by *split*. Lines may be arbitrarily long. Output from a command may be piped into *mined* so it can be viewed without scrolling off the screen, that is, *mined* can be used as a pager.

At any instant, a window of 24 lines is visible on the screen. The current position in the file is shown by the cursor. Ordinary characters typed in are inserted at the cursor. Control characters and keys on the numeric keypad (at the right-hand side of the keyboard) are used to move the cursor and perform other functions.

Commands exist to move forward and backward a word, and delete words either in front of the cursor or behind it. A word in this context is a sequence of characters delimited on both ends by white space (space, tab, line feed, start of file, or end of file). The commands for deleting characters and words also work on line feeds, making it possible to join two consecutive lines by deleting the line feed between them.

The editor maintains one save buffer (not displayed). Commands are present to move text from the file to the buffer, from the buffer to the file, and to write the buffer onto a new file. If the edited text cannot be written out due to a full disk, it may still be possible to copy the whole text to the save buffer and then write it to a different file on a different disk with CTRL-Q. It may also be possible to escape from the editor with CTRL-S and remove some files.

Some of the commands prompt for arguments (file names, search patterns, etc.). All commands that might result in loss of the file being edited prompt to ask for confirmation.

A key (command or ordinary character) can be repeated *n* times by typing *ESC n key* where *ESC* is the ‘‘escape’’ key.

Forward and backward searching requires a regular expression as the search pattern. Regular expressions follow the same rules as in the UNIX editor, *ed*. These rules can be stated as:

1. Any displayable character matches itself.
2. . (period) matches any character except line feed.
3. ^ (circumflex) matches the start of the line.
4. $ (dollar sign) matches the end of the line.
5. \c matches the character *c* (including period, circumflex, etc).

6. [*string*] matches any of the characters in the string.
7. [^string] matches any of the characters except those in the string.
8. [*x*–*y*] matches any characters between *x* and *y* (e.g., [*a*–*z*]).
9. Pattern∗ matches any number of occurrences of *pattern*.

Some examples of regular expressions are:

The boy	matches the string ‘‘The boy’’
^$	matches any empty line.
^.$	matches any line containing exactly 1 character
^A.*\.$	matches any line starting with an *A*, ending with a period.
^[A–Z]*$	matches any line containing only capital letters (or empty).
[A–Z0–9]	matches any line containing either a capital letter or a digit.
.*X	matches any line ending in ‘‘X’’
A.*B	matches any line containing an ‘‘A’’ and then a ‘‘B’’

Control characters cannot be entered into a file simply by typing them because all of them are editor commands. To enter a control character, depress the ALT key, and then while holding it down, hit the ESC key. Release both ALT and ESC and type the control character. Control characters are displayed in reverse video.

The *mined* commands are as follows.

CURSOR MOTION

arrows	Move the cursor in the indicated direction
CTRL-A	Move cursor to start of current line
CTRL-Z	Move cursor to end of current line
CTRL-^	Move cursor to top of screen
CTRL-_	Move cursor to end of screen
CTRL-F	Move cursor forward to start of next word
CTRL-B	Move cursor backward to start of previous word

SCREEN MOTION

Home key	Move to first character of the file
End key	Move to last character of the file
PgUp key	Scroll window up 23 lines (closer to start of the file)
PgDn key	Scroll window down 23 lines (closer to end of the file)
CTRL-U	Scroll window up 1 line
CTRL-D	Scroll window down 1 line

MODIFYING TEXT

Del key	Delete the character under the cursor
Backspace	Delete the character to left of the cursor

CTRL-N Delete the next word
CTRL-P Delete the previous word
CTRL-T Delete tail of line (all characters from cursor to end of line)
CTRL-O Open up the line (insert line feed and back up)
CTRL-G Get and insert a file at the cursor position

BUFFER OPERATIONS

CTRL-@ Set mark at current position for use with CTRL-C and CTRL-K
CTRL-C Copy the text between the mark and the cursor into the buffer
CTRL-K Delete text between mark and cursor; also copy it to the buffer
CTRL-Y Yank contents of the buffer out and insert it at the cursor
CTRL-Q Write the contents of the buffer onto a file

MISCELLANEOUS

numeric + Search forward (prompts for regular expression)
numeric – Search backward (prompts for regular expression)
numeric 5 Display the file status
CTRL-] Go to specific line
CTRL-R Global replace *pattern* with *string* (from cursor to end)
CTRL-L Line replace *pattern* with *string*
CTRL-W Write the edited file back to the disk
CTRL-X Exit the editor
CTRL-S Fork off a shell (use CTRL-D to get back to the editor)
CTRL- Abort whatever the editor was doing and wait for command
CTRL-E Erase screen and redraw it
CTRL-V Visit (edit) a new file

The key bindings on the Atari ST and Amiga are slightly different. The table below summarizes the *mined* commands with the corresponding ST keys, and the PC keys if they differ.

CURSOR MOTION	**ST key**	**PC key**
up,down,left,right	arrows	
start of line	CTRL-A	
end of line	CTRL-Z	
top of screen	CTRL-^	
end of screen	CTRL-_	
next word	CTRL-F	
previous word	CTRL-B	

SCREEN MOTION	ST key	PC key
first char of file	Home	
last char of file	F6	End
scroll window up	F4	PgUp
scroll window down	F3	PgDn
scroll line up	CTRL-U	
scroll line down	CTRL-D	

MODIFYING TEXT	ST key	PC key
delete this char	Delete	
delete previous char	Backspace	
delete next word	CTRL-N	
delete previous word	CTRL-P	
delete tail of line	CTRL-T	
open up line	CTRL-O	
get file at cursor	CTRL-G	

MISCELLANEOUS	ST key	PC key
search forward	F1	numeric +
search backward	F2	numeric –
file status	F5	numeric 5
repeat	Esc	
goto line	CTRL-]	
global replace	CTRL-R	
line replace	CTRL-L	
write file	CTRL-W	
exit	CTRL-X	
fork shell	CTRL-S	
abort	CTRL-\	
redraw	CTRL-E	
new file	CTRL-V	
escape next char	F8	ALT-ESC

BUFFER OPERATIONS	ST key	PC key
set mark	F7	CTRL-@
copy to buffer	CTRL-C	
delete to buffer	CTRL-K	
insert buffer	CTRL-Y	
write buffer to file	CTRL-Q	

Command: **mkdir – make a directory**
Syntax: **mkdir** *directory ...*
Flags: (none)
Examples: mkdir dir # Create *dir* in the current directory
mkdir /user/ast/dir # Create the specified directory

The specified directory or directories are created and initialized.

Command: **mkfs – make a file system**
Syntax: **mkfs** [**–Ldot**] *special prototype*
Flags: **–L** Make a listing on standard output
–d Use mod time of *mkfs* binary for all files
–o Use a drive other than 0 or 1 (safety precaution)
–t Do not test if file system fits on the medium
Examples: mkfs /dev/fd1 proto # Make a file system on */dev/fd1*
mkfs /dev/fd1 360 # Make empty 360 block file system

Mkfs builds a file system and copies specified files to it. The prototype file tells which directories and files to copy to it. If the prototype file cannot be opened, and its name is just a string of digits, an empty file system will be made with the specified number of blocks. A sample prototype file follows. The text following the # sign in the example below is comment. In real prototype files, comments are not allowed.

```
boot                                    # boot block file (ignored)
360 63                                  # blocks and i-nodes
d--755 1 1                              # root directory
  bin   d--755  2 1                     # bin dir: mode (755), uid (2), gid (1)
        sh      ---755 2 1 /user/bin/shell   # shell has mode rwxr-xr-x
        mv    -u-755 2 1 /user/bin/mv       # u = SETUID bit
        login -ug755 2 1 /user/bin/login    # SETUID and SETGID
  $                                     # end of /bin
  dev   d--755 2 1                      # special files: tty (char), fd0 (block)
        tty     c--777 2 1 4 0          # uid=2, gid=1, major=4, minor=0
        fd0     b--644 2 1 2 0 360      # uid, gid, major, minor, blocks
  $                                     # end of /dev
  user  d--755 12 1                     # user dir: mode (755), uid (12), gid (1)
        ast     d--755 12 1             # /user/ast
        $                               # /user/ast is empty
  $                                     # end of /user
$                                       # end of root directory
```

The first entry on each line (except the first 3 and the $ lines, which terminate

directories) is the name the file or directory will get on the new file system. Next comes its mode, with the first character being **–dbc** for regular files, directories, block special files and character special files, respectively. The next two characters are used to specify the SETUID and SETGID bits, as shown above. The last three characters of the mode are the *rwx* protection bits.

Following the mode are the uid and gid. For special files, the major and minor devices are needed. The size in blocks must also be specified for block special files (the MINIX block size is 1K; this can only be changed by changing *BLOCK_SIZE* and then recompiling the operating system).

Command: mount – mount a file system
Syntax: **/etc/mount** *special file* [**–r**]
Flags: **–r** File system is mounted read-only
Example: /etc/mount /dev/fd1 /user # Mount diskette 1 on */user*

The file system contained on the special file is mounted on *file*. In the example above, the root directory of the file system in drive 1 can be accessed as */user* after the mount. When the file system is no longer needed, it must be unmounted before being removed from the drive.

Command: pwd – print working directory
Syntax: **pwd**
Flags: (none)
Example: pwd # Print the name of the working directory

The full path name of the current working directory is printed.

Command: rev – reverse the characters on each line of a file
Syntax: **rev** [*file*] ...
Flags: (none)
Example: rev file # Reverse each line

Each file is copied to standard output with all the characters of each line reversed, last one first and first one last.

Command: rm – remove a file
Syntax: **rm** [**–fir**] *name* ...
Flags: **–f** Forced remove: no questions asked

–i Interactive remove: ask before removing
–r Remove directories too

Examples:
```
rm file                 # Remove file
rm –i *.c               # Remove .c files, asking about each
```

Rm removes one or more files. If a file has no write permission, *rm* asks for permission (type ‘‘y’’ or ‘‘n’’) unless **–f** is specified. If the file is a directory, it will be recursively descended and removed if and only if the **–r** flag is present.

Command: rmdir – remove a directory
Syntax: **rmdir** *directory ...*
Flags: (none)

Examples:
```
rmdir /user/ast/foobar  # Remove directory foobar
rmdir /user/ast/f*      # Remove 0 or more directories
```

The specified directories are removed. Ordinary files are not removed. The directories must be empty.

Command: sh – shell
Syntax: **sh** [*file*]
Flags: (none)

Example:
```
sh < script             # Run a shell script
```

Sh is the shell. It permits redirection of input and output, pipes, magic characters, background processes, shell scripts and most of the other features of the V7 (Bourne) shell. A few of the more common commands are listed below:

```
date                    # Regular command
sort <file              # Redirect stdin
sort <file1  >file2     # Redirect stdin and stdout
cc file.c  2>error      # Redirect stderr
a.out >f  2>&1          # Combine standard output and standard error
sort <file1  >>file2    # Append output to file2
sort <file1  >file2 &   # Background job
(ls –l; a.out) &        # Run two background commands sequentially
sort <file | wc         # Two-process pipeline
sort <f | uniq | wc     # Three-process pipeline
ls –l *.c               # List all files ending in .c
ls –l [a-c]*            # List all files beginning with a, b, or c
ls –l ?                 # List all one-character file names
ls \?                   # List the file whose name is question mark
ls ’???’                # List the file whose name is three question marks
```

```
v=/usr/ast                  # Set shell variable v
ls –l $v                    # Use shell variable v
PS1='Hi! '                  # Change the primary prompt to Hi!
PS2='More: '                # Change the secondary prompt to More:
ls –l $HOME                 # List the home directory
echo $PATH                  # Echo the search path
if ... then ... else ... fi # If statement
for ... do ... done         # Iterate over argument list
while ... do ... done       # Repeat while condition holds
case ... in ... esac        # Select clause based on condition
echo $?                     # Echo exit status of previous command
echo $$                     # Echo shell's pid
echo $#                     # Echo number of parameters (shell script)
echo $2                     # Echo second parameter (shell script)
echo $*                     # Echo all parameters (shell script)
```

Command: **sleep – suspend execution for a given number of seconds**
Syntax: **sleep** *seconds*
Flags: (none)
Example:
```
sleep 10                    # Suspend execution for 10 sec.
```
The caller is suspended for the indicated number of seconds.

Command: **sort – sort a file of ASCII lines**
Syntax: **sort** [**–bcdfimnru**] [**–t***c*] [**–o** *name*] [**+***pos1*] [**–***pos2*] *file ...*
Flags:
–b Skip leading blanks when making comparisons
–c Check to see if a file is sorted
–d Dictionary order: ignore punctuation
–f Fold upper case onto lower case
–i Ignore nonASCII characters
–m Merge presorted files
–n Numeric sort order
–o Next argument is output file
–r Reverse the sort order
–t Following character is field separator
–u Unique mode (delete duplicate lines)

Examples:
```
sort –nr file               # Sort keys numerically, reversed
sort +2 –4 file             # Sort using fields 2 and 3 as key
sort +2 –t: –o out          # Field separator is :
sort +.3 –.6                # Characters 3 through 5 form the key
```

Sort sorts one or more files. If no files are specified, *stdin* is sorted. Output is written on standard output, unless **–o** is specified. The options **+***pos1* **–***pos2* use only fields *pos1* up to but not including *pos2* as the sort key, where a field is a string of characters delimited by spaces and tabs, unless a different field delimiter is specified with **–t**. Both *pos1* and *pos2* have the form *m.n* where *m* tells the number of fields and *n* tells the number of characters. Either *m* or *n* may be omitted.

Command: sync – flush the cache to disk
Syntax: **sync**
Flags: (none)
Example: sync # Write out all modified cache blocks

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.

Command: tail – print the last few lines of a file
Syntax: **tail** [**–cl**] [*–n*] [*file*] ...
Flags: **–c** The count refers to characters
–l The count refers to lines
–n How many characters or lines to print
Examples: tail –6 # Print last 6 lines of *stdin*
tail –1 file1 file2 # Print last line of two files

The last few lines of one or more files are printed. The default count is 10 lines. The default file is *stdin*.

Command: tee – divert *stdin* **to a file**
Syntax: **tee** [**–ai**] *file ...*
Flags: **–a** Append to the files, rather than overwriting
–i Ignore interrupts
Examples: cat file1 file2 | tee x # Save and display two files
pr file | tee x | lpr # Save the output of *pr* on *x*

Tee copies *stdin* to standard output. It also makes copies on all the files listed as arguments.

Command: **time – report how long a command takes**
Syntax: **time** *command*
Flags: (none)
Examples: time a.out # Report how long *a.out* takes
time ls –l *.c # Report how long *ls* takes

The command is executed and the real time, user time, and system time (in hours, minutes, and seconds) are printed. Shell scripts cannot be timed.

Command: **true – exit with the value true**
Syntax: **true**
Flags: (none)
Example:
```
while true                # List the directory until DEL is hit
do ls –l
done
```

This command returns the value *true*. It is used for shell programming. The program is in reality not a program at all. It is the null file.

Command: **umount – unmount a mounted file system**
Syntax: **/etc/umount** *special*
Flags: (none)
Example: /etc/umount /dev/fd1 # Unmount diskette 1

A mounted file system is unmounted after the cache has been flushed to disk. A diskette should never be removed while it is mounted. If this happens, and is discovered before another diskette is inserted, the original one can be replaced without harm. Attempts to unmount a file system holding working directories or open files will be rejected with a ‘‘device busy’’ message.

3

DESCRIPTION OF FULL MINIX 1.5

This chapter gives a summary of what the full MINIX distribution contains. The first section reproduces the table of contents of the manual. The second section lists and briefly describes most of the programs provided with MINIX. The third section names most of the library routines that are standard with MINIX.

3.1. MINIX REFERENCE MANUAL TABLE OF CONTENTS

The Table of Contents of the MINIX Reference Manual is listed below. The manual is 680 pages long, which includes a cross-referenced listing of the of the operating system source code (in C).

3.2. PARTIAL LIST OF PROGRAMS SUPPLIED WITH MINIX

Below is a list of programs that are supplied with MINIX. Those lines that have [IBM], [68000], or another machine type only apply to the machine type specified in the square brackets. The complete source code of all these programs, except for *elle* and the C compiler are part of the standard MINIX package.

animals – twenty-questions type guessing game about animals
anm – print name list [68000]
aoutdump – display the contents of an object file [68000]
ar – archiver
as – MC68000 assembler [68000]
ascii – strip all the pure ASCII lines from a file
asize – report the size of an object file [68000]
asld – assembler-loader [IBM]
ast – add symbol table to executable file [IBM]
astrip – remove symbols [68000]
at – execute commands at a later time
backup – backup files
badblocks – put a list of bad blocks in a file
banner – print a banner
basename – strip off file prefixes and suffixes
bawk – pattern matching language
btoa – binary to ascii conversion
cal – print a calendar
cat – concatenate files and write them to *stdout*
cc – C compiler
cdiff – context diff
cgrep – grep and display context
chgrp – change group
chmem – change memory allocation
chmod – change access mode for files
chown – change owner
clr – clear the screen
cmp – compare two files
comm – print lines common to two sorted files
compress – compress a file using modified Lempel-Ziv coding
cp – copy file

cpdir – copy a directory and its subdirectories
crc – print the checksum of the file data
cron – clock daemon
ctags – build a tags file
cut – select out columns of a file
date – print or set the date and time
dd – disk dumper
de – disk editor
df – report on free disk space and i-nodes
dhrystone – integer benchmark
diff – print differences between two files
dis88 – disassembler [IBM]
diskcheck – check a disk for bad sectors
diskcopy – copy a disk with only one drive [AMIGA]
diskrtc – set date from a disk controller real time clock [ATARI]
diskset – set real time clock on disk controller [ATARI]
dosdir – list an MS-DOS directory [IBM]
dosread – read a file from an MS-DOS diskette [IBM]
doswrite – write a file onto an MS-DOS diskette [IBM]
du – print disk usage
echo – print the arguments
ed – editor
eject – eject a diskette from a drive [MACINTOSH]
elle – ELLE Looks Like Emacs
elvis – clone of the Berkeley *vi* editor
ex – Berkeley line editor
expand – convert tabs to spaces
expr – evaluate experession
factor – factor an integer less than 2**31
fdisk – partition a hard disk [IBM]
fgrep – fast grep
file – make a guess as to a file's type based on contents
find – find files meeting a given condition
fix – generate new file from old one and diff listing
fold – fold long lines
format – format a diskette [ATARI]
fortune – print a fortune
from – input half of a connection [IBM]
fsck – perform file system consistency check
gather – gather up the files in a directory for transmission
getlf – wait until a line has been typed
getty – get terminal line parameters for login
grep – search a file for lines containing a given pattern

gres – grep and substitute
head – print the first few lines of a file
hdclose – close hard disk partition [MACINTOSH]
hdopen – set correspondence of a HD partition [MACINTOSH]
ic – integer calculator
id – print the uid and gid
ifdef – remove #ifdefs from a file
indent – reformat the layout of a program
inodes – print i-node information
kermit – transfer a file using the kermit protocol
kill – send a signal to a process
last – display recent on-line session records
leave – warn when it is time to go home
libpack – pack an ASCII assembly code file [IBM]
libupack – convert a packed assembly code file to ASCII [IBM]
ln – create a link to a file
login – log into the computer
look – look up words in dictionary
lorder – compute the order for library modules [IBM]
lpr – copy a file to the line printer
ls – list the contents of a directory
m4 – macro processor
maccreate – create an empty macintosh file [MACINTOSH]
macfile – list, read and write Macintosh volumes [MACINTOSH]
macread – read a Macintosh file [MACINTOSH]
macwrite – write a Macintosh file [MACINTOSH]
mail – send and receive electronic mail
make – a program for maintaining large programs
man – display manual page
master – control the creation of shervers [IBM]
mdb – MINIX debugger [68000]
megartc – set date from real time clock [Mega ST]
mined – MINIX editor
minix – MINIX bootstrap [AMIGA]
mkdir – make a directory
mkfs – make a file system
mknod – create a special file
mkproto – create a MINIX prototype file
modem – switch the modem and getty state
more – pager
mount – mount a file system
mref – make listing and cross reference map of MINIX
mv – move or rename a file

nm – print name list
nroff – text formatter
od – octal dump
passwd – change a login password
paste – paste multiple files together
patch – patches up a file from the original and a diff
pr – print a file
prep – prepare a text file for statistical analysis
pretty – MINIX pretty printer
printenv – print out the current environment
printroot – print the name of the root device on standard output
ps – print process status
pwd – print working directory
rcp – remote copy [IBM]
readall – read a device quickly to check for bad blocks
readclock – read the real time clock [IBM PC/AT and AMIGA]
readfs – read a MINIX file system
recover – recover files that have been removed.
ref – look up a reference in a *tags* file
rev – reverse the characters on each line of a file
rm – remove a file
rmaker – a simple resource compiler [MACINTOSH]
rmdir – remove a directory
roff – text formatter
rsh – remote shell for networking [IBM]
rz – receive a file using the zmodem protocol
sed – stream editor
settype – set type and/or creator of a Mac file [MACINTOSH]
sh – shell
shar – shell archiver
sherver – shell server [IBM]
size – print text, data, and bss size of a program
sleep – suspend execution for a given number of seconds
sort – sort a file of ASCII lines
spell – print all words in a file not present in the dictionary
split – split a large file into several smaller files
strings – print all the strings in a binary file
strip – remove symbol table from executable file
stterm – turn system into a dumb terminal [68000]
stty – set terminal parameters
su – temporarily log in as superuser or another user
sum – compute the checksum and block count of a file
svc – shell version control system

sync – flush the cache to disk
sz – send a file using the zmodem protocol
tail – print the last few lines of a file
tar – tape archiver
tee – divert *stdin* to a file
term – turn PC into a dumb terminal [IBM]
termcap – print the current termcap entry
test – test for a condition
time – report how long a command takes
to – output half of a connection [IBM]
tos – list, read and write TOS file systems [ATARI]
touch – update a file's time of last modification
tr – translate character codes
transfer – read, write and format diskettes [AMIGA]
traverse – print directory tree under the named directory
treecmp – recursively list differences in two directory trees
true – exit with the value true
tset – set the $TERM variable
tsort – topological sort [IBM]
ttt – tic tac toe
tty – print the device name of this tty
umount – unmount a mounted file system
unexpand – convert spaces to tabs
uniq – delete consecutive identical lines in a file
unshar – Remove files from a shell archive
update – periodically write the buffer cache to disk
users – list the logged-in users
uud – decode a binary file encoded with uue
uue – encode a binary file to ASCII (e.g., for mailing)
vi – (see *elvis*)
virecover – recover from a crash
vol – split stdin into diskette-sized volumes
wc – count characters, words, and lines in a file
weidertc – set date from Weide real time clock [ATARI]
whatsnew – print a newly modified file, marking changes
whereis – examine system directories for a given file
which – examine $PATH to see which file will be executed
who – print list of currently logged in users
whoami – print current user name
width – force all the lines of a file to a given width
write – send a message to a logged-in user

3.3. PARTIAL LIST OF THE MINIX LIBRARY

Below is a listing of the three principal library directories, *ansi*, *posix*, and *other*. These lists are approximate, as there are some minor differences between the various versions. The sources for all these library routines are included in the MINIX package (as are the binaries, of course).

3.3.1. Ansi Directory

Makefile abort abs assert atoi atol bsearch ctime ctype errno exit fclose fflush fgetc fgets fopen fprintf fputc fputs fread freopen fseek ftell fwrite getenv gets malloc memchr memcmp memcpy memmove memset perror puts qsort rand scanf setbuf signal sincos sprintf strcat strchr strcmp strcoll strcpy strcspn strerror strlen strncat strncmp strncpy strpbrk strrchr strspn strstr strtok strtol strtoul strxfrm system time tmpnam ungetc vsprintf

3.3.2. Posix Directory

Makefile _exit access alarm chdir chmod chown close closedir creat ctermid cuserid dup dup2 exec execlp fcntl fdopen fork fpathconf fstat getcwd getegid geteuid getgid getgrent getlogin getpid getppid getpwent getuid isatty kill link lseek mkdir mkfifo open opendir pathconf pause pipe read readdir rename rewinddir rmdir setgid setuid sleep stat sysconf times ttyname umask unlink utime wait write

3.3.3. Other Directory

Makefile amoeba bcmp bcopy brk bzero call chroot cleanup crypt curses doprintf ffs getdents getopt getpass gtty index ioctl itoa lock lrand lsearch memccpy message mknod mktemp mount nlist popen printdat printk prints ptrace putenv regexp regsub rindex seekdir stb stderr stime stty swab sync syslib telldir termcap umount uniqport vectab

3.4. ADDITIONAL SOFTWARE

The MINIX package consists of boot diskettes, the binaries of nearly 200 system utilities, a C compiler, the sources of the complete operating system (in C), the sources of the utility programs, and the sources of the library. In addition, the package contains ANSI and POSIX compatible header files */usr/include/*.h*), and extensive test software. Furthermore, MINIX comes standard with networking software to connect multiple IBM machines on an Ethernet. This software allows file transfer, remote login, cross-machine pipes, and many other features. One particularly interesting aspect of it, is that it also works on a single machine, so you can develop

networking software alone. The binary programs so produced will run on a network of MINIX machines without modification or even recompilation.

A Pascal compiler is also available from third-party vendors.

Finally, there is a large and active MINIX community that operates on USENET. Over 16,000 people belong to this group, which is described in the MINIX Reference Manual.