

Unix

1. **Fundamentals**
2. **Working with files**
3. **Customizing the session**
4. **Editor: the GNU emacs**
5. **Compiling and running**

1

1 Fundamentals

UNIX is a multi-user, multi-tasking operating system.

UNIX is made up of the **kernel**, the heart of the operating system; the **file system**, a hierarchical directory method for organizing files on the disk; and the **shell**, the user interface through which you command the system.

Normally people share a UNIX computer system. Each user must have a UNIX account, which provides a necessary means of regulating use of the system. An account includes a username and password. The user-name is the way you identify yourself to the system; the password verifies your identity.

2

1.1 Prompt

When the computer is ready to listen to your directions, it displays a prompt. Commands are directions given to the computer by the user.

1.2 Commands

Many command names are abbreviations for English words. To give the computer a command, you type the command name and press **RETURN**.

3

1.3 Arguments and Options

Most UNIX commands allow you to attach one or more arguments. Arguments determine the way in which the command carries out its function. Some types of arguments are names of computer files or time limits. For example, the command **ls** tells the computer to list the names of files in your current working directory.

To find out whether the file name **mbox** is in your current working directory, you would type the argument **mbox** after the command **ls**:

```
ls mbox
```

if the file **mbox** exists, its name would be listed on the screen.

4

An option is an argument included on the same line as the command and preceded by a hyphen (-). It tells the computer to execute a particular variation of the command. For example:

```
ls -l
```

tells the computer to list the file names with additional information about each file.

WARNING!!

UNIX is **case sensitive**. Case sensitive means that UNIX differentiates between uppercase and lowercase letters.

5

1.4 Special Keys and Control Sequences

There are several special key and control sequences that are helpful when you use UNIX. When you type a control sequence, you hold down the CTRL key while you type a certain letter.

To type ^c, for example, you would hold down the CTRL key while you type c.

You want to:

Erase the last character you typed	DELETE or BACKSPACE
Erase the last line you typed	^u
Erase the last word you typed	^w
Stop or interrupt a program currently running	^c
Freeze the program's output scrolling on the screen, although the program may continue to run	^s
Unfreeze the output suspended by ^s	^q

6

2 Working with files

2.1 Structure

Computer files can be thought of as folders inside a file cabinet.

Each file is labeled with a name. Files are quite important allowing information storage for use at a later time.

UNIX files are organized in directories, similar to the file folders discussed above. Directories are files, but instead of containing data or text as other files do, the directory contains information about other files. Directories are organized like inverted trees.

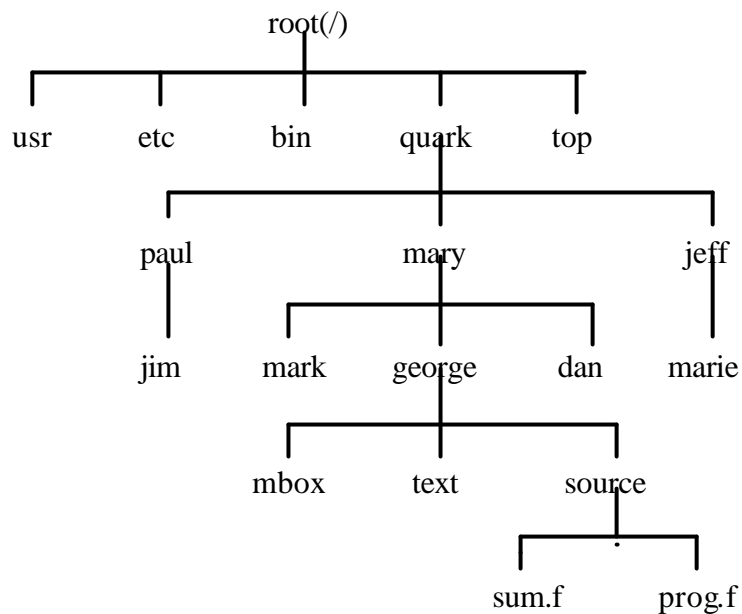
The root directory named / is at the top.

Branching out from the root directory are the rest of the directories.

Any directory can contain both directories and other files.

All directories together are referred to as a **file system**.

7



8

Example (see previous picture):

The / (slash) at the top of Figure 1 is the root directory.

Within this root directory the following files are shown:

usr, bin, etc, top, quark.

These files just happen to be directories, which contain their own files.

These directories-within-a-directory are called subdirectories.

Thus, usr and etc are subdirectories of /.

Every file on the system is contained in the root directory or in one of its subdirectories.

9

2.2 Absolute Pathnames

To identify a certain file in the file system, you can specify an absolute pathname by telling UNIX where the file is in relation to the root directory.

For example, to refer to the file sum.f above, use

/quark/mary/george/source/sum.f

The first / signifies the root directory. Each subsequent directory name is separated by an additional /.

10

2.3 Your Current Working Directory

Whenever you are using a UNIX system, you are in a specific directory. That directory is called the **current working directory**. Commands that operate on directories use the working directory, unless specified otherwise.

2.4 Relative Pathnames

Besides identifying a file by its absolute pathname, you can use a relative pathname. That is, you can tell UNIX where the file is in relation to the working directory.

Suppose the working directory is */quark*, as shown in the example above. To refer to the file *sum.f*, you could specify its relative path-name:

mary/george/source/sum.f.

11

Relative pathnames never begin with / because their origin is the working directory, not the root directory.

When you want to use the working directory as an argument for a command you can use the period (.) as a shorthand notation.

Suppose the present working directory is */quark/paul*; you could refer to it with *.* as an argument for the copy command *cp*:

cp /quark/mary/george/source/sum.f .

In this case, the file */quark/mary/george/source/sum.f* would be copied into the working directory */quark/paul* with the same name.

2.5 Your Home Directory

When you first login to a UNIX system, you are in the directory that holds your personal files, known as your home directory.

12

2.6 Handling Directories

pwd

print working directory: to find out what the working directory is.

cd *<directory name>*

to go to the directory *<directory name>* .

Use either the relative or absolute pathname.

cd

to go to the home directory.

cd ..

to go one directory up in the directory tree

mkdir *<directory name>*

to create the directory *<directory name>* .

13

2.7 Finding Out What Is in a Directory

The **ls** command is used for listing the files and subdirectories contained in a directory.

ls *<directory name>*

display the files in the directory *<directory name>*.

ls

display the files in the current working directory.

ls -a

display all the files, including the files whose names begin with a dot.

ls s*

display the files that begins with s (lower case !) in the current working directory. * is the “wild character” and indicates any string of characters.

ls <filename>

display the file <filename> if it exists in the current working directory.

ls -l

lists each file or directory in the working directory, along with its permissions (security information), creator, size in bytes, and date of last modification.

Example:

```
$ls -l
```

```
-rw-r--r-- 1 bianchi babar 11350 Dec 8 22:45 SvtSimHitChannel.cc  
-rw-r--r-- 1 bianchi babar 2284  Mar 4 1998 SvtSimHitChannel.hh  
-rw-r--r-- 1 bianchi babar 12139 Dec 4 12:00 SvtSimHitLayer.cc  
-rw-r--r-- 1 bianchi babar 2687  Aug 4 1998 SvtSimHitLayer.hh
```

15

2.8 Handling Files

cp

copies files or directories. The first argument is the source file (i.e., the file you are copying).

The second argument is either the name you want for the new copied file or the destination directory. After the source file is copied, it remains intact in the directory where it was located.

Examples:

```
$ cp sum.f sample.bak
```

Copy the file sum.f. The name of the new copied file is sample.bak.

```
$ cp /quark/paul/.cshrc .
```

Copy the .cshrc file to the working directory.

The second period indicates the working directory.

16

mv

move a file from one directory to another or rename a file.

The first argument is the source file (i.e., the file you are moving or renaming). The second argument can be the destination directory (the directory where the file should be moved) or the new name of the file (use the absolute or relative pathname). If a directory is given as the second argument, the file is moved into that directory with the same filename. The source file is deleted from its directory after the procedure is finished.

Examples:

```
$ mv sum.f test.case
```

Gives the file sum.f the new name test.case (i.e., renames the file).

```
$ mv test.case /usr/smith/cases/test.bak
```

Moves the file test.case to the file test.bak in the subdirectory 17
/usr/smith/cases.

When you use a program, such as a text editor, you **create a file** to save your work. That way, the file can be opened later to allow for editing, printing, or adding to its contents.

rm is used to remove files .

WARNING!! You cannot recover a file that has been removed by this method!

Examples:

```
$ rm test.case
```

Removes the file test.case permanently.

```
$ rm -r testdir
```

Removes the directory testdir and all its files permanently

```
$ rm -r *
```

Removes all directories and all their files permanently.

NEVER, NEVER do it !

18

more <filename>

display a screenful of file <filename> at a time. To display the next screenful, press the space bar. The more command scrolls the text as it displays the next screenful.

head -n <filename>

displays the first n lines of file <filename>

tail -n <filename>

displays the last n lines of file <filename>

grep <string> <filename(s)>

searches for text strings or consecutive words in a file. You supply the string to search for and the file(s) to search in. If the string you are looking for contains spaces, you must put the string in single or double quotes. With the grep command, any lines in the file containing the string display on the screen.

19

diff <file1> <file2>

displays the actual lines in each file that differ.

2.9 File Access Permissions

With the UNIX security system, you can determine who can see, alter, and use your files and directories. Those who may access a particular file or directory are divided into three classes :

Class of User Definition

user The user who owns the file/directory.

group The group of users who share access of a file. (Every user belongs to one or more groups. To see what groups you are in, type groups at the prompt.) Each file and directory "belongs to" one user and one group.

others All other users.

20

Each file or directory has a set of permissions that control access to it. For each class of users above, there are three permissions that may be granted to that class for each file or directory.

Permission	File	Directory
read (r)	Allows class to read the file.	Allows class to list names of files and subdirectories in the directory.
write (w)	Allows class to modify a file (e.g., to append to it or to truncate it).	Allows class to create and remove files and subdirectories from the directory.
execute (x)	Allows class to execute a file.	Allows class to access files in the directory if they know their names.

21

The **ls -l** command shows the permissions of a particular file:

```
-----
$ls -l
drwxr-xr-x 3 ilse sf 1536 Sep 24 10:15 Telecom/
-rw-r--r-- 1 ilse sf 830 Aug 23 12:39 Unix-guide-questions
-rw-rw-r-- 1 ilse sf 353280 Sep 27 10:34 Unix-guide-
slac.block.frame
-rw-rw-rw- 1 ilse sf 353280 Sep 29 11:00 Unix-guide-
slac.block.frame.auto
-----
```

The first character indicates whether this file is a directory (d) or just a regular file (-). The next three groups, consisting of three characters each (nine characters total), are the permissions granted for the user, group, and others classes, respectively. A permission is granted if the letter is present and not granted if the (-) is there instead. Thus, in the example above, the file Unix-guide-questions is a regular file (-) that allows the user reading and writing privileges (rw-), but limits the group and others to only reading the file (r-).

22

The **chmod** command allows you to change the permissions associated with a file or directory that you own.

```
$ ls -l
total 12
-rw-r--r--  1  bobcook sf 2847  Sep 6 11:42  groupfile
drwxr-xr-x  2  bobcook sf   32  Sep 6 11:41  privatedir/
-rw-r--r--  1  bobcook sf  664  Sep 6 11:42  prog
$ chmod a+x prog          add execution permission for 'all'
$ chmod g+w groupfile    add write permission to 'group'
$ chmod ga-xr privatedir remove write and execution
                          permission to 'group' and 'all'

$ ls -l
total 12
-rw-rw-r--  1  bobcook sf 2847  Sep 6 11:42  groupfile
drwx-----  2  bobcook sf   32  Sep 6 11:41  privatedir/
-rwxr-xr-x  1  bobcook sf  664  Sep 6 11:42  prog*
```

23

3.0 Customizing the session

After you log in, the shell program, or shell, is activated. A shell is a user interface program through which you can communicate with UNIX. The shell displays the prompt \$, which is changeable, as a signal that it is ready to receive your UNIX commands. It interprets and executes the UNIX command that you type, and when done, gives you the next prompt.

UNIX allows you to customize your interaction with the shell program in many ways. You can create abbreviations for files or directories or lengthy commands; determine the directories the shell uses to search for the programs and commands to run; create the prompt the shell uses when it interacts with you.

24

3.1 Shell and environment variables

Shell and environment variables store values that the shell uses in executing your commands. You can create your own shell variables and assign them values with the **set** command, and you can insert the variable value in a command line by prefixing the variable name with \$.

\$ set docdir=/usr/local/doc	Sets the variable docdir to /usr/local/doc. That is, docdir becomes an abbreviation for /usr/local/doc.
\$ echo \$docdir	Shows the value assigned /usr/local/doc to docdir.
\$ cd \$docdir	The current working directory changes to /usr/local/doc.
\$ unset docdir	Removes the shell variable docdir. Now, you can no longer use \$docdir to refer to the directory /usr/local/doc.

25

Without argument, **set** displays shell variables and their values.

One of the most important shell variables is the **path** shell variable. The value for the path shell variable is a list of directories. This list of directories is called your **search path**.

UNIX searches these directories when it looks for a program or command specified on the command line. If a program is in your search path, you can just type the name of the program; you don't have to type its absolute or relative pathname.

26

Suppose your search path is:

```
/usr/<userid>/bin /usr/local/bin /usr/afsws/bin /usr/ucb /bin /usr/bin
```

where "/usr/<userid>/bin" identifies your personal bin directory.

Whenever you type a program name or command, the shell looks first in your bin directory, second in the directory /usr/local/bin, third in the directory /usr/afsws/bin, and so on.

It keeps looking until it either finds the program or command with that name or finishes looking through all the directories in your search path.

set path=(\$path /joe/bin)

adds the directory /joe/bin to the end of your search path.

set path=(/joe/bin \$path)

adds the directory /joe/bin to the beginning of your search path.²⁷

If you find that you frequently use a program or command not included in your default search path, you can edit or add a "**set path=**" command in your **.cshrc** file.

After you add or modify command lines in the **.cshrc** file, the changes will be in effect for all subsequently started shells. However, neither your currently running shell, nor any of its ancestors will automatically see the changes. To invoke the commands added to your **.cshrc** file in your current shell, use the **source .cshrc** command. Environment variables are similar to shell variables, but other programs besides the shell use their values. They are defined with the **setenv** command.

Normally, UNIX commands needing user input require you to type the input at the keyboard.

Most UNIX commands show you the output of the command by displaying it on your terminal screen. The keyboard is the standard input and the terminal is the standard output. But often you might like to store the lengthy output of a command in a file or cause the input for a command to come from a specific file. The shell provides a means for redirecting where a command sends its output or receives its input.

The characters `<`, `>` and `>>` are used to redirect input and output.

```
-----  
$ who  
raines  ttyp0  Aug 20 09:08  (CGIBM1.SLAC.Stan)  
ohnishi ttyp1  Aug 20 09:17  (134.79.128.89:0.)  
meb     ttyp3  Aug 20 08:15  (sisyphus:0.0)  
-----
```

who shows who is logged on to the system

29

```
-----  
$ who > who.doc  
-----
```

Places the output of the **who** command into the file *who.doc*. If the file already exists, its contents are replaced by the output of the **who** command. If the file does not already exist, is created.

```
-----  
$ who >> info.doc  
-----
```

Append the output of the **who** command to the end of file *info.doc*. The contents of this file remain intact with the output of the **who** command attached at the end.

30

\$ cat sec1.doc sec2.doc > unixbook

Strings together sec1.doc and sec2.doc. Redirects the output of the **cat** command to the file unixbook.

The UNIX shell provides a means of connecting the output and input of two commands via a mechanism called a pipe. The pipe character | is placed between two commands when the first command's output should be the input of the second.

\$ ls -l /bin | more

Lists the files in directory /bin (the command **ls -l /bin**), one screenful at a time (the command **more**).

31

4.0 The GNU emacs

emacs is a UNIX editor that allows creation and modification of files.

emacs <myfile>





open the file <myfile> for editing. If it does not exist, emacs creates it.

to exit emacs, type:

^x ^c (lower case !)

32

Basic emacs commands

<code>^h ^h ^h</code>	menu of help option
<code>^h t</code>	tutorial
	move forward one character
	move backward one character
	move one line up
	move one line down
<code>^e</code>	move to end of line
<code>^a</code>	move to beginning of line
<code><esc> ></code>	move to end of buffer
<code><esc> <</code>	move to beginning of buffer

33

<code></code>	delete previous character
<code>^d</code>	delete character under cursor
<code>^k</code>	delete from cursor to end of line
<code>^<space></code>	mark beginning or end of a region
<code>^w</code>	delete a region
<code>^y</code>	restore what has been deleted
<code>^u</code>	undo last command
<code>^s 'string'</code>	move cursor to 'string'
<code>^x ^s</code>	save to file

34

5.0 Compiling and Running

To create an executable program, compile a source file containing a main program:

```
$ g++ mainprg.cc (GNU, free)
```

```
$ cxx mainprg.cc (COMPAQ)
```

```
$ xlc mainprg.cc (IBM)
```

```
$ CC mainprg.cc (Sun)
```

the c++ compiler creates an executable named a.out in the current working directory.

To execute the program, type:

```
$ a.out
```

If the source is divided among many files:

```
$ g++ mainprg.cc file1.cc file2.cc...filen.cc
```

35

5.1 Options

```
$ g++ -o <filename> mainprg.cc
```

the executable is named <filename> instead of a.out

```
$ g++ -c file.cc
```

the link phase is suppressed, the compiler generates an object file named file.o. In this example file.cc does not need to contain a main .

file.o can be linked later on:

```
$ g++ mainprg.cc file.o
```

36

5.2 Library

Libraries are collection of precompiled code.

To create a library named `libname.a` that contains the files `file1.cc`, `file2.cc` `file3.cc`:

```
$ g++ -c file1.cc  
$ g++ -c file2.cc  
$ g++ -c file3.cc  
$ ar -r libname.a file1.o file2.o file3.o
```

To display the content of a library:

```
$ ar -t libname.a
```

37