Basic Unix Commands

Contents

1 Intro

Unix is a multi-tasking, multi-user operating system. This means that on any given computer, it appears as if many things are happening at once and that there may be more than one person logged into the computer at once. Regardless of which machine you log into in the Institute, you will have access to your user files and the basic commands will be identical.

The Unix file system is hierarchical. Every file is stored in a directory. A directory can contain many files or none at all, and may also contain other directories called subdirectories. Unix has three types of files:

Normal Files These are data files which might contain text, source code, executable files, *etc.*

Special Files These represent physical devices such as terminals and disk drives. The "device drivers" will translate any references to such files into the hardware instructions needed to carry out the tasks.

Directories These contain "pointers" to normal files, special files and other directories.

File names can be as long as you like, unlike in MS-DOS.

The parent directory of all other directories is called the "root" directory and is denoted by /. Every file on the system can be accessed by tracing a path from this root directory. See the section on pathnames.

Each user has a "home" directory: this is where you will be located when you login. You are then free to traverse the directory structure of the Institute and to add to and change the part of the structure which you own.

Unix commands have the following format:

`command [`*options*`...] [`*arguments*`...]`

Where command is the command name, *options* refer to optional command modifiers (usually prefixed by a "-") and *arguments* are the optional or required command parameters (often file names). Spaces or tabs are required between commands and options and between options and each argument. Commands must be typed in the proper case (nearly always lower case).

Although each option to a command may be prefixed by a "-", with almost all of the standard commands you may put the options together. For example

```
command -a -b -c -d
```

can often be abbreviated as

```
command -abcd
```

Again, this is almost always the case - the most notable exceptions are in the GNU commands `enscript` and `gcc`.

The following are a series of Unix commands which will help you use the computers. They are given in their most basic form and more information will be available from their on-line manual pages (accessed through the man command described below).

Each command will be given in a generic form, perhaps with an example of an actual usage. In the examples, the line `alph%` will indicate the prompt you see on your screen (which you would not type in if actually using the command). Note that your prompt may be different.

2 man - Accessing On-Line Manual Pages

The man command looks up the manual page for a command.

The format of `man` is

`man [-k]` *name...*

-k Prints a list of all manual pages containing the keyword *name*.

Use the *-k* option if you do not know the name of the command or program.

3 pwd - Print the Working Directory

The pwd command prints the full pathname of your current working directory. The format of `pwd` is

`pwd`

For example,

```
alph% pwd
/home/stlawrence/user/newuser
alph%
```

The meaning of a full pathname is described in the section on "Filenames and Pathnames".

4 cd - Changing Directory

The cd command changes the current working directory to the directory specified. The format of `cd` is

`cd [`*directory*`]`

If you do not specify *directory*, cd changes to your home directory.

For example,

```
alph% cd /home/stlawrence/user/newuser
alph% pwd
/home/stlawrence/user/newuser
alph% cd Mail
alph% pwd
/home/stlawrence/user/newuser/Mail
alph% cd
alph% pwd
/home/stlawrence/user/newuser
alph%
```

5 ls - Listing the Contents of Directories

The ls command lists the contents of one or more specified directories or lists information about one or more specified files. Normally ls does not list filenames that begin with a dot ("."). The format of `ls` is

`ls [`*-aFl*`] [`*df(1)*`] [`*df(2)*`] [...] [`*df(n)*`]`

where *df(1..n)* is a directory name or a file name.

-a Lists all files, including those that begin with a ".".

-F Marks directories with a / and executable files with a *.

-l Produces a longer, more informative listing (see the section on chmod for more information).

For example,

```
alph% ls
Mail a.out year1 zeta.f zeta.o
alph% ls -F
Mail/ a.out* year1 zeta.f zeta.o
alph%
```

6 mkdir - Making Directories

The mkdir command makes directories with specified names. The format of the `mkdir` command is

`mkdir` *directory(1) directory(2)... directory(n)*

For example,

```
alph% ls -F
Mail/ prog/ zeta.f
alph% mkdir thesis zeta
alph% ls -F
Mail/ prog/ thesis/ zeta/ zeta.f
alph%
```

7 rmdir - Removing Directories

The rmdir command removes *empty* directories with specified names. The format of the `rmdir` command is

`rmdir` *directory(1) directory(2)... directory(n)*

The rmdir command will not remove a directory with files in it - for this use the rm -r command, described later in this section, but **be careful!**.

For example,

```
alph% ls -F
Mail/ prog/ thesis/ zeta/ zeta.f
alph% rmdir zeta
alph% ls -F
Mail/ prog/ thesis/ zeta.f
alph%
```

8 cp - Copying a File

The cp command makes a copy of a file or copies multiple files into a directory. The format of the `cp` command is

`cp` *source-file destination-file*

or

`cp` *source-file(1) source-file(2)... source-file(n) destination-directory*

The first form makes a copy of the file *source-file* called *destination-file* and the second copies series of files *source-file(1..n)* into directory *destination-directory*.

```
alph% ls -F
alpha.f beta.c mydir/ zeta.f
alph% cp zeta.f zeta.f.old
alph% ls -F
alpha.f beta.c mydir/ zeta.f zeta.f.old
alph% cp alpha.f zeta.f mydir
alph% ls -F mydir
```

```
alpha.f zeta.f
alph%
```

9 cat - Printing Files Onto the Screen

The cat command prints out the contents of a series of files one after the other. The format of the `cat` command is

`cat` *filename(1) filename(2)... filename(n)*

For example, to read the "message of the day",

```
alph% cat /etc/motd
```

10 more - Printing Files One Screen at a Time

The more command prints out the contents of named files, one screen full at a time. The format of the `more` command is

`more` *filename(1) filename(2)... filename(n)*

To quit more, press the **q** key, to move one line at a time press the **RETURN** key or the **SPACE** bar to move one screen full at a time.

11 mv - Moving and Renaming Files

The mv command changes the name or location of a file or directory. The formats of the `mv` command are

`mv` *oldfile newfile*

`mv` *file(1) file(2)... file(n) directory*

`mv` *olddir newdir*

For example,

```
alph% ls -F
Mail/ prog/ zeta/ zeta.f zeta.f.old
alph% mv zeta.f.old zeta-old.f
alph% ls -F
Mail/ prog/ zeta/ zeta.f zeta-old.f
alph% mv zeta.* zeta
alph% ls -F
Mail/ prog/ zeta/
alph% ls -F zeta
zeta:
zeta.f zeta-old.f
alph% mv prog program
alph% ls -F
Mail/ program/ zeta/
alph% mv zeta program
alph% ls -F
Mail/ program/
alph% ls -F program
program:
zeta/
alph%
```

12 rm - Removing Files and Directories

The rm command removes files and directories. **Caution:** There is no way to reverse this process (although see the section on backup). The format of the `rm` command is

`rm [-i] [-r]` *fd(1) fd(2)... fd(n)*

where fd(1..n) are files or directories.

-i Inquire before removing a file ("y" to delete, anything else to not delete).

-r Recursively remove a directory and all its contents and subdirectories (**Use with extreme care**).

For example,

```
alph% ls
Mail/ prog.f
alph% rm prog.f
alph% ls
Mail/
alph%
```

13 chmod - Changing Access Permissions

The chmod command changes the "permissions" on a file or directory. It gives or removes access for another user or group of users to read, change or run one of the files owned by you.

Users on the system fall into three categories:

user You.

group Anyone in the same class as yourself, such as *pg_1999*, *staff* or *postdoc*.

other Anyone who uses the Institute Computers (sometimes called **world**).

The format of the `chmod` command is

`chmod` *ugo+-=rwx fd(1) fd(2)... fd(n)*

where fd(1..n) may be a file or directory.

where

ugo Specify **u** (user), **g** (group) or **o** (other).

+-= Specify + (add), **-** (subtract) or = (set).

rwx Specify **r** (read), **w** (write) or **x** (execute).

For files, read permission means you can read the contents of a file, write permission means you can change the file and execute permission means you can execute the file (if it is executable or it is a shell script).

For directories, read permission means you can see what files are in the directory, write permission means that you can add to and remove files from the directory and execute means you can access files in that directory.

Note that to access a file you must have execute permission on all the directories above that on the file system (including the one in which it is resident). In addition, you must have the appropriate access permissions for that file.

The permissions on a file may be shown by using the command

```
alph% ls -lgF
...
-rw-r--r-- 1 newuser pg_1999 1451 Jan 18 11:02 phone
drwxr-xr-x 2 newuser pg_1999 512 Jan 22 12:37 thesis/
...
alph%
```

There are 10 fields at the start of the entry. The first letter refers to whether the entry is a file (-), directory (d), or something else (such as s or l). The next letters should be considered in groups of three.

The first group of three refers to the user. The second group of three refers to the members of the group "pg_1999". The last group of three refers to world.

Inside each group of three, the three entrys refer to read (r), write (w) and execute (x). A hyphen (-) indicates something not being set. In the example above all users on the system can read and change into

the directory called *thesis* and all users can read the file *phone*, and in addition the owner can write in both the *thesis* directory and can change the file *phone*.

If, for example, the material in the file *phone* was personal, it is possible to make sure no one else can read the file by typing

```
alph% chmod g-r o-r phone
```

and the new output of ls -agl would be

```
alph% ls -agl
...
-rw------- 1 newuser pg_1999 1451 Jan 18 11:02 phone
drwxr-xr-x 2 newuser pg_1999 512 Jan 22 12:37 thesis
...
alph%
```

in this case g-r refers to "group remove read" and o-r "others remove read". You could equally use

alph% chmod g+r

to allow anyone in the group "pg_1999" to read the file.

14 diff - Finding the Differences Between Two Files

The diff command compares the contents of two files. The format of `diff` is

`diff` *file1 file2*

For example,

```
alph% cat dataA
My travel plans are as follows:
Oxford - Heathrow by bus.
Heathrow - Paris by plane.
alph% cat dataB
Travel Plans:
Oxford - Heathrow by bus.
Heathrow - Paris by plane.
alph% diff dataA dataB
1c1
< My travel plans are as follows:
---
> Travel Plans:
```

The diff command compares files on a line-by-line basis. A < precedes lines from *file1* and a > precedes lines from *file2*.

15 grep - Searching for Strings in Files

The grep command scans a file for the occurrence of a word or string and prints out any line in which it is found. The format of `grep` is

`grep [-i] 'string'` *filename(1) filename(2)... filename(n)*

-i Ignore case in the search

The single right-quotes around *string* are necessary only if the string contains non-alphanumeric characters (such as **SPACE**, **&**, *etc.*).

For example,

```
alph% cat dataA
My travel plans are as follows:
Oxford - Heathrow by bus.
Heathrow - Paris by plane.
alph% grep us dataA
Oxford - Heathrow by bus.
alph% grep 'are as' dataA
```

```
My travel plans are as follows:
alph%
```

16 wc - Counting Words

The wc command counts the lines, words and characters in a file. The format of `wc` is

wc [-l] [-w] [-c] *filename...*

-l Prints the number of lines in the files.

-w Prints the number of words in the files.

-c Prints the number of characters in the files.

If no options are specified, wc prints out all three.

For example,

```
alph% wc dataA
3 16 85 dataA
alph% wc -l dataA
3 dataA
alph%
```

17 Starting Remote Clients

The most convenient way to connect to remote machines is to use `ssh` since this gives you a secure encrypted connection and automatically sets up the display. If however you cannot use `ssh` then you will have to setup the DISPLAY variable by hand. For more instructions on this see
`http://www.maths.ox.ac.uk/help/faqs/login/`.

18 Shells

Once you have understood the level of windows you might then consider exactly how you interface with the computer to execute the basic Unix commands. This is achieved by something called a shell. A shell is a program which can run within an xterm. It is not only a command interpreter but can also be used as a programming language. Inside the shell you type the basic Unix commands listed above. However, many shells (and there are many of them) have additional features such as automatically finishing the typing of words, the setting of "aliases" which abbreviate commonly used commands and the ability to keep a history of commands that you type so that the commands may be used again without retyping them.

There are four shells currently available in the Institute:

- sh. This is the basic Unix shell (the "Bourne" shell) with very few features.
- csh. This is the "C-shell" with extensions based on the C programming language.
- tcsh. This is the "Extended C-Shell" which contains many improvements on csh. This is the default shell in the Institute which uses the file .cshrc for its configuration.
- bash. This is a hybrid between tcsh and sh which is preferred by some people - the "Bourne Again Shell".

More information can be found on these by looking up their appropriate manual pages.

19 Files and Pathnames

When using the ls and cd commands you will have become familiar with the idea of directories and filenames. An extension of this idea is being able to refer to any file no matter which directory you are in. So, for instance, when you first login you might type

```
alph% pwd
/home/stlawrence/user/newuser
alph% ls -F
Mail/ code/ fortran/ old/ ques.1
alph%
```

Each level of the directory tree is split by a slash, /. The output of the second command means that there are four directories called Mail, code, fortran and old, and one file called ques.1.

You can read this file by typing

```
alph% more ques.1
```

or by typing

```
alph% more /home/stlawrence/user/newuser/ques.1
```

This example demonstrates the use of an absolute pathname. Any pathname beginning with / is an absolute pathname, anything else is considered relative to the current working directory.

The same result would be obtained by typing

```
alph% pwd
/home/stlawrence/user/newuser
alph% cd fortran
alph% pwd
/home/stlawrence/user/newuser/fortran
alph% more /home/stlawrence/user/newuser/ques.1
```

There are some special characters for referring to pathnames.

- . (Single fullstop). Refers to the current directory.
- .. (Two fullstops). Refers to the directory one level immediately up from the current directory.
- ~newuser (Tilde followed by a user name). Refers to the home directory (the directory first entered into on login) of a user.
- ~(Tilde on its own). Refers to your home directory.

So, for instance, to move the file ques.1 from the home directory into the fortran directory, you could type

```
alph% cd
alph% mv ques.1 fortran
```

or

```
alph% cd
alph% cd fortran
alph% mv ../ques.1 .
```

or even

```
alph% cd ~/fortran
alph% mv ~/ques.1 .
```

20 Variables and Environments

One of the features of shells is the ability to set variables and environment variables. Variables and environment variables are used by the shell and by programs as an indication of a users preference, and by users as a shorthand for words or lists of words. For example, the lpr command is used to print a POSTSCRIPT file to a printer. To print the file file.ps to the printer in room G17, you could type

```
lpr -Plpg17 file.ps
```

However, when printing a large number of files this could become laborious. When used like this

```
lpr file.ps
```

the lpr command checks the environment variable PRINTER; (it is automatically set to your nearest printer) and then prints to that printer.

So, for instance if you wish to print to a different printer, say lpg17, the command above

```
alph% lpr -Plpg17 file.ps
```

could be achieved by

```
alph% setenv PRINTER lpg17
alph% lpr file.ps
```

In addition, commands like dvips check PRINTER as well. Another commonly used variable is DISPLAY, which is required by X11. These variables are automatically set up upon your login, but can be changed by using the setenv command.

Variables, like environment variables, are used to store information. When they are set, you give them a name (like *foo*), but when they are referenced, you prefix them by a $. For instance, if you were using the filename shortcontents often, you might use

```
alph% set foo=shortcontents
alph% echo $foo
shortcontents
alph% cp $foo short
alph% cp $foo contents
alph% lpr -Plpg17 $foo
alph% rm $foo contents
```

where the last line removes the files $foo (which in this case is shortcontents) and contents.

To list all the environment variables and the variables which are currently set, type `setenv` or `set` on a line by themselves. Often variables and environment variables are set in a shells configuration script. See the section entitled "Configuring Your Working Environment".

21 Wildcards

Most shells have a mechanism which indicates groups of letters. For example, to list all the files in a directory which begin with the letters abc, type

```
alph% ls abc*
```

or to list all the four–letter files which end in h, type

```
alph% ls ???h
```

The commonly used wildcards are

- * (Asterix). Any character or any number of characters (or no characters at all).
- ? (Question mark). Any single character (not a blank).

Thus,

```
alph% ls
about above around before
alph% ls a*
about above around
alph% ls a??u*
about around
alph% ls *r*
around before
alph%
```

Note that wildcard and variable expansion occurs in the **shell** not in the command that is run (as in DOS). So it does not work as might be expected. One might niavely think the command

```
mv *.foo *.bar
```

would rename each file ending in the suffix .foo to end with the suffix .bar, but in fact it would most likely result in loss of data. To achieve this end, you could type

```
alph% foreach f (*.foo)
foreach?  mv $f ‘basename $f‘.bar
foreach?  end
alph%
```

For more information on wildcards, see the manual pages for the various shells.

22 Filename Expansion

The bash and tcsh shells have the feature of filename expansion. Expansion is carried out by using the **TAB** key. For instance,

```
alph% ls
about above around before
alph% more b{TAB}
```

will then produce the line

```
alph% more before
```

In addition, filename expansion will also work with the names of commands. To use the xmaple package, you could type

```
alph% xmap{TAB}
```

and the word xmaple will appear.

If the letters you have given are not enough to identify the command you mean you will hear a beep. You need to add another letter and press TAB again. The above example may give such a result since the commands xmaple-VR4 and xmaple-VR5 may also exist. Alternatively pressing `CTRL-d` (usually denoted as ^D) will list all possible completions from which you can then complete the choice.

23 Redirection and Pipes

Most shells support three standard means of communication with the user: the so called stdin (standard in), stdout (standard out) and stderr (standard error). stdin is usually the keyboard, both stdout and stderr are usually the monitor (in the case of X11, stdout and stderr are the window in which the shell is running). It may be convenient, for instance, to run a program and instead of it printing its results on the screen to put those results in a file. This is achieved by rerouting stdout.

```
alph% date
Mon Feb 20 12:22:00 GMT 1995
alph% date > todays.date
alph% cat todays.date
Mon FEB 20 12:22:00 GMT 1995
alph%
```

In this case, the character > redirects stdout to a file called todays.date. This means that a file called todays.date will be created (and in some shells erasing any previous contents in that file) and will contain the output from the command date (note that stderr will still be the monitor or an X term window).

An analogous procedure works to replace the function of typing in a set of values. If you had a program called addup which took all the numbers on one line of data and added them up, then you could edit a file called input.deck containing the input.

For example,

```
alph% cat input.deck
5 and 7 Plus 2
alph% addup < input.deck
The answer is 14
alph%
```

One could then possibly try

```
alph% date > todays.date
alph% cat todays.date
Mon Feb 20 12:22:00 pm GMT 1995
alph% addup < todays.date
The answer is 33
alph% addup < todays.date > answer
alph% more answer
The answer is 33
alph%
```

Note that the shell sees the first word on the line as the command, then anything after the < as the names of files to provide input, and then anything after the > as files in which to put output.

An analogous procedure to that used above (taking the output from one command and using it immediately

as the input of another) is also used in a construction called a pipe.

For example,

```
alph% date | addup
The answer is 33
alph%
```

A common usage of this is when running a command and to look at the output one page at a time.

```
alph% ls -l /usr/bin | more
```

In this case, `ls -l` generates a long listing of the files in the directory /usr/bin and then sends the output to the command more which then prints it on the screen one page at a time.

For more information on redirection and pipes (including many more types of redirection) see the manual pages of the shells.

24 Job Control

One final aspect of the shells is called job control. It is possible to run one or more than one command at a time in a given window.

For example, typing the command `xclock` in an xterm window will bring up a clock. However, you will notice that in the window you typed xclock the "prompt" does not come back up just yet. By using the "Skull and Crossbones" (kill) command from the "Window" menu, you can kill the xclock. There will be a small error in the xterm window, but the prompt will come back. To fix this, you can use an & to send the job to the background. So, typing `xclock &` will bring up the clock, run it in the background, and give you your prompt back. You can run any number of programs (within the machines capacity) from a given window by using the & character.

So, for example, if you have a program called prime which computes the first 500 prime numbers, but takes 10 minutes to run, you could type the following:

```
alph% prime > prime.out &
[2] 27103
alph%
```

and the prime program will run away happily for 10 minutes putting its output in the file prime.out; you may then continue working on another task (the number 27103 is an indicator of the process number which would be shown on typing the command ps and the number in square brackets is the job number in this shell). When it has finished, you will receive a message such as

```
alph%
[2] prime Done.
alph%
```

which indicates that the job has finished. To list the current jobs being run from within a shell, type jobs,

```
alph% jobs
[1] + Running xclock
[2] - Running prime
alph%
```

Jobs can be put in the foreground and background by using a combination of typing ^Z (suspend), fg (foreground) and bg (background). More information on job control can be found in the manual pages of the shells, and information on how to find out all the processes running at any given time can be found by typing `man ps`.

25 Configuring Your Working Environment

There are a number of files which are used to configure your working environment. They are as follows:

.cshrc Used to configure the C Shell and Extended C Shell

.profile Used to configure the Bourne Shell

.bashrc Used to configure the Bourne Again Shell

.emacs Used to configure the emacs program

.login Read whenever you login from the terminal and when using telnet

.logout Read whenever you logout from the terminal and when using telnet

.newsrc Configuration file for news readers

.fvwmrc Configuration file for the FVWM Window Manager

.plan A statement about yourself printed whenever someone "fingers" you

.project A one line statement about yourself, printed whenever someone "fingers" you

.xsession Configures your X sessions (*e.g.* contains information about which programs to run).

They can each be modified by using a text editor. A default account will have the files .cshrc, .xsession, .login. Each of these files is configured to source additional files with names .cshrc-user, .xsession-user, .login-user. Never modify the basic files, always put changes in the -user files.

Note that the configuration files .profile, .cshrc and .bashrc need only be present if you are using that particular shell. In the -user version of these files you might wish to include different *aliases* and commands to set environment variables.

26 Mtools — floppy disk tools

The Mtools package allows the transferring of files to and from floppy disks using most Linux PCs. The commands are similar to those of MS-DOS prefixed by the letter "m". Some of the commands are:

mattrib Change MS DOS file attribute flags.

mcd Change MS DOS directory.

mcopy Copy MS DOS files to/from Unix.

mdel Delete an MS DOS file.

mdir Display an MS DOS directory.

mformat Add an MS DOS filesystem to a low-level formatted diskette.

mlabel Make an MS DOS volume label.

mmd Make a MS DOS subdirectory.

mrd Remove a MS DOS subdirectory.

mread Low level read (copy) an MS DOS file to Unix.

mren Rename an existing MS DOS file.

mtype Display contents of an MS DOS file.

mwrite Low level write (copy) a Unix file to MS DOS

More information can be found in the manual page and in the Files Frequently asked Questions.

27 Xman

`xman` is a program used for reading Unix manual pages (the pages available using the `man` command).

28 Xcalc

`xcalc` is a X windows based calculator, usable by the mouse or by typing the numbers in from the keyboard.

29 Finger

`finger` is available on all the Institutes computers and is used for finding out information about other users on the Internet. See the manual page for more information.

30 Compression Utilities

See the Files Frequently asked Questions

31 Printing and Scanning

See the Printing Frequently asked Questions

32 Remote Logging In

See the Login Frequently asked Questions

33 File Transfer

See the Files Frequently asked Questions