Introduction to UNIX

Contents

1. Background

1.1. A Brief History of UNIX

1965 AT&T (Bell Labs), GE, IBM and Project MAC at MIT joined together to develop the time-sharing MULTICS system. (Multiplexed Information and Computing Service)

1969 AT&T drops out of MULTICS. A system that was supposed to support 1000 on-line users can barely handle 3!

1970 Bell Labs (Ken Thompson & Dennis Ritchie) develop UNIX.

1972 UNIX OS rewritten in C.

1975 Thompson begins a 1 year sabbatical at Berkeley. AT&T began licensing UNIX to universities.

1978 First Berkeley Software Distribution (BSD) version of UNIX released.

1982 AT&T announces official support for UNIX and makes its' first commercial release; UNIX System III.

1985 AT&T publishes UNIX System V Interface Definition (SVID) in an attempt to standardise the UNIX interfaces.

1990 UNIX International (a consortium of System V UNIX users) releases SVR4.

Detailed UNIX family tree at http://www.levenez.com/unix/

Essentially, there are two main strains of UNIX

1. BSD (Berkeley Software Distribution).
2. SVR4 (System 5, Release 4).

1.2. The Solaris Operating System

SunOS is the SUN implementation of the UNIX OS.

Solaris is SunOS packaged with a number of additional tools and a GUI environment.

Solaris Version		SunOS Version	UNIX Type
	Solaris 1.0	SunOS 4	BSD
	Solaris 2.0	SunOS 5	SVR4
Solaris 6	Solaris 2.6	SunOS 5.6	SVR4
Solaris 7	Solaris 2.7	SunOS 5.7	SVR4
Solaris 8	Solaris 2.8	SunOS 5.8	SVR4

2. Logging In and Out

You can log in at the CDE (Common Desktop Environment) by entering a valid username and password.

To log out from the CDE click on the “EXIT” button on the “Front Panel” window at the bottom of the screen.

A locked screen may be unlocked by entering the root password. This is useful if you need to access a workstation after a user has gone home or while they are out on holiday.

Once logged on to a workstation, you can log on to another networked workstation by using either the rlogin or telnet commands.

e.g. % ***rlogin*** acp123
% ***telnet*** acp123

To log out from a shell or a remote session use either ***exit*** or ***CTRL+D.***

When in a login shell, ***logout*** may also be used to log out. A login shell is any shell in which you have been prompted for a username and password.

3. UNIX Command Syntax

The general form of a UNIX command is as follows;

% command [-options] [arguments]

Options;

- Modify the way a command works.
- Are typed after the command and begin with a - (minus sign). There must be a space between the command and the first option.
  e.g. % ***ls -la*** or ***% ls -l -a***
- Are always optional, some commands have no options (e.g. ***clear***)

Arguments;

- Change what a command works on.
- Are typed after the command, and any options used. There must be a space between the command, or last option, and the first argument. Where there is more than one argument, they must be separated by spaces.
- May be optional or compulsory, depending on the command.
  e.g. ***ls *.txt*** (optional)
  or ***mv file1 file2*** (compulsory)

Some commands have neither options nor arguments. (e.g. ***pwd***, ***clear***, ***hostid***)

UNIX commands are simply files. For instance, the ***ls*** command is a binary file located in the directory /***bin***.

Some commands are not actually UNIX commands but instead are built-in commands within UNIX commands. (e.g. ***history***, ***setenv*** are built-in to the C shell command ***csh***) These commands do not have separate binary files but instead are part of the executable into which they are built-in.

Some commands have both UNIX and shell built-in versions. (e.g. ***cd***, ***echo***, ***kill***)

When a command name is entered the PATH environment variable is searched to find the location of the binary executable.

Use the command, ***which***, to find out which copy of a command is actually being executed.

Use the command, ***whereis***, to locate the binary file of a command whose path is not in the PATH environment variable.

4. Manual Pages

Manual pages (man pages for short) are the built-in UNIX on-line help documentation. (NB. They may not always be installed.)

Read them, read them, read them! The man pages are a vital source of information. If you wish to improve your UNIX you must become familiar with them. Keep revisiting them, each time you will get a little more out of them.

The manual pages are organised into sections covering different types of material. Major sections are numbered 1,2,3, etc. Where required, subsections are created and numbered 1B, 1C, etc. See ***man intro*** for an explanation of the classification used on your system. Table 1, lists the various sections of the manual pages and indicates the type of content each section deals with.

Section	Description
1	Commands and application programs.
2	System calls and errors.
3	Functions and libraries.
4	File formats.
5	Miscellaneous.
6	Games and demos.
7	Special files
9	Device driver interfaces.

Table 1

The manual pages are accessed using the ***man*** command.

e.g. % ***man {commandname}***

The ***man*** command looks for the content of the manual pages in the directories listed in the **MANPATH** environment variable. Sometimes man pages are installed in unusual locations and **MANPATH** has to be modified so that they can be accessed. e.g. The ***perl*** manual pages are located in the directory ***/usr/CVperl/man***, in order to access them **MANPATH** needs to be modified as follows, ***setenv MANPATH "${MANPATH}:/usr/CVperl/man"***.

man Examples	Description
man man	Display the man pages for the ***man*** command.
man intro	Display the man pages for ***intro*** which lists all the commands in section 1 of the manual pages and provides a one line description of each.
man -s4 intro	Display the man pages for ***intro*** in section 4.
man -s5 filesystem	Display the man pages for ***filesystem*** which is in section 5.
man perl	Display the man pages for perl. Ensure that the **MANPATH** is set correctly first.

Table 2

If invalid options are entered after a command, the system normally displays a usage statement listing all the valid options for the command. These are useful in jogging the memory as to the available options. You can get the system to display the usage statement for a command by knowingly supplying an invalid option.

e.g. % ***ls*** **-^** (I do not know of any commands that have ^ as a valid option)
ls: illegal option -- ^
usage: ls -1RaAdCxmnlogrtucpFbqisfL [files]

Note. This does not work with commands that have no options. (e.g. ***pwd***, ***cd***, etc.) But then again, in these cases there is nothing to remember!

5. The UNIX Filesystem

A filesystem is a structure on a hard or floppy disk which enables files and directories to be stored there.

The filesystem holds lots of information about the files in it. This information includes the file type, the access permissions it has, who owns it, the group it belongs to, the size of the file, the date and time it was last modified and its name.

Figure 1 below shows a portion of a sample UNIX filesystem.

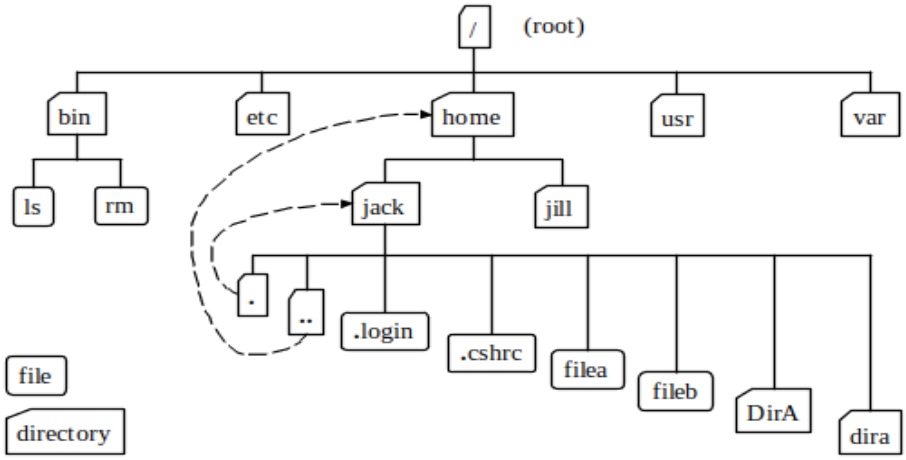

Figure 1

Table 3, identifies some special directories on the UNIX filesystem.

Special Directories	Meaning
. (dot)	Current directory.
.. (double dot)	Parent directory.
/ (slash)	Root directory.
~ (tilde) (Not bourne shell)	Current user's home directory.
~b2t (Not bourne shell)	User b2t's home directory.

Table 3

Filenames may contain any character except /, which is reserved as the separator between files and directories in a pathname. Filenames are usually made of uppercase and lowercase letters, numbers, "." (dot), and "_" (underscore). Other characters (including spaces) are legal in a filename - but they can be hard to use because the shell gives them special meanings. It is recommended that filenames only use letters, numbers, dot, and underscore characters.

A directory is really just a special kind of file, so the rules for naming directories are the same as the rules for naming files.

Metacharacters are characters or combinations of characters that can be used to represent all or part of directory or file names

Metacharacter	Meaning
?	Matches any single character.
*	Matches zero or more occurrences of any character
[…]	Represents a set of characters of which any one character can match. e.g. [a-z], [A-Z] or [0-9]

Table 4

5.1. Navigating the UNIX Filesystem

This section covers the basic commands (***pwd, ls*** and ***cd***) used for exploring the UNIX filesystem.

The ***pwd*** command (print working directory) prints the absolute pathname of the current directory.

The ***ls*** command (list) lists the contents of a directory. The long listing option of the ***ls*** command produces the following output.

% ***ls -l***

```
-rw-r-----  1 jbloggs   support  123   Jul 11  10:00  file1
drwxrwxr-x  2 jbloggs   support  345   Jul 11  09:00  dir1
lrw-rw-r--  1 jbloggs   support    1   Jul 12  13:30  file2 -> file1

1    2      3    4         5      6       7      8      9
```

The columns in the long listing have been numbered 1-9 in the listing above. They have the following meaning.

1. The file type.
   - `-` An ordinary file.
   - `d` A directory.
   - `l` A symbolic link.

		c A character device. (terminal or printer) b A block device. (disk)
2.	File permissions.	(r = read, w = write, x = execute)
3.	Number of links.	A link is a method of giving a file more than one name. This column indicates how many links a file has.
4.	Owner.	The login name of the user that created the file.
5.	Group that file belongs to.	A group is a collection of related users.
6.	File size.	The size of the file in bytes.
7.	Date.	The date when the file was last changed.
8.	Time.	The time when the file was last changed. If more than 6 months old then the year is shown instead of the time.
9.	Filename.	The name by which the file is known.

Example	Description
ls	List the contents of the current directory
ls -a	List the contents of the current directory, including any hidden files. (i.e. files beginning with a "." dot)
ls -l /usr/tmp	Print a long listing of the contents of the directory /usr/tmp. (See below)
ls -ld 510*	Print a long listing of the contents of the current directory that begin with 510. If a directory matches 510* then list only the directory and not its contents.
ls -latr ~jsmith/parts	List the contents of the parts directory belonging to user "jsmith". Print a long listing, including hidden files. Sort the list by modification time and reverse the order so that the most recently modified file is at the bottom of the list
ls a*	List all files beginning with a.
ls ?	List all files with single character names.
ls [0-9]*	List all files beginning with a digit.
ls -li ..	Print a long listing of the parent directory and include the files' inode number.

Table 5

The ***cd*** command (**c**hange **d**irectory) is used to change from the current directory to another directory. ***cd*** does not have any options and takes as an argument the pathname of the directory to which you wish to change to. If no pathname is supplied, then the user's home directory becomes the current directory.

5.2. Absolute and Relative Pathnames

Files and directories in the filesystem are located using pathnames. Pathnames can be either absolute or relative. An absolute pathname is the complete path from the top level root directory right down to the target file or directory. A relative pathname is the path from the current location in the filesystem to the target file or directory.

Absolute Pathnames	Relative Pathnames
Start at the root directory.	Start at the current directory.
Always start with a slash (/).	Never start with a slash.
The absolute pathname to a file is always the same.	The relative pathname to a file depends on the current directory.

Table 6

Which of the following examples are using absolute or relative pathnames?

1. cd /usr/tmp
2. cd parts
3. cd /
4. cd ../../files/
5. cd ~/b2t/parts
6. cd

(Answers:- 6 = abs., 5 = abs., 4 = rel., 3 = abs., 2 = rel., 1 = abs.)

6. Users and Groups

When users log on to a UNIX system, they are given a user identity (UID) and group identity (GID). These identifiers are numbers, however, both have corresponding text values also.

The ***id*** command can be used to check your UID and GID.

e.g. % id
uid = 123(jbloggs) gid = 101(support)

The ***groups [user]*** command shows all the groups that the user belongs to.

e.g. % ***groups jsmith***
support training

6.1. Changing a File or Directory's Owner and Group

As we have seen earlier, all files and directories have an owner and group associated with them. The following commands may be used to change these values.

The ***chown*** (**ch**ange **own**ership) command is used to change the ownership of a file or directory.

The ***chgrp*** (**ch**ange **gr**ou**p**) command is used to change the group of a file or directory.

Note. The ***chown*** command can be used to change can be used to change both the ownership and the group at the same time.

Examples	Explanation
chown jsmith *file1*	Set username *jsmith* to be the owner of file *file1*.
chgrp -R b2t *parts*	Set the group on directory *parts*, and the entire filesystem branch below it, to group *b2t*.
chown jsmith:b2t *f2*	Set the ownership of file *f2* to *jsmith* and its group to *b2t*.

Table 7

7. File Permissions

There are three sets of read/write/execute permissions associated with files and directories: one set for the user or owner of the file, one set for the group of the file, and one set for everyone else.

Permissions are viewed using the ***ls*** command together with the ***-l*** option.

e.g. % ***ls -l***

-rwxr-xr-- 1 jbloggs support 123 Jul 11 10:00 file1

The 9 characters to the right of the first column indicate the permissions. These 9 characters are broken down into 3 sets of 3. The 3 sets of permissions are;

1. User (u) The owner of the file, usually the user who created it.
2. Group (g) The group of the file, usually the group of the creator.
3. Others (o) All other users not included in the 2 above sets.

Each set of permissions have 3 characters which indicate the presence of a particular access right. The access rights, in order of appearance are;

1. Read (r) Allows users in the set to read (view) the file's contents.
2. Write (w) Allows users in the set to modify the file's contents.
3. Execute (x) Allows users in the set to run the file as a program or to navigate into the directory.

If any of these rights are not granted the particular character (r, w or x) is replaced by a hyphen, -.

Permission	Explanation
rw-rw-rw	Everyone (owner, group and others) may read or write.
rwxr-xr-x	Everyone has read and execute access. Only the owner has write access.
rw-r-----	The owner has read and write access. The group has read access and everyone else has no access.

Table 8

7.1. Changing Permissions

The ***chmod*** (**ch**ange **mod**e) command is used to change permissions. Permissions can be changed by either using a symbolic notation or by using an octal value.

7.1.1. Changing Permissions Symbolically

This method turns specific r, w and x permissions on or off without affecting any of the other permissions for that file. The format of the ***chmod*** command used to change permissions symbolically is as follows.

% ***chmod*** *{permission set} {change} {permission} (filename or directory)*

% ***chmod {u,g,o,a} {+,-,=} {r,w,x}*** *(filename or directory)*

The set of permissions to change can have any of the following values.

- u User's permissions.

- g Group's permissions.
- o Other's permissions.
- a All 3 sets of permissions at once.

The change to make can have any of the following values.

- - Subtract this permission.
- + Add this permission
- = Set to exactly this permission.

The permission to affect can have any of the following values.

- r Add, subtract or set read permission.
- w Add, subtract or set write permission.
- x Add, subtract or set execute permission.

Examples	Explanation
chmod g+w *file1*	Add write permission to the group on filename file1.
chmod u+x *prog1*	Add execute permissions for the owner on filename prog1.
chmod g=r,o-rwx *f1*	On file f1, set the group permissions to read and remove all permissions for all others.

Table 9

7.1.2. Changing Permissions Using Octal Values

The permissions, read, write and execute are assigned octal values of 4, 2 and 1 respectively. When quoting a files octal permission 3 numbers are used, one for each set of permissions. (i.e. user, group and others) Each of these number is the given by the sum of the octal values of the individual permissions granted to the particular permission set.

e.g. rwxr-xr-- => {4+2+1} {4+1} {4} => 754

Examples	Explanation
chmod 666 *file1*	Set the permissions on *file1* to be rw-rw-rw-..
chmod -R 755 *dir1*	Set the permissions on *dir1*, and the entire filesystem branch below it, to rwx-r-xr-x.
chmod 640 *file2*	Set the permissions on *file2* to rw-r-----.

Table 10

7.2. Default Permissions

The ***umask*** command, with no arguments, returns an octal value that represents the current default permissions.

The ***umask*** command, when supplied with an octal value as an argument, sets the default file permissions assigned to all files you create. A umask value is usually set in one of the shell's initialisation files, for example ***.login*** or ***.cshrc***.

Files and directories have different default permissions. A file is never given execute permissions by default, instead, if execute permissions are required on a file they have to be added manually by using the ***chmod*** command.

It would be dangerous to give a file execute permissions by default. For example, if a file was created with default execute permissions and the line, ***rm -r /***, were added to

it and then the file was accidentally executed by typing its name on the command line then serious damage could be done to the system!

Examples	Default File Permission	Default Directory Permission
umask 022	rw-r--r--	rwxr-xr-x
umask 002	rw-rw-r--	rwxrwxr-x
umask 026	rw-r-----	rwxr-x--x

Table 11

The maximum default permission a file can have is rw-rw-rw or 666 in octal. For a directory it is rwxrwxrwx or 777.

The following example demonstrates how to calculate the octal value required for a desired default permission.

	File Permission	Directory Permission
Desired permissions	rw- r-- ---	rwx r-x ---
Max. default permissions (octal)	6 6 6	7 7 7
Desired permissions. (octal)	6 4 0	7 5 0
umask value (Max. - Desired)	0 2 6	0 2 7

Table 12

This suggests 2 different values for ***umask***, 026 for the files and 027 for the directories. Using 026 will result in directory permissions of rwxr-x--x which is not what we want. Using 027 will give the same result for files as using 026 would. Therefore the correct ***umask*** value to use in this instance would be 027.

8. Working With Files and Directories

Table 13 below identifies some of the commands used when working with files and directories.

Action	Files	Directories
Creating	vi, touch, shell redirection	mkdir
Examining contents	cat, more, view, head, tail	ls, du
Copying	cp	cp -r
Moving	mv	mv
Deleting	rm	rmdir, rm -r

Table 13

8.1. Creating Files

In this section we look at some ways of creating text files on UNIX. These are the ***vi*** editor, the ***touch*** command and shell redirection.

The ***vi*** text editor is the most common editor used on UNIX systems. ***vi*** is used to create and edit text files. See the “UNIX VI Editor Notes” sheet for commands used within ***vi.***

The ***touch*** command updates the access time and modification timestamp to the current time and date. If the filename supplied to the command does not exist then a new, empty, file is created.

Shell redirection can be used to create a file.

e.g. % ***ls -1 > file1*** (Redirects the output from ***ls -1*** into file ***file1***.)

8.2. Creating Directories

Directories are created using the ***mkdir*** (**m**a**k**e **dir**ectory) command.

e.g. % ***mkdir dir1***
% mkdir -p dir2/dir3/dir4

8.3. Looking at the Contents of a File

The command, ***cat***, which is short for "concatenate," puts files together (concatenates them) to make a bigger file. It can also display files on your screen.

If you want to "read" a long file on the screen, the ***more*** command can be used to display one screen or "page" of text at a time. ***more*** has it own set of built-in commands. Some of these can be seen in Table 14 below. The manual pages are piped through ***more;*** these commands are very useful when reading the content of the man pages.

***more* Command**	**Description**
[SPACE]	Display next page.
[RETURN]	Display next line.
nf	Move forward n pages.
b	Move backward one page.
nb	Move backward n pages.
/word	Search forward for word.
?word	Search backward for word.
v	Start the vi editor at this point.
[CTRL-L]	Redisplay current page.
h	Help.
:n	Go to next file on command line.
:p	Go back to previous file on command line.
q	Quit more (before end of file)

Table 14

The ***view*** command is a read-only version of the ***vi*** editor command. It is useful when you want to interactively browse through a file without making any modifications.

The ***head*** command prints the first few lines from a file.

e.g. % ***head file1*** (print the first 10 lines of file1)
% ***head -20 file1*** (print the first 20 lines of file1)

The ***tail*** command prints the last few lines of from a file.

e.g. % ***tail file1*** (prints the last 10 lines of file1)
% ***tail -30 file1*** (prints the last 30 lines of file1)
% ***tail -f logfile*** (prints lines as they are added to logfile)

e.g. % ***head -100 | tail -25 log1*** (prints lines 75-100 of log1)

8.4. Looking at the Contents of a Directory

The contents of a directory can be examined by using the ***ls*** or ***du*** (**d**isk **u**sage) commands.

e.g. % ***ls -R*** (recursively list subdirectories as well as current directory)
% ***du -a*** (print disk usage for all files and not just subdirectories)

8.5. Copying and Moving Files and Directories

Files may be copied by using the ***cp*** (**c**o**p**y) command.

e.g. % ***cp file1 file2*** (copy file ***file1*** to ***file2***)

% ***cp ../file1 ../../dir2*** (copy file ***file1*** in the current directory's parent directory to directory ***dir2*** which is 2 directories above the current directory.)

Directories are copied using the ***cp*** command with the ***-r*** option.

e.g. % ***cp -r dir1 dir2*** (copy directory ***dir1*** to ***dir2***)

The ***mv*** (**m**o**v**e) command is used to move or rename files or directories.

e.g. % ***mv file1 file2*** (move, or rename, file ***file1*** to ***file2***)
% ***mv dir1 dir2*** (move, or rename, directory ***dir1*** to ***dir2***)

8.6. Deleting Files

The ***rm*** (**r**e**m**ove) command is used to delete files. Unlike Windows, UNIX has no mechanism for undeleting files so care should be taken when using ***rm***. It is a good idea to list what you would like to delete before actually deleting it.

e.g. % ***ls *.txt***
% ***rm *.txt***

8.7. Deleting Directories

An empty directory can be deleted using the ***rmdir*** (**r**e**m**ove **dir**ectory) command. If a directory contains files and/or subdirectories it and all its contents can be deleted by using the ***rm*** command with the ***-r*** option.

e.g. % ***rmdir dir1***
% rm -r dir2

8.8. Significance of Permissions

The significance of the read, write and execute permissions on working with files and directories is summarised in Table 15 below.

Permission	Significance on Files	Significance on Directories
read (r)	Grants permission to view the contents of a file. (e.g. can use ***cat***, ***more***, ***view***, etc.)	Grants permission to view the contents of the directory. (e.g. can use ***ls, du -a***, etc.)
write (w)	Grants permission to modify the contents of a file. (e.g. can use ***vi***, ***cp***, ***mv***, shell redirection, etc.)	Grants permission to modify the contents of the directory. (e.g. create a file or make a sub-directory)
execute (x)	Grants permission to run a program or script.	Grants permission to navigate to the directory using ***cd***.

Table 15

9. Processes

At the heart of UNIX lies the concept of a process. A process is an independently running program that has its own set of resources. On a UNIX system at any one time there may be tens or hundreds of processes running.

Commands that are entered, with a few exceptions, create a process that runs for the duration of the command execution. Each shell that is opened creates a process. When a command is entered inside a shell a new process is started. The new process is a ***child*** of the shell process. The shell process is the ***parent*** of the new command process.

Many processes are not associated with a user at a terminal. For example, a process may be executed to listen on the network for incoming requests for ***telnet*** sessions. These types of processes are referred to as ***daemons***.

On UNIX, the finite resources of the system, like the memory and the disks, are managed by one all-powerful program called the ***kernel***. Everything else on the system is a process.

9.1. Listing Processes

The list of currently running processes can be viewed by issuing the command ***ps -ef***. This command returns the following information for every running process.

- UID The login name of the user running the process.
- PID The process identification number.
- PPID The process ID of the parent process.
- C Processor utilisation for scheduling. (obsolete)
- STIME The starting time of the process. In hours, minutes and seconds if less than 24 hours age, otherwise in months and days.
- TTY The controlling terminal for the process.
- TIME The cumulative execution time for the process.
- CMD The full command name and all its arguments. (Limited to 80 characters)

9.2. Killing Processes

It is sometimes necessary to stop a process that has started owing to mistakes, system problems, etc. The ***kill*** command is used to “kill” or stop running processes.

e.g. % ***kill 1234*** (Kills the process with a process identification number, PID, of 1234.)

When a process is killed, a signal is sent to the process by the kernel. When kill is executed, the system sends a default signal of 15 to the process. This usually causes the process to stop however sometimes it does not. To force a kill on a process, use signal 9 in the kill command.

e.g. % ***kill -9 1234***

Roughly, kill -15 means, “Please die when you’re ready”, whereas kill -9 means, “Die now, no matter what”. Kill -9 should only be used as a last resort as the process will not “tidy up” before it dies.