Operating Systems

GNU/Linux & UNIX.

UNIX

There have been many different operating systems (OSes) developed throughout the history of computing, many were proprietary and tied to the vendors' hardware architecture. As these architectures became obsolete the OSes fell out of use.
The UNIX operating system however has had particular longevity in computing, its development can be traced back to 1969 at AT&T Bell Labs (New Jersey) and it is still in use today in its many variants and derivatives.

The first usable implementation of UNIX was run on a DEC PDP-11 "minicomputer". It was called a "minicomputer" as it only needed to occupy a single room rather than a whole floor or suite!

In the early 1970s computer hardware was very large and expensive, costing much more than a family house. These highly specialised devices were used for scientific, engineering and commercial tasks by specially trained staff. Because the computer hardware resource was so valuable, CPU time was shared amongst the various users; "jobs" were usually queued to be run, often overnight. The users' input/output console was an electro-mechanical teletype.

During the 1970s UNIX continued to be developed, primarily by Dennis Richie and Ken Thomson. In 1975, under a license agreement, UNIX was circulated to US universities for the cost of the media.

Because Bell Labs included the source code with the UNIX distribution university researchers were able to modify and extend UNIX. In 1974 Ken Thomson spent a year teaching at Berkeley, University of California and UC Berkeley became particularly noteworthy in the development of UNIX.
BSD (Berkely Software Distribution) is a derivative development of UNIX with much of the early development of Internet protocols being carried out on BSD systems.

During the late 1980s and early 1990s disagreement broke out amongst UNIX vendors regarding the development and standardisation of UNIX.
BSD UNIX had to purge all AT&T copyright code and further ownership disputes and litigation has occurred sporadically between vendors since.
(e.g. SCO vs IBM)

GNU

In 1983 Richard Stallman initiated the GNU project to promote open source software development.

The aim of this project was the creation of a non-proprietary operating system which could be used freely like UNIX was initially. The OS would be very UNIX-like but could not use any commercial copyright code.
This project created many of the tools required to build a new OS.
(i.e. gcc GNU C compilier)
The OS developed by the GNU project “HURD” has been in development since 1990 but is not in mainstream usage.

General Public License

The licensing of GNU contributions grants anyone the freedom to run, copy, modify or distribute the software, provided they grant the same rights to their contribution.
This is sometimes referred to as “copyleft”, i.e. the opposite of copyright.

Linux

In 1987 Prof Andrew S. Tanenbaum developed **Minix**, a limited UNIX look-alike OS.
Source code for this OS is distributed with Prof Tanenbaum’s book on operating system design.

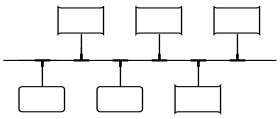
On the Aug 25th 1991 Finnish student Linus Torvalds annouced his own free operating system kernel on comp.os.minix.

Many of the tools developed for the GNU project were ported to the Linux OS kernel creating a complete, usable operating system.

As an open source UNIX-like OS, under general public license, Linux continues to attract new users and developers to this day.
Note that some purists insist that the OS should be termed “GNU/Linux”.

There are literally hundreds of OS distributions based on the Linux kernel.
Some of the most popular are:-

Android Mint Debian Ubuntu Fedora



UNIX/Linux File System

All types of data are held in files which are grouped into directories (“folders” in Windows terminology) these can, in turn, contain sub-directories.

The overall parent directory of the system is called “the root directory”, and is labelled **/** (forward slash)

Every user has a “home” directory which is normally identified by their username.

Often users’ home directories are contained in a directory called **/home**.

So for example the “home” directory for user **fred** may commonly be **“/home/fred”.**

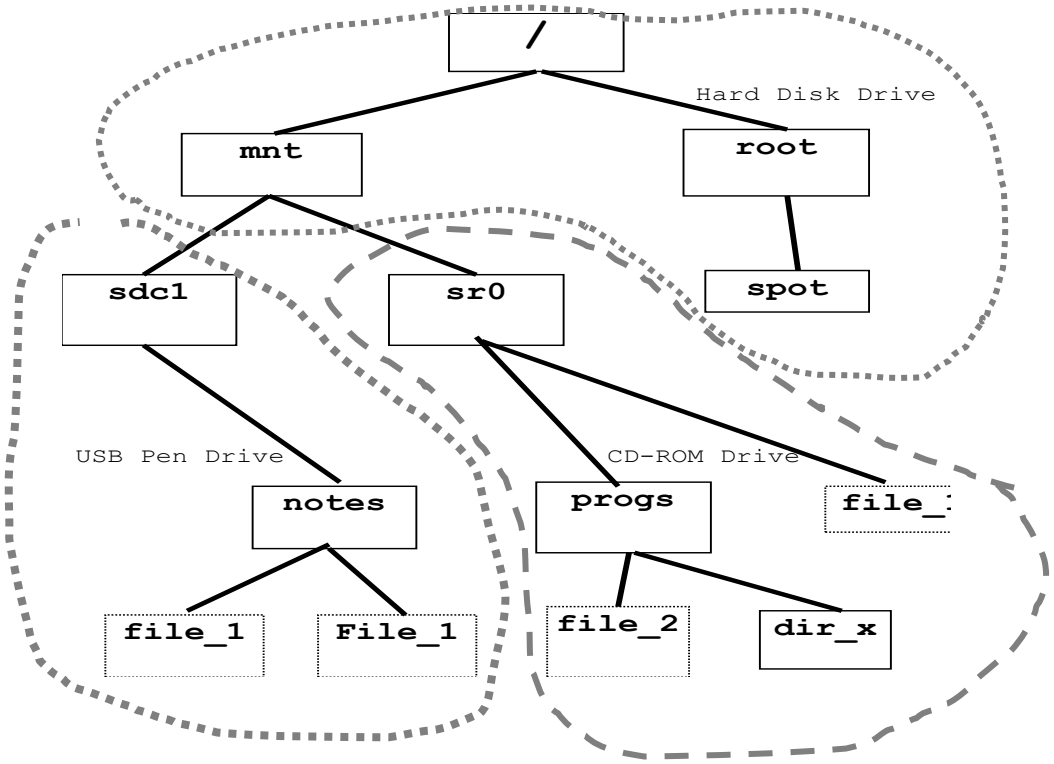
A user’s own home directory can be referred to easily using the **~** (tilde) character.

Note that **/home** is not normally your “home” directory and **/root** is not **the** “root” directory.

Sample File System

The diagram below shows 3 physical storage devices combined to create a single tree structure with "/" at the top.

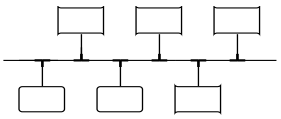
- **/** is on the hard disk drive
- **/mnt/sdc1** is on the CD-ROM
- **/mnt/sr0** is on the USB pen drive



Unlike Windows OSes UNIX does not use drive letters for different physical storage devices. This means that the physical storage arrangements of a UNIX system are slightly less obvious to the user.

In order to see this arrangement just use the "df" command

```
$ df
Filesystem      Use%   Mounted on
/dev/sda1        35%   /
/dev/sdc1        67%   /mnt/sdc1
/dev/sr0         100%  /mnt/sr0
```



File Access Privileges

Every item in the file system has an associated owner, referred to as the "**user**".
A user can be assigned to a particular "group", e.g. staff, students, admin.

Each item will also belong to a particular "**group**". So from the point of view of file access permissions there are 3 categories of user.

The owner of the file – "user"
Someone in the same group – "group"
Everyone else – "other"

Similarly there are three access attributes (read, write and execute) for each of the three categories of users, 9 in total

Permissions may be given for each user category to have certain privileges on a file or directory:-

r user read
w user write
x user execute
r group read
w group write
x group execute
r other read
w other write
x other execute

For example

r w x r - - - - -

shows that the owner has full rights (rwx) and members of the file's group may only read the file (r - -). All others cannot access the file at all (- - -).

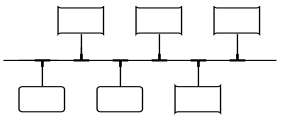
Note that in order to browse the content of a directory the execute (- - x) permission must be set.

Privileges can be specified conveniently by octal numbers:-

7 4 0 is equivalent to
(**111 100** 000) **rwx r - -** - - -

The command "chmod" changes access permissions:-

```
$ chmod 740 myfile
```



Command Shell

The shell is the most basic interface through which the user can control the kernel of the operating system. The command line interface (CLI) should be familiar to most computer users.

```
prompt$  command  argument1 argument2 argument3
```

As well as allowing the user to enter a command, the shell offers extra features to make the administrator's work easier.

Commands can be submitted in batches. This is useful if a job needs to run unattended, e.g. overnight.
The standard shell for UNIX was the Bourne shell developed by Stephen Bourne at Bell Labs.

- The output of one command can provide the input for the next command.
- The result of a command can be saved in a file for use later.
- Control structures.

Having a shell which supports control structures means that loops and decisions can be built into batches of commands to easily create utilities for carrying out routine administrative work. Thus reducing the need for a technician to laboriously type multiple commands.

bash

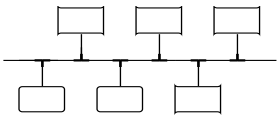
As the Bourne shell is proprietry to Bell Labs, the GNU/Linux community developed a GPL alternative which is referred to as "bash". (Bourne Again Shell). Although many alternative shells exist this is the default for Linux.

Scripts

Shell scripts are less sophisticated than applications developed using a high-level language which is then compiled into native code. However shell scripts are a very quick way to produce custom routines based around the use of existing Linux commands.

Interpreted scripts are used elsewhere in computing to allow programmability inside an application. For example MS Office applicatons can be enhanced or extended through the implementation of VBA scripts, web pages can use the scripting abilities of the client's browser by embedding scripts within the HTML page.

Network simulators and other engineering applications can be configured and automated through a scripting language.



For the remainder of this document it is assumed that the reader is familiar with the basic file system navigation and manipulation commands below.

Common UNIX/Linux Commands

The following commands are are used to explore the file system:

pwd - prints the name of the current working directory
ls - lists the names of all files in the current directory.
cd *dirname* - changes the current directory to that named.
cat *filename* - displays the contents of the file named.
cp *file1 file2* - copies the contents of the file1 to file2.
rm *pathname* - deletes the file named.
mkdir *pathname* - creates a subdirectory
rmdir *pathname* - deletes the empty directory named.

Wildcards in commands

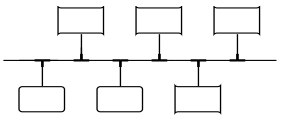
? - represents any single character in a filename
***** - represents anything (including blank)

Relative Pathnames

Pathnames which depend on the user's present working directory, they do not begin with a leading "/".

Absolute Pathnames

Pathnames which are independent of the present working directory and begin with a leading "/".



Key Linux Commands

Complex actions can be created by combining command utilities together. There are several important Linux commands which allow the user to create composite actions that are more sophisticated or specific than can be achieved with a single command on its own.

grep - the Pattern Matcher (Get Regular ExPression)

grep is a very useful utility for filtering the output produced by another command, by default it displays all lines which match a specified pattern or string.

grep *string filename*

Example:

```
grep fred userslist.txt
```

Lists only the lines from the file userslist.txt that contain the string ”fred”.

If the search string contains spaces it must be in double quotes:

grep “any text you want” filename

To omit lines containing a string use -v

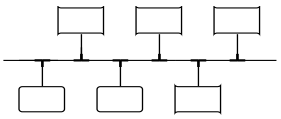
grep -v “text you do not want” filename

To list lines beginning with a character:

grep ^*The filename*

lists lines beginning with “The”

Note that the search target is expressed as a “Regular Expression”, this is a complex syntax that can be used to specify a set of possible string matches. Mastery of regex is a skill that Linux users aspire to.



Regular Expression Basics

. (dot)	matches any character
[abc]	matches any single character a ,b or c
[A-Z]	matches any single uppercase character
[0-9]	matches any single digit
Mon|Tue	matches strings “Mon” OR “Tue”
*****	matches the literal asterisk character itself, the backslash is used as the escape character.
^	matches the start point of a line
$	matches the end point of a line

Regex Example

(EH|FK)[0-9][0-9]	Matches “EH” or “FK” followed by any two digits. EH10 and FK12 would match but EH2 would not.

Quantifiers

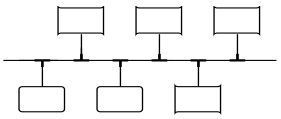
Somtimes we cannot specify how many characters there need to be in the match.

*****	matches any number (including none) of the previous character
+	matches one or more of the previous character
?	matches none or one of the previous character

Example

(EH|FK)[0-9]+	Matches “EH” or “FK” followed by one or more digits. EH2 and FK12 would match but so would EH7262542.

UNIX style regular expressions are used in many places in computing wherever pattern matching or string substitution is required.



Redirecting input & output

Normally, if the user does not specify otherwise, a command takes its input from the console (keyboard) and delivers its output to the console (screen).

Try running the “cat” command without any parameters, just “cat” on its own. You will find that whatever you type gets repeated on the screen. The cat command is reading its input from the keyboard and outputting it to the screen, you can terminate this by pressing <Ctrl><D>.

To redirect screen output to a file use:-

> filename	sends output to “filename”
	OR
>> filename	appends output to “filename”

e.g. **ls ~ >> homelist.txt**

To take input from a file rather than the keyboard:-

< *filename*	take input from “filename”

Another powerful technique is to take the output from one command and use that as input to another. This is referred to as “piping”. grep comes in useful here.

command1 | command2 The output from command1 is used as input to command2

```
$ ls public_html | grep Html
```

“ls” will list all the files in the directory public_html, “grep” will then filter this list so that only lines containing the string “Html” will pass through and be displayed.

The result of one command can also be embedded inside another.
For example, the **pwd** command returns the present working directory, if you would like to serach for any line mentioning this directory from a file then use:-

```
grep  `pwd` bigdirlist.log
```
(note the quote characters are “grave accents” **not** apostrophes)

Here **pwd** is run first and the result embedded in the command, so it becomes

```
grep /home/demo bigdirlist.log
```
before the grep operation is executed.

Asynchronous vs Sequential Statements (; &)

It is possible to enter multiple commands on the command line by separating them with semi-colons.

This is useful if the user knows that one of the commands will take a long which to complete.

Example:

```
$ bigpayrolljob ; archivesystem ; getupdates
```

Each command will not start until the previous one has completed; This is a “sequential” list.

Alternatively we could run commands simultaneously so that they are not dependent on each other (asynchronous). To do that we follow the command with an ampersand.

```
$ bigpayrolljob & archivesystem & getupdates &
```

If you put “&” after a command it runs “in the background”, which means that you get the command prompt back right way and can enter another command while the first is still running. So the previous commands could hjust as easily be run as:-

```
$ bigpayrolljob &
$ archivesystem &
$ getupdates &
$
```

More Key Linux Commands

sort – rearranging files or text output

By default, sort will rearrange any input or file it receives and put it into alphabetical order line by line.
If you don't like the order produced by an "ls" command then "pipe" the output through sort.

```
$ ls bigdirectory | sort
```
the output will be in alphabetical order

Sort can also reverse the sort order with "-r"

```
$ ls bigdirectory | sort –r
```

If you would prefer a numerical sort so that 9 comes before 10 for example use "-n"

```
$ ls bigdirectory | sort –n
```

sort also does not have to index on the first character on each line.

cut – isolating fields in text files

Whereas **grep** filters line-by-line **cut** isolates parts of a file vertically by choosing specific fields (columns) from text files.
For example if mylogfile.txt contains:-

1:Mon:10:12:start
2:Mon:11:08:run
3:Tues:07:21:end

Then the command:-

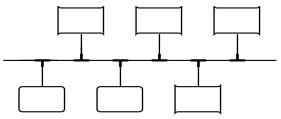
```
$ cat mylogfile.txt | cut –f3 -d:
```

Would produce the result:-

10
11
07

As the command takes the third field from each line (-f3), delimited on the colons (-d:).

Note that the default delimiter character for cut is a tab.



find – searching for objects in the filesystem

The “find” utility is a very powerful command which allows any file or directory to be found depending on any of its attributes. To use find you need to specify where in the filesystem to start looking from.

Examples
Look anywhere within the present working directory (.) for a file or directory named foobar.txt

```
$ find . –name foobar.txt
```

Search the whole of your own public_html directory for “html” files, note that the “*” wildcard must be escaped with a “\” in this instance.

```
$ find ~/public_html –name "*.html"
```

Find any files in your home directory which were modified less than 7 days ago.

```
$ find ~ -type f -mtime -7
```

Find any files that are bigger than 100000 characters (bytes) which were last modified more than 31 days ago.

```
$ find / -type f -mtime +31 –size +100000c
```

find can search based of practically any attribute, some of the most common are shown below.

OPTION	DESCRIPTION
`-atime days`	Time in days since the file was last accessed.
`-ctime days`	Days since the file was last changed.
`-mtime days`	Number of days since the file's data was last changed.
`-newer file`	File was modified more recently than *file*.
`-perm 755`	File's permissions are exactly *755*
`-size size`	File uses *size* space, in 512-byte blocks. Append *size* with `b' for bytes or `k' for kilobytes.
`-type type`	File is type *type*, where *type* can be `d' for directory or `f' for file
`-user fred`	File is owned by *user*.fred

find can also execute a command on all files that it has found using the "-exec" option.

Example, find any big old files and remove them, say files over 500Kbytes which have not been modified within the past year.

```
$ find / -size +500k –mtime +365 –exec rm {} \;
```

For every file found the command **`rm filename`** ; will be executed and the files will be removed (deleted).

Note that the "**`{}`**" shows where the found filename will be inserted.
The exec command must be terminated with a semi-colon, which in this case has to be escaped with a back slash.

Another Example
Search for any HTML files which have permissions rwx --- --- (700) and change them to 744.

```
$ find ~/pubic_html –name "*.html" –type f -perm 700
–exec chmod 744 {} \;
```

find is the useful for making any bulk changes to the contents of the file system

man pages – getting help (sort of)

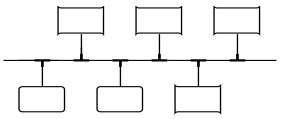
The default method for accessing help information is by using "man pages"

$ man *command*

Example
`$ man find`
`$ man grep`

This brings up comprehensive documentation on the command specified. This is useful to remind yourself about the options available with a particular command but it's not much use if you don't know which command to use in the first place.
To save space Puppy Linux does not include man pages internally, instead the man command invokes the web browser which accesses on-line man pages.

With many commands you can also use the "--help" switch get a brief overview:- Example: **`$ find --help`**



echo

"echo" merely echoes strings or values to the standard output (screen).
It is very useful for script messages, debugging or writing text to a file.

```
$ echo Hello there
Hello there

$ echo "this is the start of a run" >> mystuff.log

$ date   >> mystuff.log
```

Linux Pathname Manipulation Commands

dirname

Parses a pathname by returning only the directory part.

$ dirname /root/mydir/subdir/foobar

Result is /root/mydir/subdir/

basename

This command parses a pathname by removing the directory part returning only the filename.

$ basename /root/mydir/subdir/foobar.txt

Result is foobar.txt

readlink -f

This will provide an absolute pathname given a relative one.

$ readlink –f ~

Result is /root

$ readlink –f ../spot/myfile.txt

Result is /root/spot/myfile.txt