

JUST ENOUGH  
**LINUX**

# Contents

<b>Introduction</b>	<b>1</b>
Welcome!	1
Who is this book for?	1
Other Useful Resources	2
The Linux Information Project (linfo.org)	2
linuxconfig.org	2
raspberrypi.org	2
howtovmlinux.com	2
techmint.com: Linux Howto's Guide	2
<b>Linux Concepts</b>	<b>3</b>
What is Linux?	3
Executing Commands in Linux	5
The Command	6
Commands	9
Wildcards	9
Examples	10
Regular Expressions	11
Match a defined single character with square brackets ([ ])	13
Match at the beginning of a string (^)	14
Match at the end of a string (\$)	15
Match any single character (.)	15
Match when the preceding character occurs 0 or 1 times only (?)	16
Match when the preceding character occurs 0 or more times (*)	16
Match when the preceding character occurs 1 or more times (+)	17
Group parts of a search expression together (( ))	17
Find one group of values <i>or</i> another ( )	18
Extended Regular Expressions	18
Pipes ( )	19
Linux Directory Structure	21
/	22
/bin	22
/boot	22
/dev	22
/etc	23
/etc/cron.d	23
/etc/rc?.d	23

## CONTENTS

/home . . . . .	23
/lib . . . . .	23
/lost+found . . . . .	23
/media . . . . .	23
/mnt . . . . .	24
/opt . . . . .	24
/proc . . . . .	24
/root . . . . .	24
/sbin . . . . .	24
/srv . . . . .	24
/tmp . . . . .	25
/usr . . . . .	25
/usr/bin . . . . .	25
/usr/lib . . . . .	25
/usr/local . . . . .	25
/usr/sbin . . . . .	25
/var . . . . .	25
/var/lib . . . . .	26
/var/log . . . . .	26
/var/spool . . . . .	26
/var/tmp . . . . .	26
Absolute and Relative Path Name Addressing . . . . .	27
Pathnames . . . . .	27
Absolute Path Name Addressing . . . . .	27
Relative Path Name Addressing . . . . .	28
The ‘home’ short-cut . . . . .	28
Everything is a file in Linux . . . . .	29
Traditional Files . . . . .	29
Directories . . . . .	29
System Information . . . . .	29
Devices . . . . .	30
Linux files and inodes . . . . .	31
The file name . . . . .	31
The inode . . . . .	32
links . . . . .	33
Soft Links (aka symbolic links, aka symlinks) . . . . .	34
Hard Links . . . . .	35
Links Compared . . . . .	35
File Editing . . . . .	36
The nano Editor . . . . .	37
Scripting . . . . .	39
Writing our Script . . . . .	39
Make the script executable . . . . .	40
Place the script somewhere that the shell can find it . . . . .	41
<b>Linux Commands . . . . .</b>	<b>43</b>

## CONTENTS

File Administration	43
cd	43
chgrp	46
chmod	48
chown	54
cp	57
find	59
gzip	67
ln	70
ls	72
mkdir	77
mv	79
pwd	81
rm	82
rmdir	85
tar	87
Accessing File Contents	90
cat	90
cut	94
diff	98
grep	102
head	108
less	111
more	116
tail	119
File Systems	122
fdisk	122
mkfs	127
mount	130
umount	134
System Information	136
date	136
df	141
du	144
free	147
Processes	151
crontab	151
kill	155
killall	157
ps	159
top	164
Network	170
curl	170
host	173
ifconfig	176
ip	180

## CONTENTS

netstat . . . . .	186
ping . . . . .	192
scp . . . . .	196
sftp . . . . .	200
ssh . . . . .	204
traceroute . . . . .	207
wget . . . . .	209
Miscellaneous . . . . .	215
apt-get . . . . .	215
clear . . . . .	219
echo . . . . .	220
groupadd . . . . .	223
man . . . . .	224
modprobe . . . . .	227
passwd . . . . .	230
shutdown . . . . .	232
su . . . . .	234
sudo . . . . .	236
useradd . . . . .	241
usermod . . . . .	242
who . . . . .	245
whoami . . . . .	248
<b>Command Cheat Sheet . . . . .</b>	<b>250</b>
<b>Directory Structure Cheat Sheet . . . . .</b>	<b>253</b>
<b>Regular Expression Cheat Sheet . . . . .</b>	<b>254</b>
<b>‘find’ Cheat Sheet . . . . .</b>	<b>255</b>

# Linux Concepts

## What is Linux?

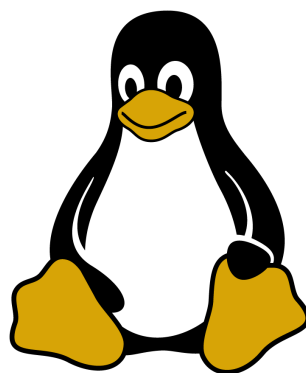
In it's simplest form, the answer to the question "What is Linux?" is that it's a computer operating system. As such it is the software that forms a base that allows applications that run on that operating system to run.

In the strictest way of speaking, the term 'Linux' refers to the Linux kernel. That is to say the central core of the operating system, but the term is often used to describe the set of programs, tools, and services that are bundled together with the Linux kernel to provide a fully functional operating system.

An operating system is software that manages computer hardware and software resources for computer applications. For example Microsoft Windows could be the operating system that will allow the browser application Firefox to run on our desktop computer.

Linux<sup>11</sup> is a computer operating system that is can be distributed as [free and open-source software](http://en.wikipedia.org/wiki/Free_and_open-source_software)<sup>12</sup>. The defining component of Linux is the Linux kernel, an operating system kernel first released on 5 October 1991 by Linus Torvalds.

Linux was originally developed as a free operating system for Intel x86-based personal computers. It has since been made available to a huge range of computer hardware platforms and is a leading operating system on servers, mainframe computers and supercomputers. Linux also runs on embedded systems, which are devices whose operating system is typically built into the firmware and is highly tailored to the system; this includes mobile phones, tablet computers, network routers, facility automation controls, televisions and video game consoles. Android, the most widely used operating system for tablets and smart-phones, is built on top of the Linux kernel.



The Linux mascot 'Tux'

---

<sup>11</sup><http://en.wikipedia.org/wiki/Linux>

<sup>12</sup>[http://en.wikipedia.org/wiki/Free\\_and\\_open-source\\_software](http://en.wikipedia.org/wiki/Free_and_open-source_software)

The development of Linux is one of the most prominent examples of free and open-source software collaboration. Typically, Linux is packaged in a form known as a Linux distribution, for both desktop and server use. Popular mainstream Linux distributions include Debian, Ubuntu and the commercial Red Hat Enterprise Linux. Linux distributions include the Linux kernel, supporting utilities and libraries and usually a large amount of application software to carry out the distribution's intended use.

A distribution intended to run as a server may omit all graphical desktop environments from the standard install, and instead include other software to set up and operate a solution stack such as LAMP (Linux, Apache, MySQL and PHP). Because Linux is freely re-distributable, anyone may create a distribution for any intended use.

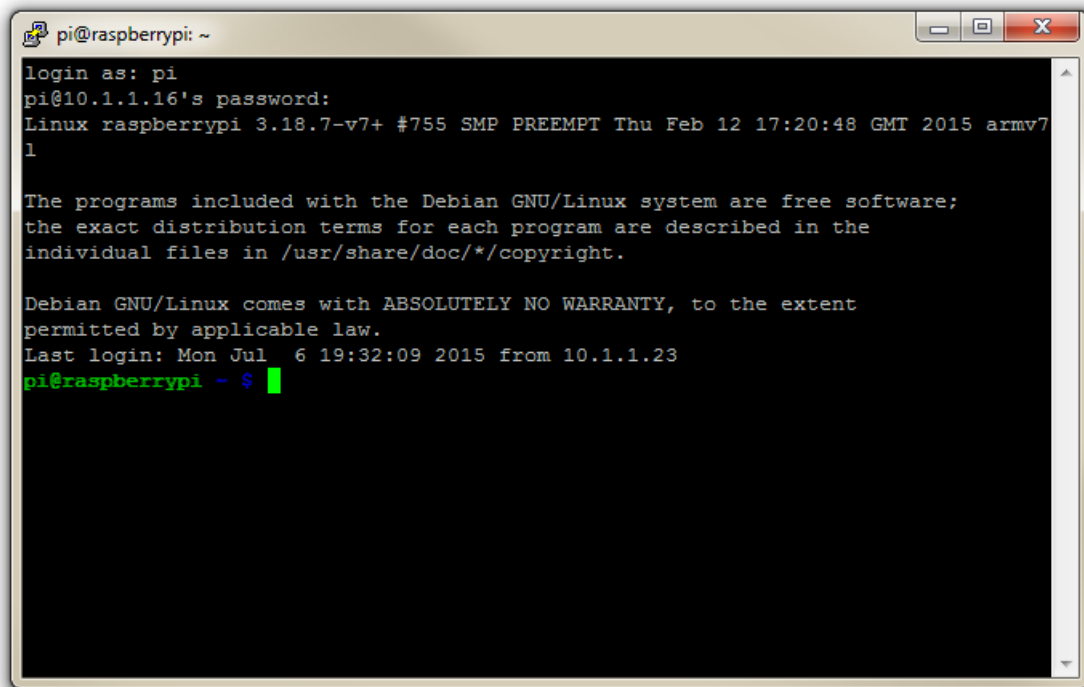
Linux is not an operating system that people will typically use on their desktop computers at home and as such, regular computer users can find the barrier to entry for using Linux high. This is made easier through the use of Graphical User Interfaces that are included with many Linux distributions, but these graphical overlays are something of a shim to the underlying workings of the computer. There is a greater degree of control and flexibility to be gained by working with Linux at what is called the 'Command Line' (or CLI), and the booming field of educational computer elements such as the [Raspberry Pi](http://raspberrypi.org/)<sup>13</sup> have provided access to a new world of learning opportunities at this more fundamental level.

---

<sup>13</sup><http://raspberrypi.org/>

## Executing Commands in Linux

A command is an instruction given by a user telling the computer to carry out an action. This could be to run a single program or a group of linked programs. Commands are typically initiated by typing them in at the command line (in a terminal) and then pressing the ENTER key, which passes them to the shell.



The Terminal

A terminal refers to a wrapper program which runs a shell. This used to mean a physical device consisting of little more than a monitor and keyboard. As Unix/Linux systems advanced the terminal concept was abstracted into software. Now we have programs such as LXTerminal (on the Raspberry Pi) which will launch a window in a Graphical User Interface (GUI) which will run a shell into which you can enter commands. Alternatively we can dispense with the GUI all together and simply start at the command line when we boot up.

The shell is a program which actually processes commands and returns output. Every Linux operating system has at least one shell, and most have several. The default shell on most Linux systems is bash.



## The Command

Commands on Linux operating systems are either built-in or external commands. Built-in commands are part of the shell. External commands are either executables (programs written in a programming language and then compiled into an executable binary) or shell scripts.

A command consists of a command name usually followed by one or more sequences of characters that include options and/or arguments. Each of these strings is separated by white space. The general syntax for commands is;

```
commandname [options] [arguments]
```

The square brackets indicate that the enclosed items are optional. Commands typically have a few options and utilise arguments. However, there are some commands that do not accept arguments, and a few with no options. As an example we can run the `ls` command with no options or arguments as follows;

```
ls
```

The `ls` command will list the contents of a directory and in this case the command and the output would be expected to look something like the following;

```
pi@raspberrypi ~ $ ls
Desktop                python_games
```

## Options

An option (also referred to as a switch or a flag) is a single-letter code, or sometimes a single word or set of words, that modifies the behaviour of a command. When multiple single-letter options are used, all the letters are placed adjacent to each other (not separated by spaces) and can be in any order. The set of options must usually be preceded by a single hyphen, again with no intervening space.

So again using `ls` if we introduce the option `-l` we can show the total files in the directory and subdirectories, the names of the files in the current directory, their permissions, the number of subdirectories in directories listed, the size of the file, and the date of last modification.

The command we execute therefore looks like this;

```
ls -l
```

And so the command (with the `-l` option) and the output would look like the following;

```
pi@raspberrypi ~ $ ls -l
total 26
drwxr-xr-x 2 pi pi 4096 Feb 20 08:07 Desktop
drwxrwxr-x 2 pi pi 4096 Jan 27 08:34 python_games
```

Here we can see quite a radical change in the formatting and content of the returned information.

## Arguments

An argument (also called a command line argument) is a file name or other data that is provided to a command in order for the command to use it as an input.

Using `ls` again we can specify that we wish to list the contents of the `python_games` directory (which we could see when we ran `ls`) by using the name of the directory as the argument as follows;

```
ls python_games
```

The command (with the `python_games` argument) and the output would look like the following (actually I removed quite a few files to make it a bit more readable);

```
pi@raspberrypi ~ $ ls python_games
4row_arrow.png      gem4.png            pentomino.py
4row_black.png      gem5.png            pinkgirl.png
4row_board.png      gem6.png            Plain_Block.png
4row_computerwinner.png  gem7.png          princess.png
4row_humanwinner.png  gemgem.py           RedSelector.png
gem1.png            match5.wav          Wall_Block_Tall.png
gem2.png            memorypuzzle_obfuscated.py  Wood_Block_Tall.png
gem3.png            memorypuzzle.py     wormy.py
```

## Putting it all together

And as our final example we can combine our command (`ls`) with both an option (`-l`) and an argument (`python_games`) as follows;

```
ls -l python_games
```

Hopefully by this stage, the output shouldn't come as too much surprise, although again I have pruned some of the files for readability's sake;

```
pi@raspberrypi ~ $ ls -l python_games
total 1800
-rw-rw-r-- 1 pi pi 9731 Jan 27 08:34 4row_arrow.png
-rw-rw-r-- 1 pi pi 7463 Jan 27 08:34 4row_black.png
-rw-rw-r-- 1 pi pi 8666 Jan 27 08:34 4row_board.png
-rw-rw-r-- 1 pi pi 18933 Jan 27 08:34 4row_computerwinner.png
-rw-rw-r-- 1 pi pi 25412 Jan 27 08:34 4row_humanwinner.png
-rw-rw-r-- 1 pi pi 8562 Jan 27 08:34 4row_red.png
-rw-rw-r-- 1 pi pi 14661 Jan 27 08:34 tetrisc.mid
-rw-rw-r-- 1 pi pi 15759 Jan 27 08:34 tetrominoforidiots.py
-rw-rw-r-- 1 pi pi 18679 Jan 27 08:34 tetromino.py
-rw-rw-r-- 1 pi pi 9771 Jan 27 08:34 Tree_Short.png
-rw-rw-r-- 1 pi pi 11546 Jan 27 08:34 Tree_Tall.png
-rw-rw-r-- 1 pi pi 10378 Jan 27 08:34 Tree_Ugly.png
-rw-rw-r-- 1 pi pi 8443 Jan 27 08:34 Wall_Block_Tall.png
-rw-rw-r-- 1 pi pi 6011 Jan 27 08:34 Wood_Block_Tall.png
-rw-rw-r-- 1 pi pi 8118 Jan 27 08:34 wormy.py
```

# Commands

## Wildcards

An important concept to grasp in Linux is that of using wildcards. In a sporting context a wildcard is something (or someone) that can be introduced to a game as a substitute. It may not have a strictly defined value, but could fill in for a range of standard objects.

Wildcards are a feature on the Linux command line (and other places) that makes the command line far more versatile than graphical file managers. Anyone who's tried to use fancy combinations of shift / ctrl and mouse clicking while sorting in a file manager will attest to a degree of difficulty.

For example, if we have a directory with a large number files and subdirectories, and we need to move all the Python (files ending in .py) files, that have the word 'pi' somewhere in their names, from that large directory into another directory. This has the potential to be a time consuming task.

At the Linux command line that task is almost as easy to carry out as moving only one Python file, and it's simple because of the wildcards. These are special characters that allow us to select file names that match certain *patterns* of characters. This helps us select a range of matching of files by typing just a few characters, and in most cases (IMHO) it's easier than using a graphical file manager.



The process of using wildcards is also referred to as '*Globbering*'. The term '*Globbering*' originated from a program called `glob` which (long ago) would expand wildcards and apply them to sets of files (depending on the wildcards used). This was seen as a pretty useful thing. So useful that it was built into the shell. It can still be run from the command line (type `man glob` in a terminal to find out more).

Here's a list of the most commonly used wildcards :

Wildcard	Matches
<code>*</code>	zero or more characters
<code>?</code>	exactly one character
<code>[abcde]</code>	exactly one of the characters listed
<code>[a-e]</code>	exactly one character in the range specified
<code>[!abcde]</code>	any character that is not listed
<code>[!a-e]</code>	any character that is not in the range specified
<code>{pi,raspberry}</code>	exactly one entire word from the options given

As well as being able to be applied in isolation, some of the real strengths of wildcards comes when using them in combination.

## Examples

### Zero or more characters

To list all Python files with 'pi' in their names;

```
ls *pi*.py
```

### Exactly one character

To list all files in the current directory with three character file extensions;

```
ls *.???
```

### Exactly one of the characters listed

To list all files that have an extension that begins with 'p' or 'j';

```
ls *.[pj]*
```

### Exactly one character in the range specified

To list all files that have a number in the file name;

```
ls *[0-9]*
```

### Combination

To list all files that start with one of the three first or three last letters of the alphabet and end with an extension 'jpg' or 'png'.

```
ls [a-cx-z]*.{jp,pn}g
```

## Regular Expressions

Regular expressions (also called ‘regex’) are a pattern matching system that uses sequences of characters constructed according to pre-defined syntax rules to find desired strings in text. The topic of regular expressions is a book in itself and I heartily recommend further reading for those who find the need to use them in anger.



Although [wildcards](#) are used extensively in regular expressions and programming languages, bear in mind that [wildcards](#) are not part of the standard regular expression set. This means that trying to use regular expressions in a terminal will not always work.

The command [grep](#) (where the [re](#) in [grep](#) stands for [regular expression](#)) is an essential tool for any one using Linux, allowing regular expressions to be used in file searches or command outputs. Although the use of regular expressions is widespread in multiple facets of computing operations.

For example, if we wanted to search the file `dmesg` which is in the `/var/log` directory and wanted to show each line that contained the string of characters `CPU`. we would use the [grep](#) command as follows;

```
grep CPU /var/log/dmesg
```

The output from the command will appear similar to the following;

```
pi@raspberrypi ~ $ grep CPU /var/log/dmesg
[ 0.000000] Booting Linux on physical CPU 0xf00
[ 0.000000] CPU: ARMv7 Processor [410fc075] revision 5 (ARMv7), cr=10c5387d
[ 0.000000] SLUB: Hwalign=64, Order=0-3, MinObjects=0, CPUs=4, Nodes=1
[ 0.004116] CPU: Testing write buffer coherency: ok
[ 0.053503] CPU0: update cpu_capacity 1024
[ 0.053577] CPU0: thread -1, cpu 0, socket 15, mpidr 80000f00
[ 0.113791] CPU1: Booted secondary processor
[ 0.113851] CPU1: update cpu_capacity 1024
[ 0.113860] CPU1: thread -1, cpu 1, socket 15, mpidr 80000f01
[ 0.133710] CPU2: Booted secondary processor
[ 0.133746] CPU2: update cpu_capacity 1024
[ 0.133755] CPU2: thread -1, cpu 2, socket 15, mpidr 80000f02
[ 0.153750] CPU3: Booted secondary processor
[ 0.153788] CPU3: update cpu_capacity 1024
[ 0.153797] CPU3: thread -1, cpu 3, socket 15, mpidr 80000f03
[ 0.153891] Brought up 4 CPUs
[ 0.154045] CPU: All CPU(s) started in SVC mode.
[ 2.406902] ledtrig-cpu: registered to indicate activity on CPUs
```

This is a basic example that utilises a simple string to match on and shouldn't necessarily be regarded as a great use of regular expressions. However, if we wanted to limit the returned results to instances where the string was the text CPU followed by the number 0, 1, 2 or 3, we could use a regular expression with a facility that included a range of options. This is accomplished by using the square brackets `[]` with the specified range inside.

In our case we want the text CPU and it must be immediately followed by a number in the range 0 to 3. This can be designated by the regular expression `CPU[0-3]`.

Which means that our search as follows;

```
grep CPU[0-3] /var/log/dmesg
```

... will result in;

```
pi@raspberrypi ~ $ grep CPU[0-3] /var/log/dmesg
[  0.053503] CPU0: update cpu_capacity 1024
[  0.053577] CPU0: thread -1, cpu 0, socket 15, mpidr 80000f00
[  0.113791] CPU1: Booted secondary processor
[  0.113851] CPU1: update cpu_capacity 1024
[  0.113860] CPU1: thread -1, cpu 1, socket 15, mpidr 80000f01
[  0.133710] CPU2: Booted secondary processor
[  0.133746] CPU2: update cpu_capacity 1024
[  0.133755] CPU2: thread -1, cpu 2, socket 15, mpidr 80000f02
[  0.153750] CPU3: Booted secondary processor
[  0.153788] CPU3: update cpu_capacity 1024
[  0.153797] CPU3: thread -1, cpu 3, socket 15, mpidr 80000f03
```

The square brackets are 'metacharacters' and it is the use of these metacharacters that provide regular expressions with the foundation of their strength.

The following are some of the most commonly used metacharacters and a very short description of their effect (we will show examples further on);

- [ ] Match anything inside the square brackets for ONE character
- ^ (circumflex or caret) Matches only at the *beginning* of the target string (when not used inside square brackets (where it has a different meaning))
- \$ Matches only at the *end* of the target string
- (period or full-stop) Matches any single character
- ? Matches when the preceding character occurs 0 or 1 times only
- \* Matches when the preceding character occurs 0 or more times
- + Matches when the preceding character occurs 1 or more times
- ( ) Can be used to group parts of our search expression together
- | (vertical bar or pipe) Allows us to find the left hand *or* right values

## Match a defined single character with square brackets ([ ])

As demonstrated at the start of this section, the use of square brackets will allow us to match any single character. The example we used below employed the use of the dash (or minus) character as a range signifier to signify that the possible characters were 0, 1, 2, or 3.

```
grep CPU[0-3] /var/log/dmesg
```

We could also have simply put each character in the square brackets as follows;

```
grep CPU[0123] /var/log/dmesg
```

In either case it should be noted that only a single character is matched for the entries in the square brackets.

We can specify more than one range and we can also distinguish between upper case and lower case characters. Therefore the following ranges will have the corresponding results;

- [a-z] : Match any single character between a to z.
- [A-Z] : Match any single character between A to Z.
- [0-9] : Match any single character between 0 to 9.
- [a-zA-Z0-9] : Match any single character either a to z or A to Z or 0 to 9

Within square brackets we can also use the circumflex or caret character (^) to negate the characters selection. I.e. with a caret we can say search for lines with the text CPU and it must be immediately followed by a character that is not in the range 0 to 3. This is done as follows;

```
grep CPU[^0-3] /var/log/dmesg
```

Which would result in an output similar to the following;



```
pi@raspberrypi ~ $ grep CPU[^0-3] /var/log/dmesg
[ 0.000000] Booting Linux on physical CPU 0xf00
[ 0.000000] CPU: ARMv7 Processor [410fc075] revision 5 (ARMv7), cr=10c5387d
[ 0.000000] CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing
[ 0.000000] PERCPU: Embedded 11 pages/cpu @ba05d000 s12864 r8192 d24000
[ 0.000000] SLUB: HWalign=64, Order=0-3, MinObjects=0, CPUs=4, Nodes=1
[ 0.004116] CPU: Testing write buffer coherency: ok
[ 0.153891] Brought up 4 CPUs
[ 0.154045] CPU: All CPU(s) started in SVC mode.
[ 2.406902] ledtrig-cpu: registered to indicate activity on CPUs
```

Note that none of the previous lines with CPU0, CPU1, CPU2 or CPU3 have been listed.

## Match at the beginning of a string (^)

We can use the circumflex or caret character (^) to match lines of text that begin with a specific set of characters.



Just be careful and don't confuse this use of the caret with its use inside [square brackets](#) where it negates the search.

Given a text file names foo.txt with the following contents;

```
First line with something
Second line with something else
Third line still going
Fourth Line but Second last
Last line. Goodbye!
```

If we run the [grep](#) command looking for the string 'Second' as follows;

```
grep Second foo.txt
```

We should have two lines returned as below;

```
pi@raspberrypi ~ $ grep Second foo.txt
Second line with something else
Fourth Line but Second last
```

But if we use the caret character to designate that we are only looking for lines that *start* with our string as follows;

```
grep ^Second foo.txt
```

... we will get the following output where only the second line is returned;

```
pi@raspberrypi ~ $ grep Second foo.txt
Second line with something else
```

## Match at the end of a string (\$)

We can use the dollar sign character (\$) to match lines of text that finish with a specific character or set of characters.

For example, given a text file names foo.txt with the following contents;

```
First line with something
Second line with something else
Third line still going
Fourth Line but Second last
Last line. Goodbye!
```

If we use the dollar sign character to search for all lines that end in 'ing' as follows;

```
grep ing$ foo.txt
```

... we will get the following output where only the second line is returned;

```
pi@raspberrypi ~ $ grep ing$ foo.txt
First line with something
Third line still going
```



Note that the \$ sign is placed after the text that it is trying to match.

## Match any single character (.)

The . (period) character will allow us to match any single character in this position.

For example, given a text file names foo.txt with the following contents;

```
First line with something
Second line with something else
Third line still going
Fourth Line but Second last
Last line. Goodbye!
```

... if we wanted to return all lines where the characters `ing` were in the middle of the line (not at the end) we could run the following `grep` command;

```
grep ing. foo.txt
```

This would produce an output similar to the following;

```
pi@raspberrypi ~ $ grep ing. foo.txt
Second line with something else
```

While there are two other lines with `ing` in them, (the first and third lines), both of them *end* with `'ing'` and as a result there is no character after them. The only one where there is `'ing'` with a character following it is in the second line.

## Match when the preceding character occurs 0 or 1 times only (?)

It may be difficult to think of a situation where we would want to match against something that occurs 0 or 1 time, but the best example comes from the world of language. In American spelling the word `'color'` differs from the British spelling by the omission of the letter `'u'` (`'colour'`). We can write a regular expression that will match either spelling as follows;

```
colou?r
```

This way the question mark denotes that for a match to occur, the preceding character must either not be present or must occur once. The additional characters (`'colo'` and the `'r'`) are *literals* in the sense that they must be present exactly as stated. The only variable in the expression is the `'u'`.

## Match when the preceding character occurs 0 or more times (\*)

The asterisk metacharacter in regular expressions can be one of the most confusing options to use, but this is mainly because its real strength is applied when matched with other metacharacters.

For example it could be argued that a regular expression such as `q*w` will match `w`, `qw` and `qqqw`, however if we use a period and an asterisk together (`.*`) we gain a function that will match zero or more of any series of characters.

In this case we can use a regular expression such as ...

```
pa.*y
```

... to find any combination of characters that start with `pa` and end with `y` and have any number of characters (including none) in between. These would include the following;

```
pacify  
painfully  
paisley  
palmistry  
palpably  
pay
```

## Match when the preceding character occurs 1 or more times (+)

The use of the `+` character to allow one or more instances of a character is similar to that of the [asterisk](#). Where the `*` metacharacter might return the following matches from the regular expression `fe*d`;

```
fd  
fed  
feed
```

The use of `fe+d` would result in;

```
fed  
feed
```

## Group parts of a search expression together (())

Regular expressions can be combined into subgroups that can be operated on as separate entities by enclosing those entities in parenthesis. For example, if we wanted to return a match if we saw the word 'monkey' or 'banana' we would use the [or metacharacter](#) `|` (the pipe) to try to match one string or another as follows;

```
(monkey)|(banana)
```



Be aware that if we are trying to operate a regular expression in a terminal we may need to enclose it in speech-marks or escape the parenthesis (and the pipe) with the forward slash character (`\`). Sometimes this sort of thing can start to look messy depending on what use our regex is being put to.

## Find one group of values *or* another (|)

The pipe metacharacter allows us to apply a logical ‘or’ operator to our pattern matching. For example if we wanted to return a match if we saw the word ‘monkey’ or ‘banana’ we would use the words [encapsulated in parenthesis](#) and the pipe metacharacter to try to match one string or another as follows;

```
(monkey)|(banana)
```



Be aware that if we are trying to operate a regular expression in a terminal we may need to enclose it in speech-marks or escape the parenthesis (and the pipe) with the forward slash character (\). Sometimes this sort of thing can start to look messy depending on what use our regex is being put to.

## Extended Regular Expressions

In basic regular expressions the meta-characters `?`, `+`, `{`, `|`, `(`, and `)` are not regarded as special and instead we need to use the backslashed versions `\?`, `\+`, `\{`, `\|`, `\(`, and `\)`.

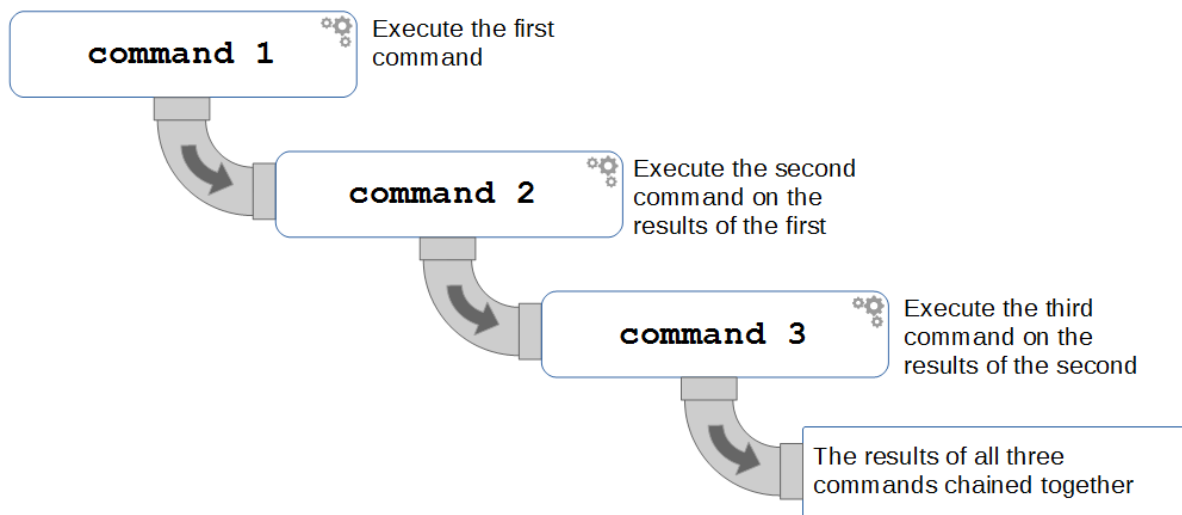
## Pipes (|)

Pipes are used in Linux to combine commands to allow them to have the output from a command feed directly into another command and so on. The idea is to use multiple commands to create a sequence of processing information from one command to another.

The commands are separated by the vertical line (|) that is normally found above the backslash key (\). This is the character that denotes the 'pipe' function. To combine three functions together with pipe symbol we would have something like the following;

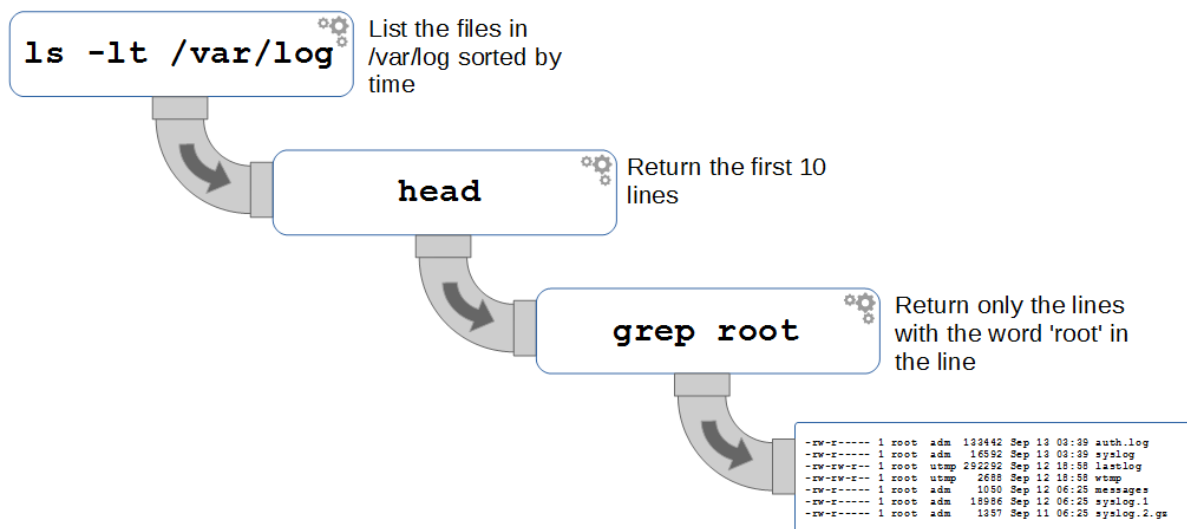
```
command1 | command2 | command3
```

As the name suggests, we can think of the pipe function as representing commands linked by a pipe where the command runs a function that is output in a pipe to flow to the second command and on to the eventual final output. To help with the visual association it might be useful to think of the connections similar to the following;



Piping one command after another

To demonstrate by example we can consider a set of commands lined as follows;



### Piping example

Here we have the `ls` command that is listing the contents of the `/var/log` directory with the listings sorted by time. This is then feeding into the `head` command that will only display the first 10 of those lines. We then feed those 10 lines into a `grep` command that filters the result to show only those lines that have the word 'root' in them.

The command as it would be run from the command line would be as follows;

```
ls -lt /var/log | head | grep root
```

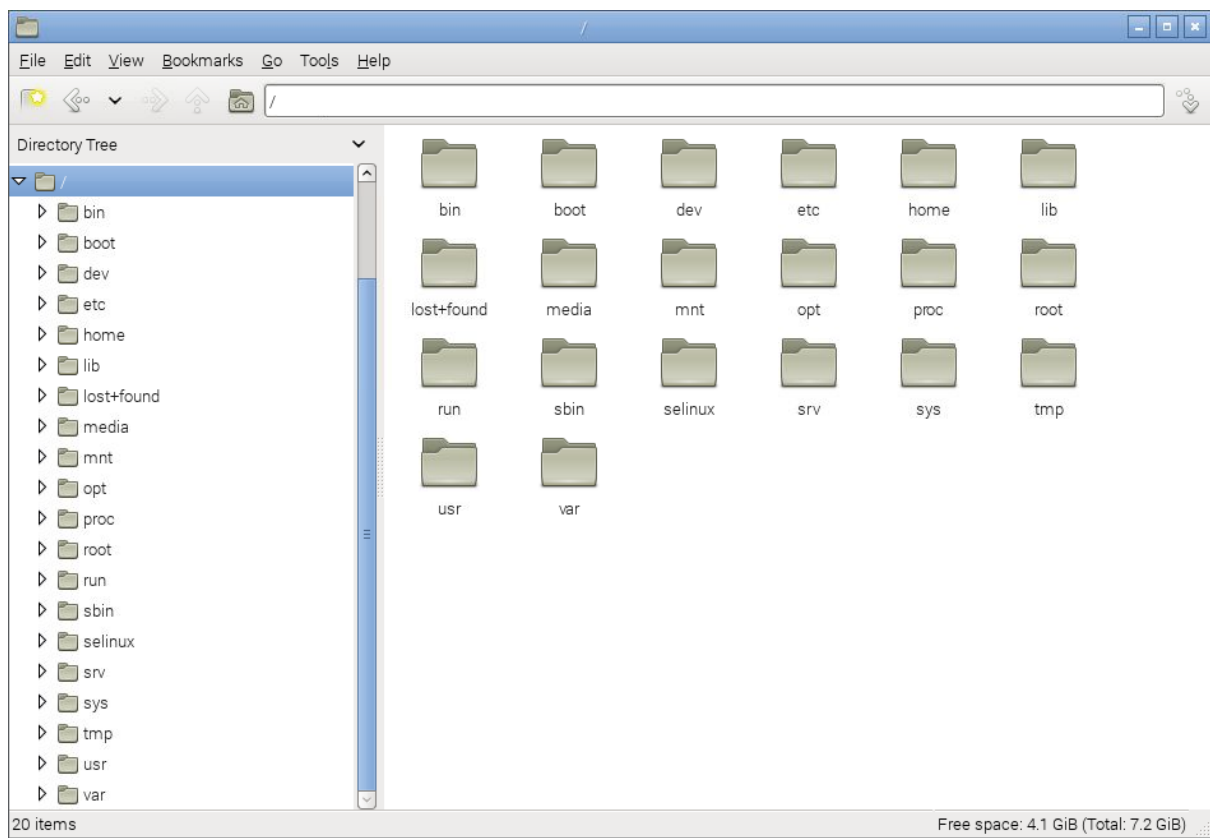
... and the output would appear something like the following;

```
pi@raspberrypi ~ $ ls -lt /var/log | head | grep root
-rw-r----- 1 root adm 133637 Sep 13 04:01 auth.log
-rw-r----- 1 root adm 16794 Sep 13 04:01 syslog
-rw-rw-r-- 1 root utmp 292292 Sep 12 18:58 lastlog
-rw-rw-r-- 1 root utmp 2688 Sep 12 18:58 wtmp
-rw-r----- 1 root adm 1050 Sep 12 06:25 messages
-rw-r----- 1 root adm 18986 Sep 12 06:25 syslog.1
-rw-r----- 1 root adm 1357 Sep 11 06:25 syslog.2.gz
```

# Linux Directory Structure

To a new user of Linux, the file structure may feel like something at best arcane and in some cases arbitrary. Of course this isn't entirely the case and in spite of some distribution specific differences, there is a fairly well laid out hierarchy of directories and files with a good reason for being where they are.

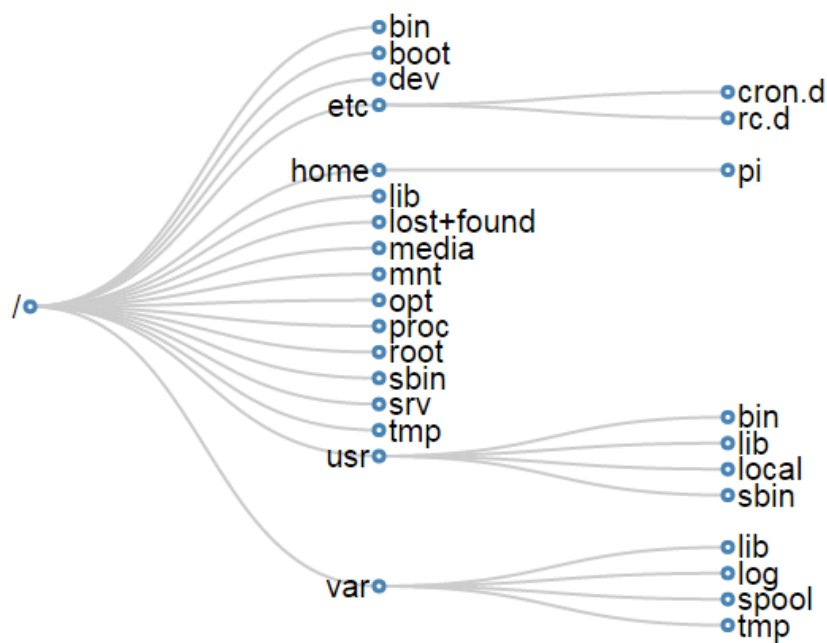
We are frequently comfortable with the concept of navigating this structure using a graphical interface similar to that shown below, but to operate effectively at the command line we need to have a working knowledge of what goes where.



## Linux Directories

The directories we are going to describe form a hierarchy similar to the following;





Directory Hierarchy

For a concise description of the directory functions check out the [cheat sheet](#). Alternatively their function and descriptions are as follows;

/

The / or 'root' directory contains all other files and directories. It is important to note that this is **not** the root users home directory (although it used to be many years ago). The root user's home directory is /root. Only the root user has write privileges for this directory.

### /bin

The /bin directory contains common essential binary executables / commands for use by all users. For example: the commands [cd](#), [cp](#), [ls](#), [ping](#), [ps](#). These are commands that may be used by both the system administrator and by users, but which are required when no other filesystems are mounted.

### /boot

The /boot directory contains the files needed to successfully start the computer during the boot process. As such the /boot directory contains information that is accessed *before* the Linux kernel begins running the programs and process that allow the operating system to function.

### /dev

The /dev directory holds [device files](#) that represent physical devices attached to the computer such as hard drives, sound devices and communication ports as well as 'logical' devices such as a

random number generator and `/dev/null` which will essentially discard any information sent to it. This directory holds a range of files that strongly reinforces the Linux precept that *Everything is a file*.

## **/etc**

The `/etc` directory contains configuration files that control the operation of programs. It also contains scripts used to startup and shutdown individual programs.

### **/etc/cron.d**

The `/etc/cron.d`, `/etc/cron.hourly`, `/etc/cron.daily`, `/etc/cron.weekly`, `/etc/cron.monthly` directories contain scripts which are executed on a regular schedule by the `crontab` process.

### **/etc/rc?.d**

The `/rc0.d`, `/rc1.d`, `/rc2.d`, `/rc3.d`, `/rc4.d`, `/rc5.d`, `/rc6.d`, `/rcS.d` directories contain the files required to control system services and configure the mode of operation (runlevel) for the computer.

## **/home**

Because Linux is an operating system that is a ‘multi-user’ environment, each user requires a space to store information specific to them. This is done via the `/home` directory. For example, the user ‘pi’ would have `/home/pi` as their home directory.

## **/lib**

The `/lib` directory contains shared library files that supports the executable files located under `/bin` and `/sbin`. It also holds the kernel modules (drivers) responsible for giving Linux a great deal of versatility to [add or remove functionality](#) as needs dictate.

## **/lost+found**

The `/lost+found` directory will contain potentially recoverable data that might be produced if the file system undergoes an improper shut-down due to a crash or power failure. The data recovered is unlikely to be complete or undamaged, but in some circumstances it may hold useful information or pointers to the reason for the improper shut-down.

## **/media**

The `/media` directory is used as a directory to temporarily mount removable devices (for example, `/media/cdrom` or `/media/cdrecorder`). This is a relatively new development for Linux and comes as a result of a degree of historical confusion over where was best to mount these types of devices (`/cdrom`, `/mnt` or `/mnt/cdrom` for example).

## **/mnt**

The `/mnt` directory is used as a generic **mount** point for filesystems or devices. Recent use of the directory is directing it towards it being used as a temporary mount point for system administrators, but there is a degree of historical variation that has resulted in different distributions doing things different ways (for example, Debian allocates `/floppy` and `/cdrom` as mount points while Redhat places them in `/mnt/floppy` and `/mnt/cdrom` respectively).

## **/opt**

The `/opt` directory is used for the installation of third party or additional **optional** software that is not part of the default installation. Any applications installed in this area should be installed in such a way that it conforms to a reasonable structure and should not install files outside the `/opt` directory.

## **/proc**

The `/proc` directory holds files that contain information about running **processes** and system resources. It can be described as a pseudo filesystem in the sense that it contains runtime system information, but not 'real' files in the normal sense of the word. For example the `/proc/cpuinfo` file which contains information about the computers cpus is listed as 0 bytes in length and yet if it is listed it will produce a description of the cpus in use.

## **/root**

The `/root` directory is the home directory of the System Administrator, or the 'root' user. This could be viewed as slightly confusing as all other users home directories are in the `/home` directory and there is already a directory referred to as the 'root' directory (`/`). However, rest assured that there is good reason for doing this (sometimes the `/home` directory could be mounted on a separate file system that has to be accessed as a remote share).

## **/sbin**

The `/sbin` directory is similar to the `/bin` directory in the sense that it holds binary executables / commands, but the ones in `/sbin` are essential to the working of the operating system and are identified as being those that the system administrator would use in maintaining the system. Examples of these commands are [fdisk](#), [shutdown](#), [ifconfig](#) and [modprobe](#).

## **/srv**

The `/srv` directory is set aside to provide a location for storing data for specific services. The rationale behind using this directory is that processes or services which require a single location and directory hierarchy for data and scripts can have a consistent placement across systems.

## **/tmp**

The `/tmp` directory is set aside as a location where programs or users that require a temporary location for storing files or data can do so on the understanding that when a system is rebooted or shut down, this location is cleared and the contents deleted.

## **/usr**

The `/usr` directory serves as a directory where user programs and data are stored and shared. This potential wide range of files and information can make the `/usr` directory fairly large and complex, so it contains several subdirectories that mirror those in the root (`/`) directory to make organisation more consistent.

### **/usr/bin**

The `/usr/bin` directory contains binary executable files for users. The distinction between `/bin` and `/usr/bin` is that `/bin` contains the essential commands required to operate the system even if no other file system is mounted and `/usr/bin` contains the programs that users will require to do normal tasks. For example; `awk`, `curl`, `php`, `python`. If you can't find a user binary under `/bin`, look under `/usr/bin`.

### **/usr/lib**

The `/usr/lib` directory is the equivalent of the `/lib` directory in that it contains shared library files that supports the executable files for users located under `/usr/bin` and `/usr/sbin`.

### **/usr/local**

The `/usr/local` directory contains users programs that are installed locally from source code. It is placed here specifically to avoid being inadvertently overwritten if the system software is upgraded.

### **/usr/sbin**

The `/usr/sbin` directory contains non-essential binary executables which are used by the system administrator. For example `cron` and `useradd`. If you can't locate a system binary in `/usr/sbin`, try `/sbin`.

## **/var**

The `/var` directory contains variable data files. These are files that are expected to grow under normal circumstances For example, log files or spool directories for printer queues.

**/var/lib**

The `/var/lib` directory holds dynamic state information that programs typically modify while they run. This can be used to preserve the state of an application between reboots or even to share state information between different instances of the same application.

**/var/log**

The `/var/log` directory holds log files from a range of programs and services. Files in `/var/log` can often grow quite large and care should be taken to ensure that the size of the directory is managed appropriately. This can be done with the `logrotate` program.

**/var/spool**

The `/var/spool` directory contains what are called ‘spool’ files that contain data stored for later processing. For example, printers which will queue print jobs in a spool file for eventual printing and then deletion when the resource (the printer) becomes available.

**/var/tmp**

The `/var/tmp` directory is a temporary store for data that needs to be held between reboots (unlike `/tmp`).

## Absolute and Relative Path Name Addressing

A strength of working with files from the command line in Linux is the flexibility and variety of options available to us as users. Some may claim that too many choices just makes it more difficult to know what is the right or wrong way to perform an action, but others (myself included) will propose that often [there is no perfect way to carry out a task, there are only perfect ways<sup>14</sup>](http://ed.ted.com/lessons/malcolm-gladwell-on-spaghetti-sauce).

What should be remembered is that the way that we choose to carry out a task should suit the task and the context. In that light there are plenty of options available in Linux and this is especially true when it comes to the process of specifying file locations.

### Pathnames

A pathname represents a text string that acts as a pointer to a file or directory. The following pathname points to the file `foo.txt` which is inside the directory `pi` which is itself inside the directory `home`

```
/home/pi/foo.txt
```

The only ‘special’ or ‘reserved’ character when using a pathname is the forward slash `/`. This character is used solely to separate directories and file names as appropriate.

Individual directories and file names are typically restricted to 255 characters with a total length of a path restricted to 4096 characters. For the sake of context, there are approximately 1500 characters up until this point in this chapter.

### Absolute Path Name Addressing

An absolute pathname will always start at the root directory (`/`). It’s an interesting quirk that people often identify the root directory as the forward slash, when it actually refers to a directory which has no name which exists on the left hand side of the `/` mark (remember that there can be no use of the `/` in directory or file names).

The following is an absolute pathname to the file `foo.txt` which is inside the directory `pi` which is itself inside the directory `home` which is in the root directory.

```
/home/pi/foo.txt
```

---

<sup>14</sup><http://ed.ted.com/lessons/malcolm-gladwell-on-spaghetti-sauce>

## Relative Path Name Addressing

A relative pathname is one where the location of the file or directory is taken *relative* to the current directory the user is in.

For example, if we were in the directory `home/pi` (the 'pi' users home directory) and we wanted to list (`ls`) the file `foo.txt` we could use absolute addressing to reach the file by typing;

```
ls /home/pi/foo.txt
```

However, since we are in the directory `/home/pi` we could use relative addressing and simply type in;

```
ls foo.txt
```

To use relative addressing effectively we need to be able to let the computer know when we want to move *up* the directory hierarchy (moving down is done by simply typing the name of the directory). Moving up to a parent directory is accomplished by using the `..` (dot dot) characters. If we imagine ourselves in the `/home/pi` directory again and want to list the file `bar.txt` which is in the `/home/raspberry` directory, we can type the following;

```
ls ../raspberry/foo.txt
```

Here the `..` characters first jumped up into the `home` directory and then went into the `raspberry` directory.

## The 'home' short-cut

The tilde (`~`) character is used as a short cut to designate the home directory for a user. Therefore assuming that the 'pi' user has the home directory of `/home/pi`, but the directory that we (as the 'pi' user) are in is the `/tmp` directory, we can list the `foo.txt` file using the following relative path and the home shortcut character;

```
ls ~/foo.txt
```

# Everything is a file in Linux

A phrase that will often come up in Linux conversation is that;

*Everything is a file*

For someone new to Linux this sounds like some sort of ‘in joke’ that is designed to scare off the unwary and it can sometimes act as a barrier to a deeper understanding of the philosophy behind the approach taken in developing Linux.

The explanation behind the statement is that Linux is designed to be a system built of a group of interacting parts and the way that those parts can work together is to communicate using a common method. That method is to use a file as a common building block and the data in a file as the communications mechanism.

The trick to understanding what ‘*Everything is a file*’ means, is to broaden our understanding of what a file can be.

## Traditional Files

The traditional concept of a file is an object with a specific name in a specific location with a particular content. For example, we might have a file named `foo.txt` which is in the directory `/home/pi/` and it could contain a couple of lines of text similar to the following;

```
This is the first line
This is the second line
```

## Directories

As unusual as it sounds a directory is also a file. The special aspect of a directory is that it is a file which contains a list of information about which files (and / or subdirectories) it contains. So when we want to list the contents of a directory using the `ls` command what is actually happening is that the operating system is getting the appropriate information from the file that represents the directory.

## System Information

However, files can also be conduits of information. The `/proc/` directory contains files that represent system and process information. If we want to determine information about the type of CPU that the computer is using, the file `cpuinfo` in the `/proc/` directory can list it. By running the command ‘`cat /proc/cpuinfo`’ we can list a wealth of information about our CPU (the following is a subset of that information by the way);



```
pi@raspberrypi ~ $ cat /proc/cpuinfo
processor       : 0
model name     : ARMv7 Processor rev 5 (v7l)
BogoMIPS      : 57.60
Features       : half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva idi\
vt vfpd32 lpae evtstrm
CPU implementer : 0x41
CPU architecture: 7
CPU variant    : 0x0
CPU part       : 0xc07
CPU revision   : 5

Hardware       : BCM2709
Revision       : a01041
Serial        : 000000002a4ea712
```

Now that might not mean a lot to us at this stage, but if we were writing a program that needed a particular type of CPU in order to run successfully it could check this file to ensure that it could operate successfully. There are a wide range of files in the `/proc/` directory that represent a great deal of information about how our system is operating.

## Devices

When we use different devices in a Linux operating system these are also represented as a file. In the `/dev/` directory we have files that represent a range of physical devices that are part of our computer. In larger computer systems with multiple disks they could be represented as `/dev/sda1` and `/dev/sda2`, so that when we wanted to perform an action such as formatting a drive we would use the command `mkfs` on the `/dev/sda1` file.

The `/dev/` directory also holds some curious files that are used as tools for generating or managing data. For example `/dev/random` is an interface to the kernel's random number device. `/dev/zero` represents a file that will constantly stream zeros (while this might sound weird, imagine a situation where you want to write over an area of disk with data to erase it). The most well known of these unusual files is probably `/dev/null`<sup>15</sup>. This will act as a 'null device' that will essentially discard any information sent to it.

---

<sup>15</sup>[https://en.wikipedia.org/wiki/Null\\_device](https://en.wikipedia.org/wiki/Null_device)

## Linux files and inodes

Having already established that [everything is a file](#) in a Linux operating system we find ourselves in the situation where files are suddenly slightly more interesting and mysterious, and it's useful to get a better understanding of how files are represented in a Linux operating system so that the context of other functions can be better understood (thinking of [links](#)).

Linux files can be best understood by thinking of them as being described by three parts;

- The file name
- An inode
- The data

### The file name

Individual file names are typically restricted to 255 characters and can be made up of almost any combination of characters. The only 'special' or 'reserved' character when assigning a file name is the forward slash (/). This character is used solely to separate directories and file names as appropriate.

The other component of a file name is an inode number. This is a number assigned by the operating system which acts as a pointer to an inode (Index Node) which is an object that can be thought of as an entry in a database that stores information specifically about that file.

We can list the inode numbers associated with files by using the `-i` option when [listing](#) files. For example;

```
ls -i python_games/
```

The output (reproduced in edited form below) shows;

```
pi@raspberrypi ~ $ ls -i python_games/
45014 4row_arrow.png          63554 inkspillresetbutton.png
49906 4row_black.png            63556 inkspillsettingsbutton.png
54215 4row_red.png              63559 match0.wav
63530 cat.png                63569 princess.png
63533 drawing.py             63488 RedSelector.png
63544 gem6.png                63576 star_title.png
63545 gem7.png                63578 tetrisb.mid
63546 gemgem.py              63579 tetrisc.mid
63553 inkspilllogo.png        63522 Wood_Block_Tall.png
63552 inkspill.py            63582 wormy.py
```

The numbers to the left of the file names are the associated inode numbers.

The file name and associated inode is in the directory that the file is contained in. Each directory (which is itself just a file) contains a table of the files it contains with the associated inode numbers.

## The inode

Each file has an associated inode (**index node**). The inode is analogous to a database entry that describes a range of attributes of the file. These attributes include;

- The unique inode number
- Access Control List (ACL)
- Extended attributes such as append only or immutability
- Disk block location
- Number of blocks
- File access, change and modification time
- File deletion time
- File generation number
- File size
- File type
- Group
- Number of links
- Owner
- Permissions
- Status flags

You may well ask what the point of having numbered inodes is. Good question. They have a range of uses including being used when there are files to be deleted with complex file names that cannot be easily reproduced at the command line or enabling [linking](#).

Interestingly, space for inodes must be assigned when an operating system is installed. Within any file-system, the maximum number of inodes, and hence the maximum number of files, is set when the file-system is created.

Therefore there are two ways in which a file-system can run out of space;

1. It can consume all the space for adding new data (i.e. run out of hard drive space), or
2. It can use up all the inodes.

Running out of inodes will bring a computer to a halt because exhaustion of the inodes prohibits the creation of additional files. Even if sufficient hard drive space exists. It is particularly easy to run out of inodes if a file-system contains a very large number of very small files.

## links

A link in a Linux file system provides a mechanism for making a connection between files. While comparisons can be made to shortcuts in Windows, this is not terribly accurate as linking in Linux has a lot more utility.

To get the full benefit from the description of links we should be passingly familiar with the previous discussion on [files and inodes](#). We should recall that a Linux file consists of three parts;

1. The filename and it's associated inode number
2. An inode that describes the attributes of the file
3. The data associated with the file.

The file name and inode number point to an inode that has all the information about the file and in turn points to the data stored on the hard drive.

Links can be identified when listing files with `ls -l` by a 1 that appears as the first character in the listing for the file and by the arrow (`->`) at the end that illustrates the link. For example by executing the following list command we can see (as only one amongst many files) that the file `/etc/vtrgb` is a link to the file `/etc/alternatives/vtrgb`.

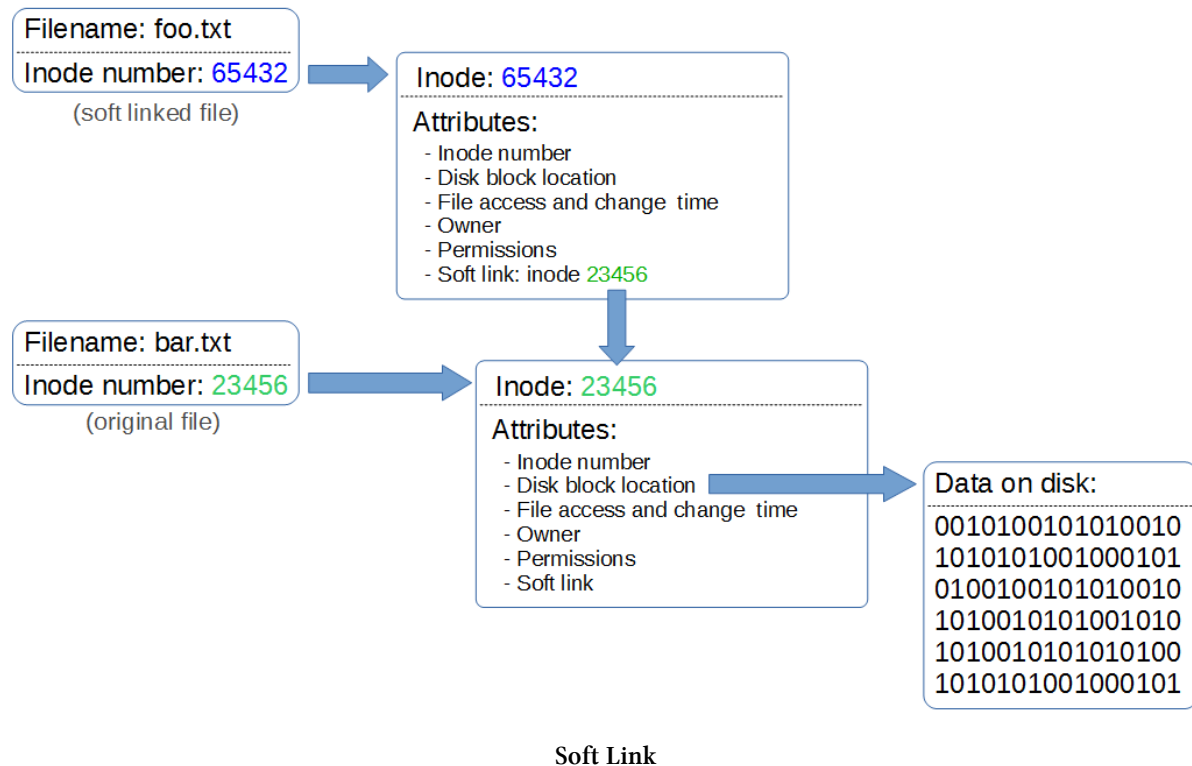
```
ls -l /etc
```

Note the leading 1 and link sign for `vtrgb` in the output.

```
drwxr-xr-x  2 root root    4096 Apr 17  2014 vim
lrwxrwxrwx  1 root root      23 Jul 10 07:45 vtrgb -> /etc/alternatives/vtrgb
-rw-r--r--  1 root root   4812 Feb  8  2014 wgetrc
drwxr-xr-x  2 root root    4096 Jul 10 08:04 wildmidi
```

## Soft Links (aka symbolic links, aka symlinks)

A soft link can also be referred to as a symbolic link, but is probably more frequently called a 'symlink'. A symlink has a file name and inode number that points to its own inode, but in the inode there is a path that redirects to *another* inode that in turn points to a data block.



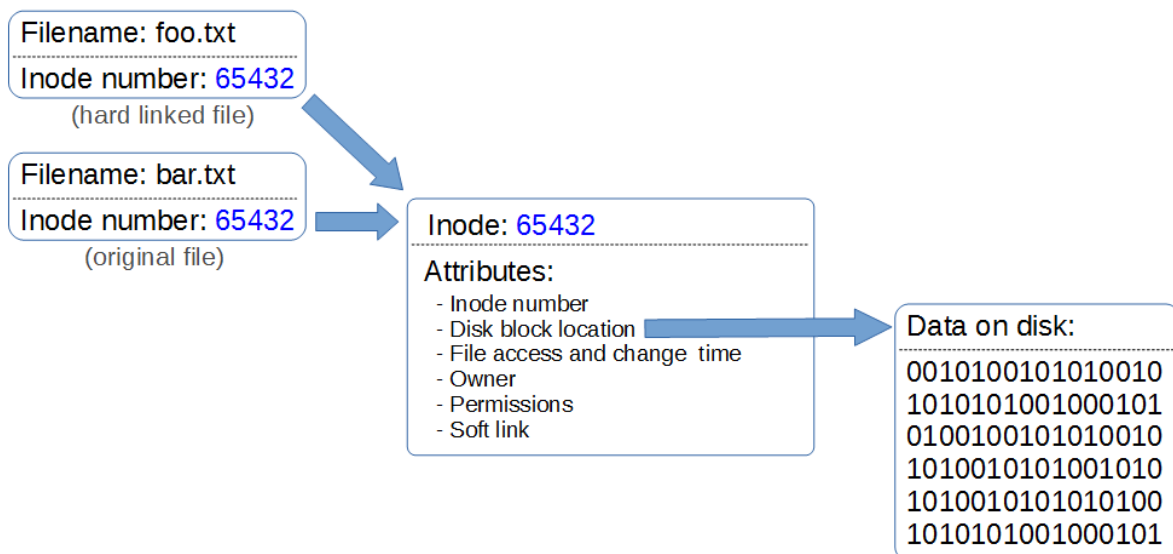
Because inodes are restricted to the partition they are created on, a symbolic link allows redirecting to a path that can cross partitions.

If we edit the linked file, the original file will be edited. If we delete the linked file, the original file will not be deleted. But if we delete the original file without deleting the link, we will be left with an orphaned link.

Soft links can link to directories *and* files but if the file is moved the link will no longer work.

## Hard Links

A hard link is one where more than one file name links to an inode.



Hard Link

This means that two file names are essentially sharing the same inode and data block, however they will behave as independent files. For example if we then delete one of the files the link is broken, but the inode and data block will remain until *all* the file names that link to the inode are deleted.

Hard links only link to a file (no directories), cannot span partitions and will still link to a file even if it is moved.

## Links Compared

### Hard links

- Will only link to a file (no directories)
- Will **not** link to a file on a different hard drive / partition
- Will link to a file even when it is moved
- Links to an inode and a physical location on the disk

### Soft links (or symbolic links or symlinks)

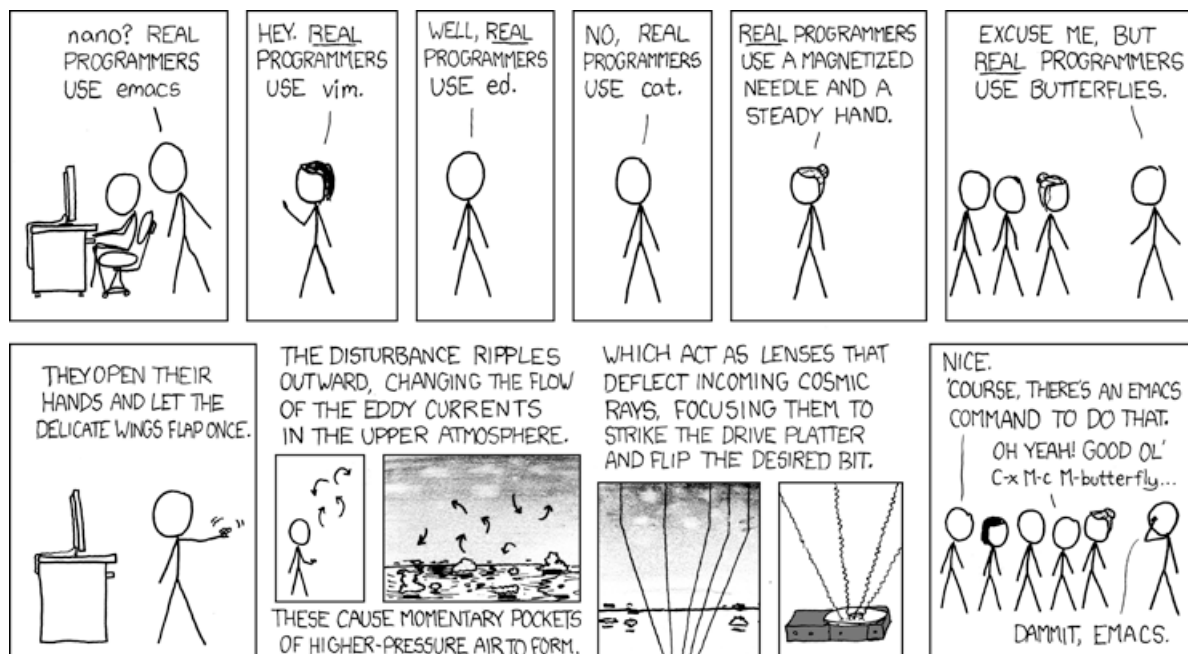
- Will link to directories or files
- Will link to a file or directory on a different hard drive / partition
- Links will remain if the original file is deleted
- Links will **not** connect to the file if it is moved
- Links connect via abstract (hence symbolic) conventions, not physical locations on the disk. They have their own inode

## File Editing

Working in Linux is an exercise in understanding the concepts that Linux uses as its foundations such as ‘Everything is a file’ and the use of [wildcards](#), [pipes](#) and the [directory structure](#).

While working at the command line there will very quickly come the realisation that there is a need to know how to edit a file. Linux being what it is, there are many ways that files can be edited.

An outstanding illustration of this is via the excellent cartoon work of the [xkcd comic strip](#)<sup>16</sup> ([Buy his stuff](#)<sup>17</sup>, it’s awesome!).



### Real Programmers

For a taste of the possible options available [Wikipedia](#)<sup>18</sup> has got our back. Inevitably where there is choice there are preferences and where there are preferences there is bias. Everyone will have a preference towards a particular editor and don’t let a particular bias influence you to go down a particular direction without considering your options. Speaking from personal experience I was encouraged to use ‘vi’ as it represented the preference of the group I was in, but because I was a late starter to the command line I struggled for the longest time to try and become familiar with it. I know I should have tried harder, but I failed. For a while I wandered in the editor wilderness trying desperately to cling to the GUI where I could use ‘gedit’ or ‘geany’ and then one day I was introduced to ‘nano’.

This has become my preference and I am therefore biased towards it. Don’t take my word for it. Try alternatives. I’ll describe ‘nano’ below, but take that as a possible path and realise that whatever editor works for you will be the right one. The trick is simply to find one that works for you.

<sup>16</sup><http://xkcd.com/378/>

<sup>17</sup><http://store.xkcd.com/>

<sup>18</sup>[https://en.wikipedia.org/wiki/List\\_of\\_text\\_editors](https://en.wikipedia.org/wiki/List_of_text_editors)

## The nano Editor

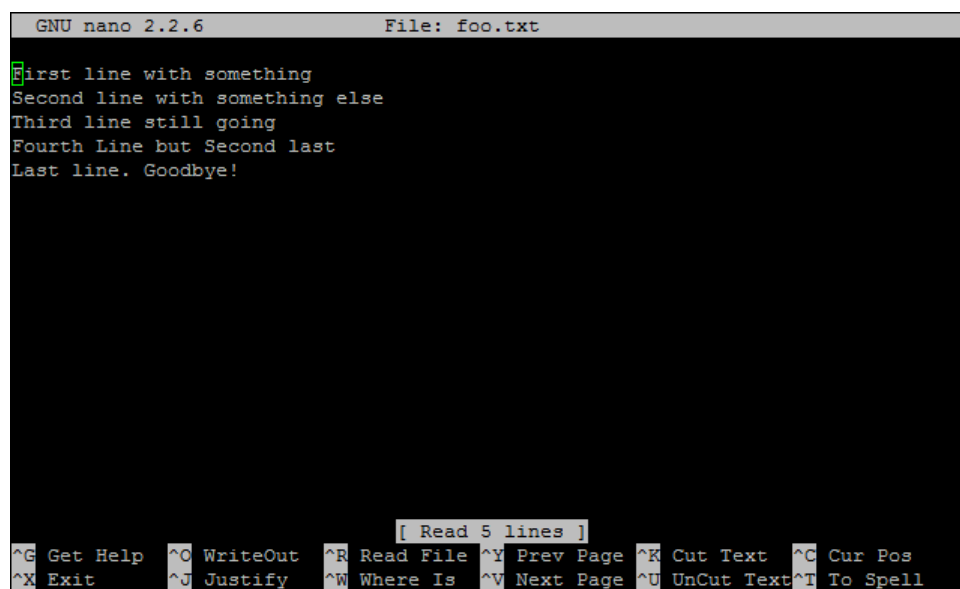
The nano editor can be started from the command line using just the command and the /path/name of the file.

```
nano foo.txt
```

If the file requires administrator permissions it can be executed with `'sudo'`.

```
sudo nano foo.txt
```

When it opens it presents us with a working space and part of the file and some common shortcuts for use at the bottom of the console;



```
GNU nano 2.2.6      File: foo.txt
First line with something
Second line with something else
Third line still going
Fourth Line but Second last
Last line. Goodbye!

[ Read 5 lines ]
^G Get Help  ^O WriteOut  ^R Read File  ^Y Prev Page  ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is   ^V Next Page  ^U UnCut Text ^T To Spell
```

nano Interface

It includes some simple syntax highlighting for common file formats;



```

GNU nano 2.2.6 File: scrape-books.php

// Regex looks for text between the span tags where there is
// <span> content followed by a newline and then <p> content
// with 'Readers in it. This has the 's' outside the encompassing
// forward slashes (/) so that the regex ignores newlines while searching.
// $regex = '/<ul class=\'book-details-list\'>\n<li class=\'detail\'>\n<span>($

$regex = '/<ul class=\'book-details-list\'>\n<li class=\'detail\'>\n<span c$
preg_match($regex,$file_string,$title);
$title[1] = str_replace(array('.', ','), ' ', $title[1]); // strips out the$
$downloads = $title[1];

$link = new PDO("mysql:host=$hostname;dbname=$dbname",$username,$password);

$stmt = $link->prepare("INSERT INTO downloads(book_name, downloaded)
VALUES(?, ?)");

$stmt->execute(array($books[$x][1], $downloads));

^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell

```

### nano Syntax Highlighting

This can be improved if desired (cue Google).

There is a swag of shortcuts to make editing easier, but the simple ones are as follows;

- CTRL-x - Exit the editor. If we are in the middle of editing a file we will be asked if we want to save our work
- CTRL-r - Read a file into our current working file. This enables us to add text from another file while working from within a new file.
- CTRL-k - Cut text.
- CTRL-u - Uncut (or Paste) text.
- CTRL-o - Save file name and continue working.
- CTRL-t - Check the spelling of our text.
- CTRL-w - Search the text.
- CTRL-a - Go to the beginning of the current working line.
- CTRL-e - Go to the end of the current working line.
- CTRL-g - Get help with nano.

## Scripting

A shell script is a file containing a series of commands. The process of scripting is the writing of those commands in the file. The shell can read this file and act on the commands as if they were typed at the keyboard.

Because there are many commands available to use we can realise the real power of computing and the reduction of repetitive tasks by automating processes.

Think of shell scripts as program commands which are chained together and which the system can execute as a scripted event. They make use of functions such as command substitution where we can invoke a command, like `date`, and use its output as part of a file-naming scheme. Scripts are programs in their own right which use programming functions such as loops, variables and if/then/else statements. The beauty of scripting is that we don't have to learn another programming language to script effectively, we simply use the commands we use at the command line.

To successfully write a script to do something we have to do three things;

1. Write a script (a file with a '.sh' extension) that uses common commands.
2. Set the permissions on the file so that it can be executed
3. Place the file somewhere that the shell can locate it when it's invoked

## Writing our Script

The script itself is just a text file. The only thing fancy about it is that it contains commands and we're going to make it executable.

We can start with a simple example that demonstrates echoing a simple message to the screen.

Starting in our home directory (/home/pi) we can use the `nano` editor to create our script by running the following command;

```
nano sayhello.sh
```

This will open the nano editor and we can type in something like the following;

```
#!/bin/bash
# Hello World Script

echo "Hello World!"
```

The first line of the script is important because it provides information to the shell about what program should be used to interpret and run the script. The line starts with a 'shebang' (`#!`) and then the path to the program (in this case `/bin/bash`).

The second line is a comment that we would place in a script to tell ourselves or others that open the file what is going on. Anything that appears after a # mark (except for on the first line with the 'shebang') will be ignored by the script and is entered to make notes for the reader.

The last line is the command we will run. In this case it only one command, but it could also be any number. The command in this case is a very simple `echo` command to print the words 'Hello World!' to the screen.

Once finished we can close and save the file (CTRL-x to close and say 'y' to save);

That's our script written.

## Make the script executable

If we list the properties of our file `sayhello.sh` with `ls` as follows;

```
ls -l sayhello.sh
```

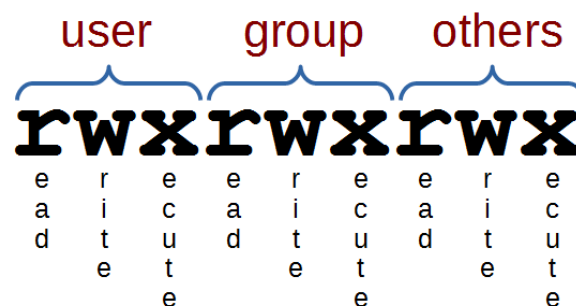
... we will see the details of the file something like this;

```
-rw-r--r-- 1 pi pi 54 Feb 17 17:18 sayhello.sh
```

Linux [permissions](#) specify what the owning user can do, what a member of the owning group can do and what other users can do with the file. For any given user, we need three bits to specify access permissions: the first to denote read (r) access, the second to denote (w) access and the third to denote execute (x) access.

We also have three levels of ownership: 'user', 'group' and 'others' so we need a triplet (three sets of three) for each, resulting in nine bits.

The following diagram shows how this grouping of permissions can be represented on a Linux system where the user, group and others had full read, write and execute permissions;



Linux permissions as rwx

The permissions of our `sayhello.sh` file tell us that the user can read and write the file, members of the group and anyone else can read the file. What we need to do is to make it executable using the `chmod` command

```
chmod 755 sayhello.sh
```

If we check again with `ls -l sayhello.sh` we should see something similar to this;

```
-rwxrwxr-x 1 pi pi 54 Feb 17 17:18 sayhello.sh
```

Here everyone has execute permissions and the 'pi' user and the 'pi' group have read and write permissions.

## Place the script somewhere that the shell can find it

Currently we're working in the 'pi' home directory where we've saved our file. This should be the easiest example of making the script accessible and as such we should be able to simply run the command directly from the command line as follows;

```
./sayhello.sh
```

The output from the command should look something like this;

```
Hello World!
```

The ./ notation in front of the script designates the path to the file being our current directory. We could have also provided the full path as follows;

```
/home/pi/sayhello.sh
```

(Since /home/pi/ is the directory that the file is currently in.)

So that works, but in the perfect world we wouldn't need to designate a path to the script in order to execute it. We can make that happen by placing the script in one of the directories in the users 'PATH'.

PATH is an *environmental variable* that tells the shell which directories to search for executable files. It should not be confused with the term 'path' which refers to a file's or directory's address in a files system

A user's PATH consists of a series of colon-separated absolute paths. When we type in a command at the command line that is *not* distributed with the operating system or which does not include its absolute path and then the shell searches through the directories in PATH, which constitute the our search path, until it finds an executable file with that name.

We can check what the paths available are by `echo`-ing the PATH variable to the screen like so;

```
echo $PATH
```

This will then report PATH to the screen something like the following

```
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

We can therefore add our script to any of those directories and it will work like a normal command. However, this will make it available to all users on the system and if we want to only make it available to ourselves as the current user we will need to add a new path to PATH.

The good news is that this is simple to do and in Debian Jessie all we need to do is to create a directory in our home directory (if we are logged in as the 'pi' user, that would be /home/pi) called `bin`. We can place any script that we want to be available to the 'pi' user in there and they will have access to it. The reason they will have access is that while it doesn't appear in PATH at the moment, when we add the `bin` directory to our home directory, there is a script called `.profile` which gets checked whenever we start our bash shell. If `.profile` sees that we have a `bin` directory in our home directory it adds it to our PATH.

So if we add the directory using `mkdir` as follows;

```
mkdir ~/bin
```

... we will be set up with a new path in PATH the next time we start our shell.

Let's not forget that we should move our file into our new directory with `mv` as follows;

```
mv ~/sayhello.sh ~/bin/
```

If we open a new terminal and log on as our user (it's 'pi' in the example) we can now go anywhere (`cd`) in the directory structure and simply run the command;

```
sayhello.sh
```

... and get the response;

```
Hello World!
```

That's the completion of our simple first example. More advanced work with loops, variables and if/then/else statements will have to be an exercise for the reader. Hopefully that's 'Just Enough' to get you started.

# Linux Commands

## File Administration

### cd

The `cd` command is used to move around in the directory structure of the file system (change directory). It is one of the fundamental commands for navigating the Linux directory structure.

`cd [options] directory` : Used to change the current directory.

For example, when we first log into the Raspberry Pi as the 'pi' user we will find ourselves in the `/home/pi` directory. If we wanted to change into the `/home` directory (go up a level) we could use the command;

```
cd /home
```

Take some time to get familiar with the concept of moving around the [directory structure](#) from the command line as it is an important skill to establish early in Linux.

### The `cd` command

The `cd` command will be one of the first commands that someone starting with Linux will use. It is used to move around in the directory structure of the file system (hence `cd` = **change directory**). It only has two options and these are seldom used. The arguments consist of pointing to the directory that we want to go to and these can be [absolute or relative paths](#).

The `cd` command can be used without options or arguments. In this case it returns us to our home directory as specified in the `/etc/passwd` file.

If we `cd` into any random directory (try `cd /var`) we can then run `cd` by itself;

```
cd
```

... and in the case of a vanilla installation of Raspbian, we will change to the `/home/pi` directory;

```
pi@raspberrypi ~ $ cd /var
pi@raspberrypi /var $ cd
pi@raspberrypi ~ $ pwd
/home/pi
```

In the example above, we changed to `/var` and then ran the `cd` command by itself and then we ran the `pwd` command which showed us that the **p**resent **w**orking **d**irectory is `/home/pi`. This is the Raspbian default home directory for the pi user.

## Options

As mentioned, there are only two options available to use with the `cd` command. This is `-P` which instructs `cd` to use the physical directory structure instead of following symbolic links and the `-L` option which forces symbolic links to be followed.

For those beginning Linux, there is little likelihood of using either of these two options in the immediate future and I suggest that you use your valuable memory to remember other Linux stuff.

## Arguments

As mentioned earlier, the default argument (if none is included) is to return to the users home directory as specified in the `/etc/passwd` file.

When specifying a directory we can do this by absolute or relative addressing. So if we started in the `/home/pi` directory, we could go the `/home` directory by executing;

```
cd /home
```

... or using relative addressing and we can use the `..` symbols to designate the parent directory;

```
cd ..
```

Once in the `/home` directory, we can change into the `/home/pi/Desktop` directory using relative addressing as follows;

```
cd pi/Desktop
```

We can also use the `-` argument to navigate to the previous directory we were in.

## Examples

Change into the root (/) directory;

```
cd /
```

## Test yourself

1. Having just changed from the /home/pi directory to the /home directory, what are the five variations of using the cd command that will take the pi user to the /home/pi directory
2. Starting in the /home/pi directory and using only relative addressing, use cd to change into the /var directory.



## chgrp

The `chgrp` command is used for changing the group ownership of a file or directory. In this sense it is very similar to `chown` command, except that it is used for changing group membership rather than that of users.

- `chgrp [options] group files` : Change group ownership of one or more files & directories

For example, the following command (which would most likely be prefixed with `sudo`) sets the group ownership for the `/srv/shared` directory. The group permissions on this directory now apply to any users in the `users` group. In most cases the `chmod` command would then be used to ensure that the permissions for the group on that directory are correct.

```
chgrp users /srv/shared
```

We can verify this using `ls -l`:

```
pi@raspberrypi ~ $ ls -l
total 4
drwxr-xr-x 2 root users 4096 Jul 18 21:24 shared
```

At this stage members of the `users` group can read and browse the contents of the `/srv/shared` directory.

If we add the group write permission using `chmod g+w` then any user who is a member of the `users` group will then be able to write into this directory.

This process can be useful for creating a directory that regular users of the system are all able to freely share data in.

## The command

The name `chgrp` is simply short-hand for 'change group'. You will probably recall that `chown` can also change group ownership, so `chgrp` is really just quicker means for changing group ownership of a file or directory if we don't need to change user ownership.

## Options

The most commonly used option with `chgrp` is the `-R` option. Just like `chown` and `chgrp`, this causes `chgrp` to change permissions recursively; that is to the target directory itself and all of its contents.

Another possible option is `--reference`. This changes the target group ownership to the same as that of the reference file. For example:

```
chgrp --reference=file1.txt file2.txt
```

## Arguments

Fortunately `chgrp` is pretty simple! Simply make sure the group name is the first argument, and any subsequent arguments are files and directories you intend to modify.

```
chgrp groupname file1 file2 ...
```

## Examples

Show some example usage and outputs (these should be different from the examples used above and should demonstrate re-world usage and combinations of arguments and options).

## Test yourself

1. What would running the command `chgrp root ~/file.txt` do? Who can access the file after running this?
2. What security issues could result from using `chgrp` to grant a group shared access to a directory? How could these be prevented?

## chmod

The `chmod` command allows us to set or modify a file's permissions. Because Linux is built as a multi-user system there are typically multiple different users with differing permissions for which files they can read / write or execute. `chmod` allows us to limit access to authorised users to do things like editing web files while general users can only read the files.

- `chmod [options] mode files` : Change access permissions of one or more files & directories

For example, the following command (which would most likely be prefixed with `sudo`) sets the permissions for the `/var/www` directory so that the user can read from, write to and change into the directory. Group owners can also read from, write to and change into the directory. All others can read from and change into the directory, but they cannot create or delete a file within it;

```
chmod 775 /var/www
```

This might allow normal users to browse web pages on a server, but prevent them from editing those pages (which is probably a good thing).

## The `chmod` command

The `chmod` command allows us to change the permissions for which user is allowed to do what (read, write or execute) to files and directories. It does this by changing the 'mode' (hence `chmod` = **change file mode**) of the file where we can make the assumption that 'mode' = permissions.

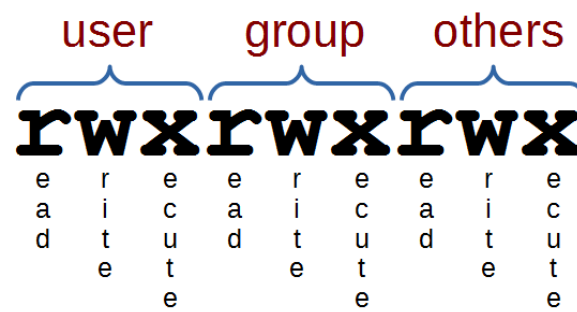
Every file on the computer has an associated set of permissions. Permissions tell the operating system what can be done with that file and by whom. There are three things you can (or can't) do with a given file:

- read it,
- write (modify) it and
- execute it.

Linux permissions specify what the owning user can do, what the members of the owning group can do and what other users can do with the file. For any given user, we need three bits to specify access permissions: the first to denote read (r) access, the second to denote (w) access and the third to denote execute (x) access.

We also have three levels of ownership: 'user', 'group' and 'others' so we need a triplet (three sets of three) for each, resulting in nine bits.

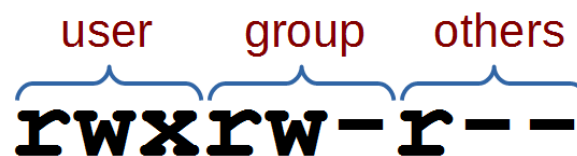
The following diagram shows how this grouping of permissions can be represented on a Linux system where the user, group and others had full read, write and execute permissions;



Linux permissions as rwx

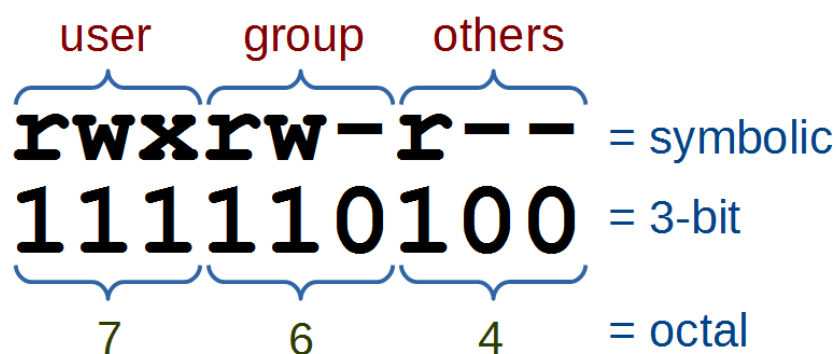
**i** Usually in Linux (when you execute the `ls -l` command) there is also another bit that leads this 9-bit pattern, but we will ignore this in the mean time.

If we had a file with more complex permissions where the user could read, write and execute, the group could read and write, but all other users could only read it would look as follows;



Slightly more complex Linux permissions

This description of permissions is workable, but we will need to be aware that the permissions are also represented as 3 bit values (where each bit is a '1' or a '0' (where a '1' is *yes you can*, or '0' is *no you can't*)) or as the equivalent octal value.



Linux permissions as symbolic, 3 bit and octal

The full range of possible values for these permission combinations is as follows;

Permission	Symbolic	3-bit	Octal
read, write and execute	rwX	111	7
read and write	rw-	110	6
read and execute	r-X	101	5
read only	r--	100	4
write and execute	-wX	011	3
write only	-w-	010	2
execute only	--X	001	1
none	---	000	0

Another interesting thing to note is that permissions take a different slant for directories.

- read determines if a user can view the directory's contents, i.e. execute `ls` in it.
- write determines if a user can create new files or delete file in the directory. (Note here that this essentially means that a user with write access to a directory can delete files in the directory even if he/she doesn't have write permissions for the file! So be careful.)
- execute determines if the user can `cd` into the directory.



It's also worth noting at this point that only the owner (or root via `sudo`) of a file may use `chmod` to alter a file's permissions.

We can check the permissions of files using the `ls -l` command which will list files in a long format as follows;

```
ls -l /tmp/foo.txt
```

This command will list the details of the file `foo.txt` that is in the `/tmp` directory as follows

```
pi@raspberrypi ~ $ ls -l /tmp
-rwxrw-r-- 1 pi pi-group 20 Jul 10 13:14 foo.txt
```

The permissions on the file, the user and the group owner can be found as follows;

file  
type

user

permissions

group

```
-rwxrw-r-- 1 pi pi-group 20 Jul 10 13:14 /tmp/foo.txt
```

File details

From this information we can see that the file's user ('pi') has permissions to read, write and execute the file. The group owner ('pi-group') can read and write to the file and all other users can read the file.

## Options

The main option that is worth remembering is the `-R` option that will **Recursively** apply permissions on the files in the specified directory and its sub-directories.

The following command will change the permissions for all the files in the `/srv/foo` directory and in all the directories that are under it;

```
chmod -R 764 /srv/foo
```

## Arguments

Simplistically (in other words it can be more complicated, but we're simplifying it) there are two main ways that `chmod` is used. In either symbolic mode where the permissions are changed using symbols associated with read, write and execute as well as symbols for the user (u), the group owner (g), others (o) and all users (a). Or in numeric mode where we use the octal values for permission combinations.

### Symbolic Mode

In symbolic mode we can change the permissions of a file with the following syntax:

- `chmod [who][op][permissions] filename`

Where `who` can be the user (u), the group owner (g) and / or others (o). The operator (op) is either + to add a permission, - to remove a permission or = to explicitly set permissions. The permissions themselves are either readable (r), writeable (w), or executable (x).

For example the following command adds executable permissions (x) to the user (u) for the file `/tmp/foo.txt`;

```
chmod u+x /tmp/foo.txt
```

This command removes writing (w) and executing (x) permissions from the group owner (g) and all others (o) for the same file;

```
chmod go-wx /tmp/foo.txt
```



Hopefully you will note that you can combine the 'who' and 'permissions' fields to allow multiple values.

Note that removing the execute permission from a directory will prevent you from being able to list its contents (although root will override this). If you accidentally remove the execute permission from a directory, you can use the `+X` argument to instruct `chmod` to only apply the execute permission to directories.

```
chmod -R u+X /home/pi/*
```

## Numeric Mode

In numeric mode we can explicitly state the permissions using the octal values, so this form of the command is fairly common.

For example, the following command will change the permissions on the file `foo.txt` so that the user can read, write and execute it, the group owner can read and write it and all others can read it;

```
chmod 764 /tmp/foo.txt
```

## Examples

To change the permissions in your home directory to remove reading and executing permissions from the group owner and all other users;

```
chmod go-rx ~
```

To make a script executable by the user;

```
chmod u+x foo.sh
```

Windows marks all files as executable by default. If you copy a file or directory from a Windows system (or even a Windows-formatted disk) to your Linux system, you should ideally strip the unnecessary execute permissions from all copied files unless you specifically need to retain it. Note that we still need it on all *directories* so that we can access their contents! Here's how we can achieve this in one command:

```
chmod -R a-x+X ~/copied_from_windows
```

This instructs `chmod` to remove the execute permission for each file and directory, and then immediately set execute again if working on a directory.

### Test yourself

1. Prevent everyone from executing the script `/tmp/foo.sh`.
2. What does the following command do? `chmod u=rwx,g=rx,o=r file.txt`
3. While changing permissions on some files you accidentally remove the execute bit from your entire home directory. What could be a quick way to fix this?



## chown

The `chown` command changes the user and/or group ownership of given files. Because Linux is built as a multi-user system there are typically multiple different users (not necessarily actual people, but daemons or other programs who may run as their own user) responsible for maintaining clear permission boundaries that separate services to prevent corruption or maintain security or privacy. This allows us to limit access to authorised users to do things like editing web files.

- `chown [options] newowner files` : Change the ownership of one or more files & directories

For example, if we want to make the user `www-data` the owner of the directory `www` (in the `/var` directory) and we want to pass the group ownership of that directory to the group `www-data` we would run the following command;

```
chown www-data:www-data /var/www
```

There is a good likelihood that we would need to prefix the command with `sudo` to run it as root depending on which user we were when we executed it.

## The `chown` command

The `chown` command changes the user and/or group ownership of given files (hence `chown` = **change owner**). It is used to help specify exactly who or what group can access certain files. There are several different options, but only one that could be deemed important enough to try and remember. There are also a number of different ways to assign ownership depending if we're trying to assign a single user and / or group permissions. For more information on modifying permissions see [chmod](#).

## Options

The main option that is worth remembering is the `-R` option that will **R**ecursively apply permissions on the files in the specified directory and its sub-directories.

The following command will change the owner to the user 'apache' for the `/var/www` directory and all the directories that are under it;

```
chown -R apache /var/www
```

## Arguments

The object that has its ownership changed can be a file or a directory and its contents.

One of the clever things about assigning permissions using `chown` is the way that user and group ownership can be applied in the same command (if desired).

If only a user name is given, that user is made the owner of each given file, and the files' group is not changed.

```
chown apache /var/www
```

If the owner is followed by a colon and a group name (with no space in between them) the group ownership of the files is changed as well. In the following example the user `apache` and the group `apache-group` are given ownership of the files in the `/var/www` directory;

```
chown apache:apache-group /var/www
```

If a colon but no group name follows the user name, that user is made the owner of the files and the group of the files is changed to that user's initial login group. So if the `apache` users initial login group was `apache-group` then the following command would accomplish the same thing as the previous example;

```
chown apache: /var/www
```

If the colon and group are given, but the owner is omitted, only the group of the files is changed.

```
chown :apache-group /var/www
```



Actually a period (.) can be used in place of a colon (:) to separate the user and the group when executing the command. Additionally the numeric User ID (UID) and / or Group ID (GID) can be used in place of the user and group names.

## Examples

To change the ownership of the file `/home/pi/foo.txt` to the UID 3456 and the group ownership to GID 4321.

```
chown 3456:4321 /home/pi/foo.txtt
```

## Test yourself

1. What has gone horribly wrong if you execute this command;

```
chown user:group / var/ftp
```

2. What command would you execute to change the group ownership of the files in the /tmp/junk directory to the group junk-owners, but not the directory itself

## cp

The `cp` command is used to copy files or directories. It is one of the basic Linux commands that allow for management of files from the command line.

- `cp [options] source destination` : Copy files and directories

For example: to make a copy of the file `foo.txt` and to call it `foo-2.txt` we would enter the following;

```
cp foo.txt foo-2.txt
```

This makes the assumption that we are in the same directory as the file `foo.txt`, but even if we weren't we could explicitly name the file with the directory structure and thereby not just make a copy of the file, but place it somewhere different;

```
cp /home/pi/foo.txt /home/pi/stuff/foo-2.txt
```

## The `cp` command

The `cp` command is used to copy files and directories (`cp` is an abbreviated form of the word **copy**). Any copies are independent of the originals and exist in their own right. While `cp` can copy directories, by default it will copy files. If we tell `cp` to copy a file when that file already exists at the destination, the old file will be overwritten. However, the owner, group and permissions for the new copied file become the same as those of the file with the same name that it replaced. The access and time of the source file and the modification time of the new file are set to the time the copying was carried out.

The normal set of [wildcards](#) and [addressing](#) options are available to make the process more flexible and extensible.

## Options

While the base functionality of the `cp` command is pretty self explanatory, there are several options which extend that functionality

- `-r` (also works with `-R`) allows us to copy directories and their contents **recursively**. In other words we can copy the directory, its file and subdirectories ad infinitum.
- `-p` preserves the mode, ownership and timestamps of copied files
- `-u` updates copied files by only copying the file when the source file is newer than the destination file or when the destination file is missing

For example, to copy the directory 'directory1' and all its contents in terms of the contained files and subdirectories into 'directory2' we can run the following command;

```
cp -r directory1/ directory2/
```

## Examples

To copy all the files from directory1 to directory2;

```
cp directory1/* directory2/
```

To copy all the files and subdirectories from directory1 to directory2;

```
cp -r directory1/* directory2/
```

To copy the files foo.txt and bar.txt to the directory foobar;

```
cp foo.txt bar.txt foobar/
```

To copy all the 'txt' files from the users home directory to a directory called backup;

```
cp ~/.txt backup/
```

## Test yourself

1. What command can be used to copy all the 'txt' files from the users home directory to the directory /home/pi/backup but only if the files that are being copied are newer than the files in the /home/pi/backup directory?
2. What characters cannot be used when naming directories or files?

## find

The `find` command is an extremely powerful and flexible tool for locating files based on matching criteria. Because there are always going to be a large range of files required to make up a system, a utility like `find` is essential to understand.

- `find [start-point] [search-criteria] [search-term] : find files`

It's important to make the conceptual leap when using `find` that finding a file or files is not just about looking for a name. A file can be described and searched for by a range of properties including name, location, permissions, user, modification time/date and size (and probably more). So while at first the `find` command might appear to be slightly arcane, the logic behind its usage is sound and once we master it, it becomes a very powerful tool.

A simple example might appear as follows;

```
find /home/pi -name missingfile.txt
```

Here we're using the start point of the 'pi' users home directory (`/home/pi`), the search criteria is the name of the file (`-name`) and the search term is a file named `missingfile.txt`.

The output reported back is as follows;

```
/home/pi/foodir/missingfile.txt
```

Here we can see that there is a file named `missingfile.txt` in the directory `/home/pi/foodir`.

The search was conducted with the start point of the directory `/home/pi` and it checked that directory and all the directories under that one, which included `/home/pi/foodir`.

## The `find` command

`find` is a command for searching that is quite 'old'. There are other commands that have been introduced to do a similar job such as `locate` and `mlocate` but their adoption has not been universal and as a result they cannot be relied upon to be available in all distributions. However, it is highly unusual to find a flavour of Linux (or Unix) that does not support `find`, so for the sake of being able to learn a command that should be easily transportable, `find` has an advantage.

As mentioned in the initial description above, it is important to make the conceptual leap when using `find` that the process of searching is about combining three things;

1. The start point for the search
2. The type of criteria or property that we are going to be evaluating on our search
3. The specific value of the criteria we want to search against.

Then once we are comfortable about how the search against the criteria works we can chain them together to make our searches more refined.

Numeric arguments below can be specified as;

- $+n$  : for greater than  $n$ ,
- $-n$  : for less than  $n$ ,
- $n$  : for exactly  $n$ .

We can also employ [wildcards](#) to assist in matching names.

- `-name pattern` : the file name which matches the pattern *pattern*
- `-iname pattern` : Like `-name`, but the match is case insensitive
- `-mmin n` : the file's data was last modified  $n$  minutes ago
- `-mtime n` : the file's data was last modified  $n * 24$  hours ago
- `-newer file` : the file was modified more recently than *file*
- `-amin n` : the file was last accessed  $n$  minutes ago.
- `-atime n` : the file was last accessed  $n * 24$  hours ago
- `-user uname` : the file is owned by user *uname*
- `-group gname` : the file belongs to group *gname*
- `-executable` : matches files which are executable and directories which are searchable
- `-type c` : the file is of type *c*:
  - `b` block (buffered) special
  - `c` character (unbuffered) special
  - `d` directory
  - `p` named pipe (FIFO)
  - `f` regular file
  - `l` symbolic link
  - `s` socket
- `-size n[cwbkMG]` : File uses  $n$  units of space (rounding up). The following suffixes can be used:
  - `b` for 512-byte blocks (this is the default if no suffix is used)
  - `c` for bytes
  - `w` for two-byte words
  - `k` for Kilobytes (units of 1024 bytes)
  - `M` for Megabytes (units of 1048576 bytes)
  - `G` for Gigabytes (units of 1073741824 bytes)
- `-perm mode`: The files have a specific set of permissions with *mode* different depending on how we want to access them
  - `mode` File's permission bits are **exactly** *mode*
  - `-mode` **All** of the permission bits *mode* are set for the file.
  - `/mode` **Any** of the permission bits *mode* are set for the file.

## Examples of use

### Case Insensitive (-iname)

To search by name and to make the search case insensitive, we use `-iname`.

```
find /home/pi -iname missingfile.txt
```

The output reported back is as follows;

```
/home/pi/foodir/missingfile.txt  
/home/pi/foodir/MissingFile.txt
```

### Modified Minutes or Hours Ago (-mmin or -mtime)

To search for files that have been modified some number of minutes ago we would use `-mmin`.

```
find /home/pi -mmin -10
```

The output reported back is as follows;

```
/home/pi/foodir  
/home/pi/foodir/MissingFile.txt
```

The file `MissingFile.txt` and the directory it is in `/home/pi/foodir` have been modified less than 10 minutes ago.



Remember that `-10` means files less than 10 minutes ago, `+10` would mean more than 10 minutes ago and `10` would mean 10 minutes ago.

If we were using `-mtime`, `-10` would mean less than 10 days ago and `+10` would mean more than 10 days ago.

### More Recently Modified than Another File (-newer)

We can search for files have been modified more recently than a specified file using `-newer`.

I created the file `missingfile.txt` a couple of days ago, and since that time I have made some changes. So just what files have changed in my home directory since I modified that file?

```
find /home/pi -newer missingfile.txt
```

The output reported back is as follows;



```
/home/pi
/home/pi/.xsession-errors
/home/pi/.bash_history
/home/pi/.vnc
/home/pi/.vnc/raspberrypi:1.log
/home/pi/.vnc/raspberrypi:1.pid
/home/pi/.Xauthority
/home/pi/foodir
/home/pi/foodir/MissingFile.txt
```

We can see some files that have been looking after the user experience of the 'pi' user and we can also see that the file `MissingFile.txt` is newer than `missingfile.txt`.

### By User or Group Name (-user or -group)

We can search by user using the `-user` criteria. In the following example we will look for files that belong to the 'root' user in the directory `/home/pi`.

```
find /home/pi -user root
```

The output to the screen is;

```
/home/pi/foodir/rootsfile.txt
```

If we check the files in the `/home/pi/foodir` directory using `ls -l` we see the following;

```
-rw-r--r-- 1 pi    pi    0 Jan 31 10:08 anotherfoo.txt
-rw-r--r-- 1 pi    pi    0 Feb 20 19:22 missingfile.txt
-rwxrw-r-- 1 pi    pi    0 Feb 21 13:28 MissingFile.txt
-rwxr-xr-x 1 pi    pi    0 Feb 21 13:29 runme.sh
-rw-r--r-- 1 root  root  0 Feb 21 20:45 rootsfile.txt
```

Sure enough, there's the file `rootsfile.txt` which is owned by root we can perform the same search and get the same result (at least in *this* directory using the `-group` search criteria.

### Find Executable Files (-executable)

We can search for executable files using the `-executable` criteria. In this case we are looking for files that have been set as executable in their permissions;

```
find /home/pi/foodir -executable
```

The output to the screen in this case is;

```
/home/pi/foodir/runme.sh
/home/pi/foodir/MissingFile.txt
```

If we check the files in the `/home/pi/foodir` directory using `ls -l` we see the following;

```
-rw-r--r-- 1 pi    pi    0 Jan 31 10:08 anotherfoo.txt
-rw-r--r-- 1 pi    pi    0 Feb 20 19:22 missingfile.txt
-rwxrw-r-- 1 pi    pi    0 Feb 21 13:28 MissingFile.txt
-rwxr-xr-x 1 pi    pi    0 Feb 21 13:29 runme.sh
-rw-r--r-- 1 root  root  0 Feb 21 20:45 rootsfile.txt
```

Sure enough the files `runme.sh` and `MissingFile.txt` are executable.

### By the Type of File (-type)

Individual files can be one of a fairly wide range of types. We can use the type as our search criteria using `-type`. The ranges of types that can be searched for includes;

- `b` block (buffered) special
- `c` character (unbuffered) special
- `d` directory
- `p` named pipe (FIFO)
- `f` regular file
- `l` symbolic link
- `s` socket

So if we search using the directory type `d` as follows;

```
find /home/pi/ -type d
```

All the directories are output to the screen as follows;

```
/home/pi/
/home/pi/.vnc
/home/pi/foodir
/home/pi/.ssh
/home/pi/bin
```

### By the Size of the File (-size)

Searching for files by size with `-size` involves including a suffix to the value to indicate the type of unit involved. The range of units are;

- `b` for 512-byte blocks (this is the default if no suffix is used)

- `c` for bytes
- `w` for two-byte words
- `k` for Kilobytes (units of 1024 bytes)
- `M` for Megabytes (units of 1048576 bytes)
- `G` for Gigabytes (units of 1073741824 bytes)

If we wanted to search for files larger than 1 Megabyte we would use;

```
find /home/pi/ -size +1M
```

Those files would be output to the screen as follows;

```
/home/pi/master.zip.2  
/home/pi/alpha.zip  
/home/pi/master.zip.1  
/home/pi/master.zip.4  
/home/pi/master.zip.3
```

A very scruffy check to see what size those files are with `ls -l *zip*` shows us the following;

```
-rw-r--r-- 1 pi pi 3204673 Feb  8 07:35 alpha.zip  
-rw-r--r-- 1 pi pi    116 Feb 10 17:14 master.zip  
-rw-r--r-- 1 pi pi 3204673 Feb  7 09:10 master.zip.1  
-rw-r--r-- 1 pi pi 3204673 Feb  8 07:29 master.zip.2  
-rw-r--r-- 1 pi pi 3204673 Feb  8 07:39 master.zip.3  
-rw-r--r-- 1 pi pi 3204673 Feb 10 17:20 master.zip.4  
-rw-r--r-- 1 pi pi 295696 Feb 10 17:47 test2.zip  
-rw-r--r-- 1 pi pi    116 Feb 10 17:19 test.zip
```

Sure enough, the files returned by the search were above 1 Megabyte and there were others there of lesser size that did not get returned.

### By the permissions of the File (-perm)

Searching for files by permission using `-perm` can be accomplished in one of three ways

- `mode` File's permission bits are **exactly** *mode*
- `-mode` **All** of the permission bits *mode* are set for the file.
- `/mode` **Any** of the permission bits *mode* are set for the file.

Like `chmod`, `find` understands permissions specified in either symbolic or numeric (octal).

To repeat our search from earlier where we looked for executable files we could run the following using symbolic mode and looking for *any* of the permission bits set to executable;

```
find /home/pi/foodir -perm /a=x
```

The output from this might look like;

```
/home/pi/foodir/runme.sh
```

Perhaps we want to locate any file with the permissions 764 in octal (rwxrw-r-- in symbolic):

```
find /home/pi -perm 0764
```

Which in this case shows us;

```
/home/pi/foodir/MissingFile.txt
```

Since `ls -ls /home/pi/foodir/MissingFile.txt` shows us the following

```
0 -rwxrw-r-- 1 pi pi 0 Feb 21 13:28 MissingFile.txt
```

### Chaining Search Criteria

The `find` command starts to really flex its muscles when search criteria are combined. For example we could search for all files belonging to the user 'pi' and which were above 1 Megabyte as follows;

```
find /home/pi -user pi -size +1M
```

Or perhaps narrowing down a size window for a files above 500 Kilobytes and below 1 Megabyte;

```
find /home/pi -size +500k -size -1M
```

### Avoiding the 'Permission Denied' Messages

When running a search across a larger part of the file system (for example starting in the root directory), there will be a large number of directories that the user conducting the search may not have permissions to view. As a result the feedback from the `find` command will include a large number of messages informing us that we were denied permission to search in those directories. A resolution to this is to use the `-print` criteria to redirect any error messages to 'dev/null' (which is kind of like saying 'go away').

The following is a search that will return any files larger than 1 Megabyte from the entire file system (starting at root (/)). And when it strikes an error (like permission denied) it prints that error to /dev/null;

```
find / -user pi -size +1M -print 2>/dev/null
```

## Test yourself

1. Why would we use `find` in preference to alternative Linux search commands?
2. Craft a search to look for files in the current users home directory (without using absolute addressing) and looking for files that were created more than 20 minutes ago but less than 36 hours ago.
3. If we don't specify a search point, what is the default?

## gzip

gzip is a tool for compressing and decompressing a file (or multiple files) using the gzip format. By itself gzip can only compress one file at a time, so if multiple files need to be compressed then `tar` is normally used to combine the files into a single archive file first. That archive file can then be compress with gzip. It can be a useful action to compress files when transferring them or storing them to save bandwidth or space.

- `gzip [options] filename` : compress or expand files

In its simplest use case we only need to use the gzip command and the file name as follows;

```
gzip files.txt
```

By default this will take the file `files.txt` and compress it and replace the original file with one named `file.txt.gz`.

We can check the compression by comparing the output from the command `ls -l files.txt` from before compressing.....

```
-rw-r--r--  1 pi pi      398 Feb  8 08:04 files.txt
```

With that from after compression;

```
-rw-r--r--  1 pi pi     146 Feb  8 08:04 files.txt.gz
```

Before the file size was 398 bytes and afterwards it was 146 bytes.



The amount of compression will depend on a range of factors, not the least of which will be the type of file that is being compressed. Some files are compressed by design (think \*.jpg or \*.png files) and these will not have the same type of compression ratio.

Compression is always performed, even if the compressed file is slightly larger than the original. The worst case theoretical expansion is a few extra bytes but in reality the actual number of used disk blocks almost never increases.

## The gzip command

The `gzip` command reduces the size of the named files using the Lempel-Ziv coding algorithm (this is the same algorithm used in zip and PKZIP). When the compression takes place, each file is replaced by one with the same name and the extension `.gz`, while keeping the same ownership modes, access and modification times.

If the compressed file name is too long for its file system, `gzip` will attempt to truncate the parts of the file name longer than 3 characters.

Compressed files can be restored to their original form using `gzip -d` or `gunzip`. If the original name saved in the compressed file is not suitable for the file system it is being decompressed into, a new name is constructed from the original one to make it legal.

`gunzip` takes a list of files on its command line and replaces each file whose name ends with `.gz`, `-gz`, `.z`, `-z`, `_z` or `.Z`. It also recognizes the special extensions `.tgz` and `.taz` as short-hand for `.tar.gz` and `.tar.Z` respectively. When compressing, `gzip` uses the `.tgz` extension if necessary instead of truncating a file with a `.tar` extension.

## Options

The options that are the most common and arguably the most useful include;

- `-d` : decompresses a compressed file.
- `-l` : lists the details of a decompression process

## Decompress

To decompress a gzipped file we can use the `gzip` command with the `-d` option as follows;

```
gzip -d files.txt.gz
```

Or we can use the `gunzip` command which is the equivalent of `gzip -d` like this;

```
gunzip files.txt.gz
```

## List Decompression Details

To get a feel for how well the compression has worked we can use the `-l` option when decompressing and it will print out the compressed size, the uncompressed size, the compression ratio and the name of the decompressed file. The command would look like this;

```
gzip -dl files.txt.gz
```

And the output would look something like this;

compressed	uncompressed	ratio	uncompressed_name
146	398	70.4%	files.txt

## Test yourself

1. Will the command `gzip file1.txt file2.txt` compress both of these files into one? If “yes”, what will it be called? If “no”, what will the result be?



## ln

The `ln` command is used to make [links](#) between files. This allows us to have a single file or directory referred to by different names.

- `ln [options] originalfile linkfile` : Create links between files or directories
- `ln [options] originalfile` : Create a link to the original file in the current directory

The `ln` command will create a hard link by default and a soft link (symlink) by using the `-s` option.

For example to create a hard link in the folder `/home/pi/foobar/` to the file `foo.txt` which is in `/home/pi/` we could use;

```
ln /home/pi/foo.txt /home/pi/foobar/
```

The target directory for the new link must exist for the command to be successful.

Once the link is created if we were to edit the file from either location it will be the same file that is being changed.

## The `ln` command

The `ln` command is used to make [links](#) between files (hence `ln` = **link**). By default the links will be 'hard' meaning that the links point to the same [inode](#) and therefore are pointing to the same data on the hard drive. By using the `-s` option a soft link (also known as a symbolic link or a symlink) can be created. A soft link has it's own inode and can span partitions.

This allows us to have a single file or directory referred to by different names.

### Hard links

- Will only link to a file (no directories)
- Will **not** link to a file on a different hard drive / partition
- Will link to a file even when it is moved
- Links to an inode and a physical location on the disk

### Soft links (or symbolic links or symlinks)

- Will link to directories or files
- Will link to a file or directory on a different hard drive / partition
- Links will remain if the original file is deleted
- Links will **not** connect to the file if it is moved
- Links connect via abstract (hence symbolic) conventions, not physical locations on the disk. They have their own inode

## Options

There are several different options, but the main one that will be used the most is the `-s` option to create a soft link.

If we repeat our example from earlier where we wanted to create a link in the folder `/home/pi/-foobar/` to the file `foo.txt` which is in `/home/pi/`, by including the `-s` option we can make the link soft instead;

```
ln -s /home/pi/foo.txt /home/pi/foobar/
```

If we then list the contents of the `foobar` directory we will see the following;

```
pi@raspberrypi ~/foobar $ ls foobar/ -l
total 0
lrwxrwxrwx 1 pi pi 16 Aug 16 01:30 foo.txt -> /home/pi/foo.txt
```

The read/write/execute descriptors for the permissions are prefixed by an `l` (for link) and there is a stylised arrow (`->`) linking the files.

## Test yourself

1. Can the 'root' user create a hard link for a directory?
2. What command would be used to create links for multiple 'txt' files at the same time?

## ls

The `ls` command lists the contents of a directory and can show the properties of those objects it lists. It is one of the fundamental commands for knowing what files are where and the properties of those files.

- `ls [options] directory` : List the files in a particular directory

For example: If we execute the `ls` command with the `-l` option to show the properties of the listings in long format and with the argument `/var` so that it lists the content of the `/var` directory...

```
ls -l /var
```

... we could see the following;

```
pi@raspberrypi ~ $ ls -l /var
total 102440
drwxr-xr-x  2 root    root      4096 Mar  7 06:25 backups
drwxr-xr-x 12 root    root      4096 Feb 20 08:33 cache
drwxr-xr-x 43 root    root      4096 Feb 20 08:33 lib
drwxrwsr-x  2 root    uucp      4096 Jan 11 00:02 local
lrwxrwxrwx  1 root    root        9 Feb 15 11:23 lock -> /run/lock
drwxr-xr-x 11 root    root      4096 Jul  7 06:25 log
drwxrwsr-x  2 root    mail      4096 Feb 15 11:23 mail
drwxr-xr-x  2 root    root      4096 Feb 15 11:23 opt
lrwxrwxrwx  1 root    root        4 Feb 15 11:23 run -> /run
drwxr-xr-x  4 root    root      4096 Feb 15 11:26 spool
-rw-----  1 root    root    104857600 Feb 16 14:03 swap
drwxrwxrwt  2 root    root      4096 Jan 11 00:02 tmp
drwxrwxr-x  2 www-data www-data 4096 Feb 20 08:21 www
```



### What's the information in the long list format?

- 1st column will give detailed information regarding file permission,
- 2nd column will tell you about the number of links to the file,
- 3rd and 4th columns are associated with owner and group of the file,
- 5th column will be displaying the size of the file in bytes,
- 6th column will display the recent time and date at which the file was modified,
- and the last and 7th column is the actual file/directory/link name.

## The `ls` command

The `ls` command will be one of the first commands that someone starting with Linux will use. It is used to list the contents of a directory (hence `ls` = list). It has a large number of options for displaying listings and their properties in different ways. The arguments used are normally the name of the directory or file that we want to show the contents of.

By default the `ls` command will show the contents of the current directory that the user is in and just the names of the files that it sees in the directory. So if we execute the `ls` command on its own from the pi users home directory (where we would be after booting up the Raspberry Pi), this is the command we would use;

```
ls
```

... and we should see the following;

```
pi@raspberrypi ~ $ ls
Desktop  python_games
```

This shows two directories (`Desktop` and `python_games`) that are in pi's home directory, but there are no details about the directories themselves. To get more information we need to include some options.

## Options

There are a very large number of options available to use with the `ls` command. For a full listing type `man ls` on the command line. Some of the most commonly used are;

- `-l` gives us a long listing (as explained above)
- `-a` shows us aLL the files in the directory, including hidden files
- `-s` shows us the size of the files (in blocks, not bytes)
- `-h` shows the size in "human readable format" (ie: 4K, 16M, 1G etc). (must be used in conjunction with the `-s` option).
- `-S` sorts by file Size
- `-t` sorts by modification time
- `-r` reverses order while sorting

A useful combination of options could be a long listing (`-l`) that shows all (`-a`) the files with the file size being reported in human readable (`-h`) block size (`-s`).

```
ls -lash
```

... will produce something like the following;

```
pi@raspberrypi ~ $ ls -lash
```

```
total 84K
```

```
4.0K drwxr-xr-x 13 pi pi 4.0K May 7 11:46 .
4.0K drwxr-xr-x 3 root root 4.0K May 7 10:20 ..
4.0K -rw-r--r-- 1 pi pi 69 May 7 11:46 .asoundrc
4.0K -rw----- 1 pi pi 854 Jul 8 12:55 .bash_history
4.0K -rw-r--r-- 1 pi pi 3.2K May 7 10:20 .bashrc
4.0K drwxr-xr-x 4 pi pi 4.0K May 7 11:46 .cache
4.0K drwxr-xr-x 7 pi pi 4.0K May 7 11:46 .config
4.0K drwxr-xr-x 2 pi pi 4.0K May 7 11:46 Desktop
4.0K drwxr-xr-x 2 pi pi 4.0K May 7 11:46 .fontconfig
4.0K drwxr-xr-x 2 pi pi 4.0K May 7 11:46 .gststreamer-0.10
4.0K drwx----- 3 pi pi 4.0K May 7 11:46 .local
4.0K -rw-r--r-- 1 pi pi 675 May 7 10:20 .profile
4.0K drwxrwxr-x 2 pi pi 4.0K Jan 27 21:34 python_games
4.0K drwxr-xr-x 3 pi pi 4.0K May 7 11:46 .themes
```



The size of the reported files when using the human readable option are designated by the following respective letters;

- K = Kilobyte
- M = Megabyte
- G = Gigabyte
- T = Terabyte
- P = Petabyte
- E = Exabyte
- Z = Zettabyte
- Y = Yottabyte

## Arguments

The default argument (if none is included) is to list the contents of the directory that the user is currently in. Otherwise we can specify the directory to list. This might seem like a simple task, but there are a few tricks that can make using *ls* *really* versatile.

The simplest example of using a specific directory for an argument is to specify the location with the full address. For example, if we wanted to list the contents of the `/var` directory (and it doesn't matter which directory we run this command from) we simply type;

```
ls /var
```

... will produce the following;

```
pi@raspberrypi ~ $ ls /var
backups  cache  lib  local  lock  log  mail  opt  run  spool  swap  tmp  www
```

We can also use some of the relative addressing characters to shortcut our listing. We can list the home directory by using the tilde (ls ~) and the parent directory by using two full stops (ls ..).

The asterisk (\*) can be used as a **wildcard** to list files with similar names. E.g. to list all the png file in a directory we can use ls \*.png.

If we just want to know the details of a specific file we can use its name explicitly. For example if we wanted to know the details of the swap file in /var we would use the following command;

```
ls -l /var/swap
```

... which will produce the following;

```
pi@raspberrypi ~ $ ls -l /var/swap
-rw----- 1 root root 104857600 May  7 11:29 /var/swap
```

## Examples

List all the configuration (.conf) files in the /etc directory;

```
ls /etc/*.conf
```

... which will produce the following;

```
pi@raspberrypi ~ $ ls /etc/*.conf
/etc/adduser.conf      /etc/host.conf        /etc/ntp.conf
/etc/ca-certificates.conf /etc/imapd.conf      /etc/pam.conf
/etc/debconf.conf      /etc/insserv.conf     /etc/resolv.conf
/etc/deluser.conf      /etc/ld.so.conf       /etc/resolvconf.conf
/etc/dhcpd.conf        /etc/libaudit.conf    /etc/rsyslog.conf
/etc/fuse.conf          /etc/logrotate.conf   /etc/sysctl.conf
/etc/gai.conf           /etc/mke2fs.conf      /etc/ts.conf
/etc/gssapi_mech.conf   /etc/nsswitch.conf    /etc/ucf.conf
```

## Test yourself

1. List files sorted by the time they were last modified in reverse order (most recently modified files last).

2. Find the permissions of the `/etc/passwd` file.
3. List all the files in the `/etc` directory in long format and sorted by extension. (you'll have to read the `ls` man page).
4. List the files in the `/etc` directory from largest to the smallest and store it into `/tmp/my-files.txt` file (not easy for a beginner and might require some Googling).

## mkdir

The `mkdir` command creates directories. It is one of the fundamental file management commands in Linux.

- `mkdir [options] directory` : Create a directory

The `mkdir` command is used to create directories or folders. It's a fairly simple command with a few options for additional functionality to allow paths and permissions to be set when creating. At its simplest, the following command will create a directory called `foobar` in the current working directory;

```
mkdir foobar
```

We can check on the creation by listing the files using `ls` with the `-l` option as follows;

```
ls -l
```

Which should show something like the following;

```
pi@raspberrypi ~ $ ls -l
total 68
drwxr-xr-x 2 pi pi 4096 Feb 20 08:07 Desktop
drwxr-xr-x 2 pi pi 4096 Aug 16 03:47 foobar
-rw-r--r-- 1 pi pi    5 Aug 16 01:27 foo.txt
drwxrwxr-x 2 pi pi 4096 Jan 27  2015 python_games
```

The read/write/execute descriptors for the permissions of the directories are prefixed by a `d` (for **d**irectory) and in some terminals the colour of the text showing the directory will be different from that of other types of files (let's not forget that while we call a directory a directory because of its function, [it is really a type of file](#)).

## The 'mkdir' command

The `mkdir` command is used to create directories which are used as containers for files and subdirectories. Directories created by `mkdir` are automatically created with two hidden directories, one representing the directory just created (and shown as a single dot `.`) and the other representing its parent directory (and represented by two dots `..`). These hidden directories can be seen by using the `ls` command with the `-a` option (`ls -a`).

Directories can be removed with the `rm` and `rmdir` commands.



## Options

The `mkdir` command has a small number of options and the two most likely to be used on anything approaching a regular basis would be;

- `-p` creates the specified **p**arent directories for a new directory if they do not already exist
- `-m` controls the permission **m**ode of new directories (in the same way as `chmod`)

For example to create the nested directories `foo/bar/foobar` in the current working directory we would execute the following;

```
mkdir -p foo/bar/foobar
```

Without the `-p` option we would need to create each layer separately.

To create a directory with a specific set of read, write and execute permissions we can use the `-m` option with the same mode arguments as used with the `chmod` command. For example the following command will create the `foobar` directory where the owner has read and write permissions, the group has read permission and other users have no permissions, the following would be used;

```
mkdir -m 640 foobar
```

If we subsequently check those permissions with `ls -l` we will see the following;

```
pi@raspberrypi ~ $ ls -l
total 68
drw-r----- 2 pi pi 4096 Aug 16 05:36 foobar
```

## Arguments

The normal set of [addressing](#) options are available to make the process of creating the right directories more flexible and extensible.

## Test yourself

1. How would we use the `mkdir` command to create multiple directories at the same time?
2. What directories are automatically created when using `mkdir`?

## mv

The `mv` command is used to rename and move files or directories. It is one of the basic Linux commands that allow for management of files from the command line.

- `mv [options] source destination` : Move and/or rename files and directories

For example: to rename the file `foo.txt` and to call it `foo-2.txt` we would enter the following;

```
mv foo.txt foo-2.txt
```

This makes the assumption that we are in the same directory as the file `foo.txt`, but even if we weren't we could explicitly name the file with the directory structure and thereby not just rename the file, but move it somewhere different;

```
mv /home/pi/foo.txt /home/pi/stuff/foo-2.txt
```

To move the file without renaming it we would simply omit the new name at the destination as so;

```
mv /home/pi/foo.txt /home/pi/stuff/
```

## The `mv` command

The `mv` command is used to move or rename files and directories (`mv` is an abbreviated form of the word **move**). This is a similar command to the `cp` (copy) command but it does not create a duplicate of the files it is acting on.

If we want to move multiple files, we can put them on the same line separated by spaces.

The normal set of [wildcards](#) and [addressing](#) options are available to make the process more flexible and extensible.

## Options

While there are a few options available for the `mv` command the one most commonly ones used would be `-u` and `-i`.

- `u` This updates moved files by only moving the file when the source file is newer than the destination file or when the destination file does not exist.
- `i` Initiates an interactive mode where we are prompted for confirmation whenever the move would overwrite an existing target.

## Examples

To move all the files from directory1 to directory2 (directory2 must initially exist);

```
mv directory1/* directory2/
```

To rename a directory from directory1 to directory2 (directory2 must not already exist);

```
mv directory1/ directory2/
```

To move the files foo.txt and bar.txt to the directory foobar;

```
mv foo.txt bar.txt foobar/
```

To move all the 'txt' files from the users home directory to a directory called backup but to only do so if the file being moved is newer than the destination file;

```
mv -u ~/.txt backup/
```

## Test yourself

1. How can we move a file to a new location when that act might overwrite an already existing file?
2. What characters cannot be used when naming directories or files?

## pwd

The `pwd` command allows us to determine which directory we are currently in. The flexibility of working in Linux means that often it becomes unclear exactly which directory we are working in. As a result, the `pwd` command becomes a simple way to affirm the present working directory which aids in ensuring that we are operating on the correct files.

- `pwd` : prints the current working directory

The `pwd` command is relatively unique in that it is most often executed by itself with no options or arguments;

```
pwd
```

it will print the current working directory to the command line;

```
/home/pi
```

It is a simple command to do a simple (but useful) function.

## The `pwd` command

The `pwd` command is short for **p**rint **w**orking **d**irectory. and it does just that. It is often taken to be short for **p**resent **w**orking **d**irectory and while that is an equally fine description, the name derives from the Unix world where computer output was commonly onto paper.

The simplicity of the command mirrors the Unix philosophy of small specialised commands that are modular. It is a testament to the veracity of the statement that `pwd` has continued to remain as one of the simplest and most useful commands.

While it has a couple of options, these are seldom if ever used and the command stands on its own right in isolation.

## Test yourself

1. What directory will `pwd` report if you execute the command `cd ~`?

## rm

The `rm` command is used to remove file or directories. This is a basic Linux file management command that should be understood by all Linux users.

- `rm [options] file` : Delete files or directories

For example if we want to remove the file `foo.txt` we would use the command as follows;

```
rm foo.txt
```

## The `rm` command

The `rm` command is used to remove (hence `rm` = **remove**) files. Any act that involves deleting files should be treated with a degree of caution and the `rm` command falls into the ‘treat with caution’ category. This is especially true if using wildcards or complex relative [addressing](#).

It will not remove directories by default although it can be directed to do so via the `-r` option (covered later). The command will return an error message if a file doesn’t exist or if the user doesn’t have the appropriate permissions to delete it. Files which are located in write-protected directories can not be removed, even if those files are not write-protected (they are protected by the directory).

If a file to be deleted is a symbolic link, the link will be removed, but the file or directory to which that link refers will not be deleted.



Believe it or not, it is sometimes possible to recover the contents of a file that has been deleted using `rm`. If our objective when deleting the file is to ensure that it can’t be retrieved (presumably for a security reason) another option should be used. The command `shred` would be a start, please explore your options in this case depending on the level of assurance you need.

## Options

There are a small number of options available for use with `rm`. The following would be the most common;

- `-r` allows us to recursively remove directories and their contents
- `-f` allows us to force the removal of files irrespective of write-protected status
- `-i` tells the `rm` command to interactively prompt us for confirmation of every deletion

So the following will delete the `foobar` directory and all its contents;

```
rm -r foobar/
```

To delete all the txt files in the current directory irrespective of their write-protect status (and without prompting us to tell us that it's happening);

```
rm -f *.txt
```



Seriously. Please be aware of what you're doing if you're using the `-r` or `-f` options and be doubly careful if using them together!

To take a nice careful approach and to have a prompt for the removal of each file we can use the `-i` option. The following example will look for each txt file and ask us if we want to delete it;

```
rm -i *.txt
```

The output will look something like the following;

```
pi@raspberrypi ~ $ rm -i *.txt
rm: remove regular file `bar.txt'? y
rm: remove regular file `foo.txt'? y
```

We can't use the `-f` and `-i` options simultaneously. Whichever was the last one in the command takes affect.

The `rm` command supports the `--` (two consecutive dashes) parameter which acts as a delimiter that indicates the end of any options. This is used when the name of a file or directory begins with a dash. For example, the following removes a directory named `-directory1`;

```
rm -r -- -directory1
```

## Arguments

The normal set of [wildcards](#) and [addressing](#) options are available to make the process of finding the right files more flexible and extensible.

To remove more than one file we can simply separate them with a space as follows;

```
rm file1 file2 file3
```

## Test yourself

1. Will the `-f` option successfully delete a write protected file in a write protected directory?  
Justify your answer.
2. What are the implications of running the following command;

```
rm -if *.png
```

## **rmmdir**

The `rmmdir` command is used to remove empty directories. This is a basic Linux file management command that should be understood by all Linux users.

- `rmmdir [options] directory` : Delete empty directories

For example if we want to remove the directory `foobar` we would use the command as follows;

```
rmmdir foobar
```

## **The `rmmdir` command**

The `rmmdir` command is used to remove empty directories (hence `rmmdir` = **remove directory**). It may seem like an unusual thing to only be able to delete empty directories, but this acts as a safeguard that can be used to prevent the unintentional loss of files that might result from trying to remove a directory when it wasn't empty.

There is no equivalent option as with `rm` that allows recursive deletion of directories, but there is a variation that will allow the deletion of parent directories if they are subsequently emptied (more later).

In Linux, directories are not *actual* file containers. Instead they are files that have the appearance of containing files because they store a set of data structures that describe files and subdirectories that are assigned to the 'directory'. Therefore deleting a directory has the potential to significantly lose a great deal of data and the restriction on only deleting empty directories is understandable.

## **Options**

The only potentially common option used with the `rmmdir` command is the `-p` option which removes explicitly specified directories if they are emptied. For example, the following command will delete `directory3`, then `directory2` and then `directory1` as each deletion subsequently leaves it's parent empty;

```
rmmdir -p directory1/directory2/directory3/
```

## **Arguments**

The normal set of [wildcards](#) and [addressing](#) options are available to make the process of finding the right directories more flexible and extensible.

To remove more than one directory we can simply separate them with a space as follows;



```
rm -rf directory1 directory2 directory3
```

## Test yourself

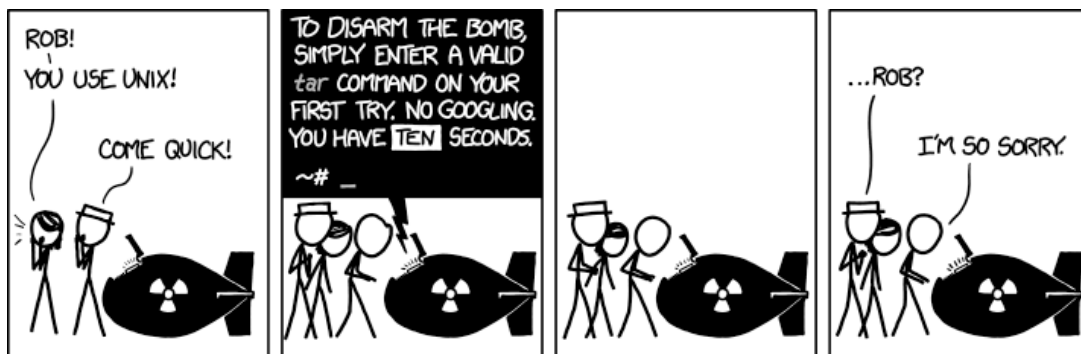
1. What command can be used to delete a directory that contains files?
2. Why does the `rm -rf` command not allow directories with contents to be deleted?

## tar

The `tar` command is designed to facilitate the creation of an archive of files by combining a range of files and directories into a single file and providing the ability to extract these files again. While `tar` does not include compression as part of its base function, it is available via an option. `tar` is a useful program for archiving data and as such forms an important command for good maintenance of files and systems.

- `tar [options] archivename [file(s)]` : archive or extract files

`tar` is renowned as a command that has a plethora of options and flexibility. So much so that it can appear slightly arcane and (dare I say it) ‘over-flexible’. This has been well illustrated in the excellent cartoon work of the [xkcd comic strip](https://xkcd.com/149/)<sup>19</sup> (Buy his stuff<sup>20</sup>, it’s awesome!).



Sudo courtesy xkcd

However, just because it has a lot of options does not mean that it needs to be difficult to use for a standard set of tasks and at its most basic is the creation of an archive of files as follows;

```
tar -cvf foobar.tar foo.txt bar.txt
```

Here we are creating an archive in a file called `foobar.tar` of the files `foo.txt` and `bar.txt`.

The options used allow us to;

- `c` : create a new archive
- `v` : verbosely list files which are processed.
- `f` : specify the following name as the archive file name

The output of the command is the echoing of the files that are placed in the archive;

<sup>19</sup><https://xkcd.com/149/>

<sup>20</sup><http://store.xkcd.com/>

```
foo.txt  
bar.txt
```

The additional result is the creation of the file containing our archive `foobar.tar`.

To carry the example through to its logical conclusion we would want to extract the files from the archive as follows;

```
tar -xvf foobar.tar
```

The options used allow us to;

- `x` : extract an archive
- `v` : verbosely list files which are processed.
- `f` : specify the following name as the archive file name

The output of the command is the echoing of the files that are extracted from the archive;

```
foo.txt  
bar.txt
```

## The `tar` command

tape archive, or `tar` for short, is a command for converting files and directories into a single data file. While originally written for reading and writing from sequential devices such as tape drives, it is nowadays used more commonly as a file archive tool. In this sense it can be considered similar to archiving tools such as WinZip or 7zip. The resulting file created as a result of using the `tar` command is commonly called a 'tarball'.

Note that `tar` does not provide any compression, so in order to reduce the size of a tarball we need to use an external compression utility such as [gzip](#). While this is the most common, any other compression type can be used. These switches are the equivalent of [piping](#) the created archive through [gzip](#).

One advantage to using `tar` over other archiving tools such as `zip` is that `tar` is designed to preserve Unix filesystem features such as user and group permissions, access and modification dates, and directory structures.

Another advantage to using `tar` for compressed archives is the fact that any compression is applied to the tarball in its entirety, rather than individual files within the archive as is the case with `zip` files. This allows for more efficient compression as the process can take advantage of data redundancy across multiple files.

## Options

- `c` : Create a new tar archive.
- `x` : Extract a tar archive.
- `f` : Work on a file.
- `z` : Use Gzip compression or decompression when creating or extracting.
- `t` : List the contents of a tar archive.



While we have already seen `c`, `x`, `v` and `f` in action in the examples above it is worth noting that it is necessary that the `-f` option be the *final* option in the sequence of options; otherwise, the command will not be able to specify the name for the archive file and may use the next option in the sequence as the name.

The tar program does not compress the files, but it can incorporate the [gzip](#) compression via the `-z` option as follows;

```
tar -cvzf foobar.tar foo.txt bar.txt
```

If we want to list the contents of a tarball without un-archiving it we can use the `-t` option as follows;

```
tar -tf foobar.tar
```

When using tar to distribute files to others it is considered good etiquette to have the tarball extract into a directory of the same name as the tar file itself. This saves the recipient from having a mess of files on their hands when they extract the archive. Think of it the same way as giving your friends a set of books. They would much rather you hand them a box than dump a pile of loose books on the floor.

For example, if you wish to distribute the files in the `foodir` directory then we would create a tarball from the directory containing these files, rather than the files themselves:

```
tar -cf foodir.tar foodir
```

Remember that tar operates recursively by default, so we don't need to specify all of the files below this directory ourselves.

## Test yourself

1. Do you need to include the `z` option when decompressing a tar archive?
2. Enter a valid tar command on the first try. No Googling. You have 10 seconds.

# Accessing File Contents

## cat

The `cat` command is a really versatile command that is typically used to carry out three different functions. It can display a file on the screen, combine different files together (concatenate them) or create new files. This is another core command that is immensely useful to learn in when working with Linux from the command line. It's simple, flexible and versatile.

- `cat [options] filename filename` : Display, combine or create new files.

For example: To display the file `foo.txt` on the screen we would use;

```
cat foo.txt
```

To display the files `foo.txt` and `bar.txt` on the screen one after the other we would use;

```
cat foo.txt bar.txt
```

Or to combine the files `foo.txt` and `bar.txt` into a new file called `foobar.txt` using the redirection symbol `>`;

```
cat foo.txt bar.txt > foobar.txt
```

## The `cat` command

The `cat` command is a vital tool to use on the Linux command line because ultimately Linux is an operating system that is [file driven](#). Without a graphical user interface there needs to be a mechanism whereby creating and manipulating text files can be accomplished easily. The `cat` command is one of the commands that makes this possible. The name 'cat' is short for 'catenate' or 'concatenate' (either appears to be acceptable, but 'concatenate' appears to be more widely used), which is to say to connect things in a series. This is certainly one of it's common uses, but a better overview would be to say that the `cat` command is used to;

- Display text files at the command line
- Join one text file to the end of another text file, combining them
- Copy text files into a new document

## Options

The only option that gets any degree of use with cat is the -n option that numbers the output lines.

## Arguments and Examples

### To display text

For example, to display a text file (foo.txt) on the screen we can use the following command;

```
cat foo.txt
```

The output would be;

```
pi@raspberrypi ~ $ cat foo.txt  
This line is the contents of foo.txt
```

As we can see, the contents of the file 'foo.txt' is sent to the screen (be aware, if the file is sufficiently large, it will simply dump the contents in a long scrolling waterfall of text).

### To join more than one file together

We could just as easily display two files one after the other (concatenated) as follows;

```
cat foo.txt bar.txt
```

The output would be;

```
pi@raspberrypi ~ $ cat foo.txt bar.txt  
This line is the contents of foo.txt  
This line is the contents of bar.txt
```

### To create a new file

Instead of having the file sent to the screen, we can specify that cat send our file to a new (renamed) file as follows;

```
cat foo.txt > newfoo.txt
```

This could be thought of as an equivalent to a file copy action and uses the redirection symbol `>`.



`>>` and `>` are called append symbols. They are used to append the output to a file and not to the screen. `>` will delete a file if it already exists and create a new file hence for safety it is advisable to use `>>` wherever possible to write the output to a new file without overwriting or deleting an existing file.

Taking the process one step further we can take our original two files and combine them into a single file with;

```
cat foo.txt bar.txt > foobar.txt
```

We can then check the results of our combination using `cat` on the new file as follows;

```
cat foobar.txt
```

And the output would be;

```
pi@raspberrypi ~ $ cat foobar.txt
This line is the contents of foo.txt
This line is the contents of bar.txt
```

Then we could use `cat` to append a file to an already existing file by using the redirection operator `>>`;

```
cat newfoo.txt >> foobar.txt
```

Here we use the redirection operator `>>` to add the contents of the file `newfoo.txt` to the already existing file `foobar.txt`.

The resulting file content would be;

```
pi@raspberrypi ~ $ cat foobar.txt
This line is the contents of foo.txt
This line is the contents of bar.txt
And this is newfoo.txt
```

Finally, we can use `cat` to create a file from scratch. In this scenario if we use `cat` without a source file and redirect to a new file (here called `newfile.txt`). It will take the input from the command line to add to the file until `CONTROL-d` is pressed.

```
cat >> newfile.txt
Just keep typing
and entering information until...
we press ctrl-d to finish and the file gets written!
```

The resulting file content would be;

```
pi@raspberrypi ~ $ cat newfile.txt
Just keep typing
and entering information until...
we press ctrl-d to finish and the file gets written!
```

## Test yourself

1. Which is the safest redirector to use?
2. Create a new file using `cat` and enter a few lines of text to give it some content
3. Copy that file using `cat` to a new file.
4. Combine the original file and the copy into a new file.
5. Display that new file on the screen.



## cut

The `cut` command is used to ‘cut out’ sections of each line of a file or files. It can perform the cut based on *byte*, *character* or *field* position in a row. When using field position the default separator is a tab, but this can be specified as an alternative using the options. This command will allow us to filter text files easily in situations where alternatives would be manual editing.

- `cut [options] file` : ‘cut out’ sections of each line of a file or files.

For example, let's assume that we have a file named `cutme.txt` with the following contents;

one	two	three	four	five
uno	dos	tres	cuatro	cinco
tahi	rua	toru	wha	rima



It is important to note that each line has each word separated by a tab character, not a space or spaces.

Using the `cut` command as follows we can cut out the third field of each line where a field is any number of characters between delimiters;

```
cut -f 3 cutme.txt
```

... and the results are as follows;

```
three
tres
toru
```

In the command above we have used the `-f` option that specifies which number ‘field’ to cut out between the tab (default) delimiters.

## The `cut` command

The `cut` command is not one that will see everyday usage, but when dealing with manipulation of information in a [file](#) or even a stream of input data from STDIN, it can be invaluable. It is one of the commands that can be used to efficiently filter information.

Its utility is demonstrated by the ability to ‘cut’ information based on a flexible scheme for specifying a position, character(s) or fields for cutting and the ability to specify the delimiters to be used to

Using our sample file `cutme.txt` from above we can specify cutting out the fields from 2 to 4 as follows;

```
cut -f 2-4 cutme.txt
```

... and the results are as follows;

```
two    three    four
dos    tres     cuatro
rua     toru     wha
```

Remembering that the command has assumed the use of tabs as a delimiter by default.

## Options

There are a few different options for use, but the more commonly used are as follows;

- `-c` cut based on the number of characters
- `-b` cut based on the number of bytes
- `-f` cut based on the number of fields that are delimited
- `-d` use a specific character as a delimiter

Cutting each line based on character or byte position can be thought of as almost the same thing and for most purposes it can be considered so. The difference is that in most cases a character is a single byte long, but for some international character sets the characters require more than one byte to encode them. So while for the vast majority of cases, the use of `-c` or `-b` can be considered to have the same result, always be mindful of the difference between the two and use the option that is best suited to the task.

For our next set of examples let's assume that we have a file named `cutme2.txt` with the following contents;

```
one:two:three:four:five
uno:dos:tres:cuatro:cinco
tahi:rua:toru:wha:rima
```

Obviously in this example file we have replaced the tabs with colons (:) which can be used as delimiters.

Using the `-c` option to cut on character, we could select the range of characters from the second to the sixth as follows;

```
cut -c 2-6 cutme2.txt
```

... which would produce an output as follows;

```
ne:tw  
no:do  
ahi:r
```

We can include more than one section from a row to cut out by separating the sections with a comma. For example, if we wanted to select the range of characters from the second to the sixth and the eighth to the tenth (omitting the ninth) we could do it as follows;

```
cut -c 2-6,8-9 cutme2.txt
```

... which would produce an output as follows;

```
ne:tw:t  
no:do:t  
ahi:ra:
```

In addition we can use the command to cut every character from a specified position to the end of the row by omitting the 'to' number. For example we could select the range of characters from the second character to the end of the row as follows;

```
cut -c 2- cutme2.txt
```

... or we could cut out every character from the start of the row up to the second character as follows;

```
cut -c -2 cutme2.txt
```

To set a delimiter other than the default tab, we use the `-d` option. The following example cuts out the second field using a colon as the delimiter;

```
cut -f 2 -d : cutme2.txt
```

The output will appear as follows;

```
two  
dos  
rua
```

The delimiter in the command can be encapsulated by quote marks or speachmarks as per the following list of equivalent commands, but the delimiter can only be a single character;

```
cut -f 2 -d : cutme2.txt  
cut -f 2 -d ':' cutme2.txt  
cut -f 2 -d ":" cutme2.txt
```

## Test yourself

1. Show two different ways of using the cut command that will cut out every character from the start of a row up till the fourth character and from the sixth character till the end of the row.

## diff

The `diff` command is used for comparing two files or directories and reporting what the differences are between them. It is a common situation to find ourselves wondering what the difference is between two almost identical files or directories and this command will determine those differences.

- `diff [options] from-file to-file` : Display the differences between two files

For example, let's assume that we have two directories, 'dir1' and 'dir2'. Each of these directories holds three files. 'dir1' contains 'file\_a.txt', 'file\_b.txt' and 'file\_c.txt'. 'dir2' contains 'file\_a.txt', 'file\_b.txt' and 'file\_d.txt'.

For this simple example we can see that the difference between the two directories is that only 'dir1' contains 'file\_c.txt' and only dir2 contains 'file\_d.txt'. We can illustrate this slightly better as follows;

```
|
├─ dir1
|   ├── file_a.txt
|   ├── file_b.txt
|   └─ file_c.txt
└─ dir2
    ├── file_a.txt
    ├── file_b.txt
    └─ file_d.txt
```

Using the `diff` command as follows we can compare the 'dir1' and 'dir2' directories;

```
diff -q dir1 dir2
```

... and the results are as follows;

```
pi@raspberrypi ~ $ diff -q dir1 dir2
Files dir1/file_a.txt and dir2/file_a.txt differ
Files dir1/file_b.txt and dir2/file_b.txt differ
Only in dir1: file_c.txt
Only in dir2: file_d.txt
```

The reporting in this case is in a fairly reasonable, descriptive human readable form. It tells us that not only is there different files in each directory, but where there are files of the same name, it let's us know that they are different from each other.

In the command above we used the `-q` option which omits the individual differences between the files which have the same name, but different contents (otherwise the results from the command may have been fairly confusing).

## The `diff` command

There comes times in our computing lives when we've made changes to files or directories and in the process of moving things about or installing different components we lose track of the changes that we've made. This can happen in programming, system administration, application installation or user management, but whatever the cause, we find ourselves needing to compare two things, be they files or directories, to try see what the **differences** are between them. This is where the `diff` command comes in.

Comparing two complex objects from the command line has the potential to be difficult. If there are substantial differences, then using a plain text interface to get good context for the changes could be challenging. However, if the changes being looked for are small or we are needing to carry out comparisons a lot, then being able to compare files at the command line is a quick method once we're used to it.

In it's purest form, `diff` analyses two [files](#) and prints the lines that are different. We can think of the output to the screen as providing direction on how to make the changes to make one thing identical to the other.

In the simplest case, `diff` compares the contents of the two files *from-file* and *to-file*.

If both *from-file* and *to-file* are [directories](#), `diff` compares corresponding files in both directories, in alphabetical order. This comparison is not recursive unless the `-r` option is given.

As an example of a comparison of files we can consider the two files 'file\_a.txt' and 'file\_b.txt' as below;

### file\_a.txt

The Raspberry Pi is a mighty machine  
Leanpub has a great range of books  
d3.js is awesome for visualization  
Linux is a marvel of collaboration

### file\_b.txt

The Raspberry Pi is a wonderful machine  
Leanpub has a great range of books  
d3.js is awesome for visualization  
Linux is a marvel of collaboration

The differences between the files is in the first line where we describe the Raspberry Pi as either a *mighty* or a *wonderful* machine.

We can then execute the `diff` command on those two files as in the example below;

```
diff file_a.txt file_b.txt
```

This will produce an output as follows;

```
pi@raspberrypi ~/dir1 $ diff file_a.txt file_b.txt
1c1
< The Raspberry Pi is a mighty machine
---
> The Raspberry Pi is a wonderful machine
```

The first line of the output that reads `1c1` tells us that line 1 ('`1c1`') from the *from-file* needs to

be changed to match line 1 ('1c1') of the *to-file*. If either file had a range of lines they would be designated as something like 2,4 which would indicate that the affected lines were from 2 to 4.

The remaining output is the different lines. To show which lines come from which file, each one is prefixed by either a greater-than symbol (<) to show that the line is from the *from-file* or a less-than sign (>) to show that the line is from the *to-file*. The dashes between the lines (---) are simply there to separate the lines from the different files.

## Options

There are plenty of different options for use, but some of the more commonly used are as follows;

- -q Report only when files differ, not the actual differences (per first example)
- -c Format the output with more context
- -i ignore any differences in case
- -r recursively compare subdirectories.

Because the process of comparing the differences in files and directories is a difficult thing to do in a simple way with text there is an alternative option to use to provide an extra degree of context. This is the -c option.

If we use this option on the same two files that we tested earlier (`diff -c file_a.txt file_b.txt`) we will see something similar to this;

```
pi@raspberrypi ~/dir1 $ diff -c file_a.txt file_b.txt
*** file_a.txt  2015-09-18 21:10:27.608873625 +0000
--- file_b.txt  2015-09-18 21:29:26.269130682 +0000
*****
*** 1,4 ****
! The Raspberry Pi is a mighty machine
  Leanpub has a great range of books
  d3.js is awesome for visualization
  Linux is a marvel of collaboration
\ No newline at end of file
--- 1,4 ----
! The Raspberry Pi is a wonderful machine
  Leanpub has a great range of books
  d3.js is awesome for visualization
  Linux is a marvel of collaboration
\ No newline at end of file
```

The first two lines of output give us some modification date / time detail about our *from-file* (prefixed by three asterisks (\*\*\*)) and our *to-file* (prefixed by three hyphens (---)).

Then there is a separator composed of a row of asterisks before there is a line with the numbers 1,4 between some asterisks. This tells us that the following lines are a listing of lines 1 through 4 from the *from-file*. Likewise, the next section with the number 1,4 between hyphens tells us that the following lines are a listing of lines 1 through 4 from the *to-file*.

**Test yourself**

1. Rank the ease of understanding the output when carrying out the diff command using the default method without options, the -c option and -q.



## grep

The `grep` command is used to search text from files or [piped](#) input for lines containing a match to a given pattern. It is one of the most powerful and versatile commands in Linux and mastery of the `grep` command is synonymous with a good understanding of the command line interface.

- `grep [options] pattern [file]` : Search text and match patterns

For example the following command will search for the text “cpus” in the file `/var/log/dmesg`;

```
grep cpus /var/log/dmesg
```

The output from the command will appear similar to the following;

```
pi@raspberrypi ~ $ grep cpus /var/log/dmesg
[    0.000000] [bcm2709_smp_init_cpus] enter (8620->f3003010)
[    0.000000] [bcm2709_smp_init_cpus] ncores=4
[    0.053615] [bcm2709_smp_prepare_cpus] enter
```

It should be noted that each of the lines above features the text “cpus”.

## The `grep` command

The `grep` command is used to search text from files or piped input for lines containing a match to a given pattern. When it finds a match, it copies the line that the match is on to standard output (by default), or whatever other sort of output you have requested with options. The pattern may be a single character, a group of characters, a single word or a sentence.

The name `grep` derives from the command used with the Unix/Linux text editor ‘ed’ to carry out a similar search function. The command takes the form `g/re/p`, which signifies searching globally for matches to the regular expression (i.e., `re`), and `p`rinting lines that are found onto the screen.

Though `grep` anticipates matching on text, it can match arbitrary characters on a line. Its only limitation on the amount of information that it can search is the available memory.

## Regular Expressions

The `re` portion of `grep` is an indicator of the use of [regular expressions](#) to match patterns while searching. This means that as well as being able to search for literal patterns (such as our `cpu` text from the initial example), we are able to use a schema that allows a huge range of flexibility in the returned matches.

For example, if we wanted to search for occurrences of the text ‘CPU’ where there could be a number from 0 to 3 after it, we could use the following command

```
grep CPU[0-3] /var/log/dmesg
```

... which will result in;

```
pi@raspberrypi ~ $ grep CPU[0-3] /var/log/dmesg
[  0.053503] CPU0: update cpu_capacity 1024
[  0.053577] CPU0: thread -1, cpu 0, socket 15, mpidr 80000f00
[  0.113791] CPU1: Booted secondary processor
[  0.113851] CPU1: update cpu_capacity 1024
[  0.113860] CPU1: thread -1, cpu 1, socket 15, mpidr 80000f01
[  0.133710] CPU2: Booted secondary processor
[  0.133746] CPU2: update cpu_capacity 1024
[  0.133755] CPU2: thread -1, cpu 2, socket 15, mpidr 80000f02
[  0.153750] CPU3: Booted secondary processor
[  0.153788] CPU3: update cpu_capacity 1024
[  0.153797] CPU3: thread -1, cpu 3, socket 15, mpidr 80000f03
```

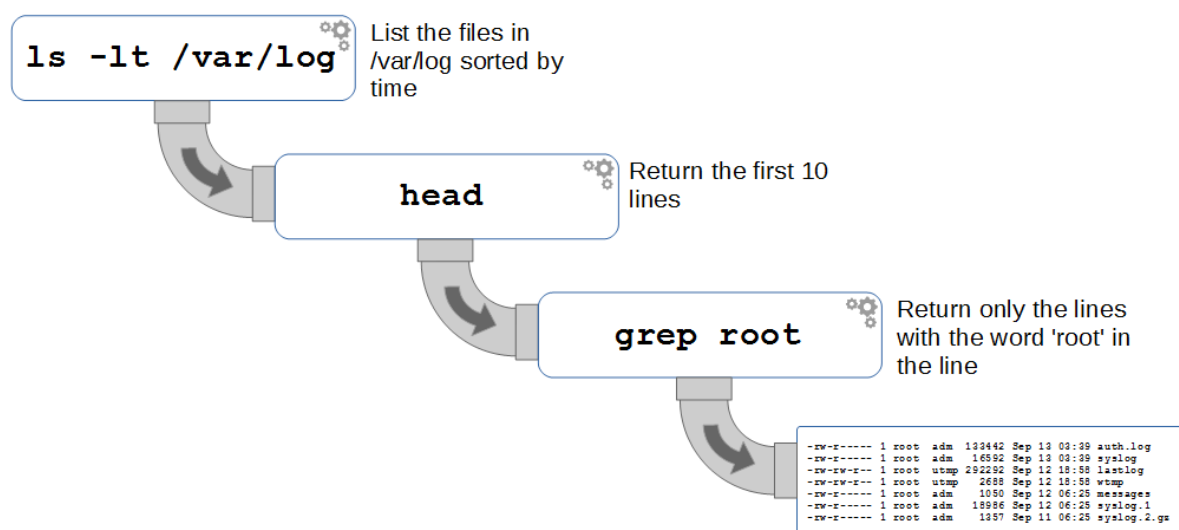
Here `grep` has used the text characters ‘CPU’ as ‘*literals*’ that should appear exactly as stated and the square brackets denote a regular expression that would allow any numbers in the range 0 to 3.

## Pipes

Because the `grep` command can act as a filter, it is commonly employed as a mechanism to accept an input from one command and to produce an output that has been altered by that filter. That connection of command is carried out by [pipes](#) which are designated by the `|` character.

As the name suggests, we can think of the pipe function as representing commands linked by a pipe where the command runs a function that is output in a pipe to flow to the second command and on to the eventual final output.

To demonstrate by example we can consider a set of commands linked as follows;



### Piping example

Here we have the `ls` command that is listing the contents of the `/var/log` directory with the listings sorted by time. This is then feeding into the `head` command that will only display the first 10 of those lines. We then feed those 10 lines into a `grep` command that filters the result to show only those lines that have the word 'root' in them.

The command as it would be run from the command line would be as follows;

```
ls -lt /var/log | head | grep root
```

... and the output would appear something like the following;

```
pi@raspberrypi ~ $ ls -lt /var/log | head | grep root
-rw-r----- 1 root adm 133637 Sep 13 04:01 auth.log
-rw-r----- 1 root adm 16794 Sep 13 04:01 syslog
-rw-rw-r-- 1 root utmp 292292 Sep 12 18:58 lastlog
-rw-rw-r-- 1 root utmp 2688 Sep 12 18:58 wtmp
-rw-r----- 1 root adm 1050 Sep 12 06:25 messages
-rw-r----- 1 root adm 18986 Sep 12 06:25 syslog.1
-rw-r----- 1 root adm 1357 Sep 11 06:25 syslog.2.gz
```

## Options

There are a large number of options available to use with the `grep` command. For a full listing type `man grep` on the command line. Some of the most commonly used are;

- `-c` Print a count of matching lines for each input file
- `-E` Interpret the *pattern* as an Extended [regular expression](#)
- `-n` Prefix each line of output with the line number within its input file

- -i Make the search case insensitive (so letters could be upper or lower case)
- -o Show **only** the part of a matching line that matches *pattern*

If we wanted to find out how many lines in the input we are testing have instances of the pattern we are trying to match we can use the -c option.

For example if we wanted to find out how many times the text 'CPU2' occurs in the /var/log/dmesg file we would run the following command;

```
grep -c CPU2 /var/log/dmesg
```

... and the output would appear something like the following;

```
pi@raspberrypi ~ $ grep -c CPU2 /var/log/dmesg
3
```

We can therefore be confident that the number of lines where 'CPU2' occurs is 3.

The bash shell where we could be running our grep command regards the following as characters as 'special'; ?, +, {, |, ( and ). As such, when they are used in a [regular expression](#) they need to be treated slightly differently to allow them to be processed properly. There are two ways to accomplish this. Either to run the grep command as egrep or to use the -E option, both of which allow the use of extended [regular expressions](#).

If we try to run a grep command without the -E option we will get an error essentially telling us that the command doesn't recognise part of our command. For example the following command

...

```
grep (CPU1)|(CPU2) /var/log/dmesg
```

... will result in an error similar to the one below;

```
pi@raspberrypi ~ $ grep (CPU1)|(CPU2) /var/log/dmesg
-bash: syntax error near unexpected token `('
```

But if we use the -E option and then contain our search pattern in quotes (either speech-marks or single quotes) we will take advantage of the special characters and execute the command correctly. For example the following command searches for instances of either CPU1 or CPU2 in the file '/var/log/dmesg';

```
grep -E "(CPU1)|(CPU2)" /var/log/dmesg
```

The output will then look a little like the following;

```
pi@raspberrypi ~ $ grep -E "(CPU1)|(CPU2)" /var/log/dmesg
[  0.113791] CPU1: Booted secondary processor
[  0.113851] CPU1: update cpu_capacity 1024
[  0.113860] CPU1: thread -1, cpu 1, socket 15, mpidr 80000f01
[  0.133710] CPU2: Booted secondary processor
[  0.133746] CPU2: update cpu_capacity 1024
[  0.133755] CPU2: thread -1, cpu 2, socket 15, mpidr 80000f02
```

As an alternative we could ‘escape’ the special characters by placing a ‘backslash’ character in front of them like so;

```
grep -E \"(CPU1\\)|(CPU2\\)\" /var/log/dmesg
```

Or alternatively we could use the ‘egrep command for either option;

```
egrep \"(CPU1\\)|(CPU2\\)\" /var/log/dmesg
```

... or ...

```
egrep \"(CPU1)|(CPU2)\" /var/log/dmesg
```

If we want to tell which number line the returned output comes from we can use the -n option. The following command will list the number of the line from the /var/log/dmesg file that matches the pattern CPU2;

```
grep -n CPU2 /var/log/dmesg
```

As a result, the output should look something like the following;

```
pi@raspberrypi ~ $ grep -n CPU2 /var/log/dmesg
65:[    0.133710] CPU2: Booted secondary processor
67:[    0.133746] CPU2: update cpu_capacity 1024
68:[    0.133755] CPU2: thread -1, cpu 2, socket 15, mpidr 80000f02
```

The last option we will consider will return the portions of the matching line that matches the pattern provided. So for example if we look for the range of CPU numbers with the pattern CPU[0-4] in the file /var/log/dmesg using the following command;

```
grep -o CPU[0-4] /var/log/dmesg
```

... we will get a list of the matches similar to the following;

```
pi@raspberrypi ~ $ grep -o CPU[0-4] /var/log/dmesg
CPU0
CPU0
CPU1
CPU1
CPU1
CPU2
CPU2
CPU2
CPU3
CPU3
CPU3
```

## Test yourself

1. Craft a grep command that returns the matches and line numbers for all instances of 'usb' in /var/log/dmesg
2. Craft a grep command that will return all 'idVendor' numbers.

## head

The `head` command is designed to print the first part of a file to the screen. By default it will print out the first 10 lines. It is a very handy program for quickly checking out the start of a file.

- `head [options] filename(s)`: List the first part of a specified file.

For example if we want to list the first 10 lines of the file `/var/log/dmesg` we could do so as follows;

```
head /var/log/dmesg
```

This will result in an output similar to the following;

```
pi@raspberrypi ~ $ head /var/log/dmesg
[    0.000000] Booting Linux on physical CPU 0xf00
[    0.000000] Initializing cgroup subsys cpu
[    0.000000] Initializing cgroup subsys cpuacct
[    0.000000] Linux version 3.18.7-v7+ (dc4@dc4-XPS13-9333) (gcc version 4.8
[    0.000000] CPU: ARMv7 Processor [410fc075] revision 5 (ARMv7), cr=10c5387d
[    0.000000] CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing
[    0.000000] Machine model: Raspberry Pi 2 Model B
[    0.000000] cma: Reserved 8 MiB at 0x3a800000
[    0.000000] Memory policy: Data cache writealloc
[    0.000000] On node 0 totalpages: 241664
```

## The head command

The `head` command is designed to carry out the simple task of reading the first few lines of a file and presenting them to an output (by default, the screen). The default number of lines that the command will print out is 10, but that can be changed using the hyphen (-) option.

It is complimented by the [tail](#) command which will print out the *last* several lines of a file.

## Options

There are a small number of options available for `head`. There are two that are more commonly used than others;

- `-c[num]` prints the first *num* number of bytes of the file (where 8 bytes is a character)
- `-n[num]` prints the first *num* number of lines of the file

At first glance the `-c` option might look a little unusual, but what it does is provide a very exact method for printing out a specific number of letters of a file. For example if we wanted to print out the first 50 characters of the file `/var/log/dmesg` we would use the following;

```
head -c50 /var/log/dmesg
```

The subsequent output would look as follows;

```
pi@raspberrypi ~ $ head -c41 /var/log/dmesg
[    0.000000] Booting Linux on physical pi@raspberrypi ~ $
```

We can see that the console prompt has actually been placed on the ends of the printed out text and that the text returned is 41 characters.

With the `-n` option, there is a shortcut with `head` where we can omit the `n` and it will automatically assume that if we put a hyphen and a number we are wanting that number of lines to be returned. Therefore a command like the following will print out the first 5 lines of the `/var/log/dmesg` file;

```
head -5 /var/log/dmesg
```

With a result that looks as follows;

```
pi@raspberrypi ~ $ head -5 /var/log/dmesg
[    0.000000] Booting Linux on physical CPU 0xf00
[    0.000000] Initializing cgroup subsys cpu
[    0.000000] Initializing cgroup subsys cpuacct
[    0.000000] Linux version 3.18.7-v7+ (dc4@dc4-XPS13-9333) (gcc version 4.8
[    0.000000] CPU: ARMv7 Processor [410fc075] revision 5 (ARMv7), cr=10c5387d
pi@raspberrypi ~ $
```

For either the `-c` or `-n` options we can specify a multiplier adjacent to the *num* number that will increase the number of returned characters or lines. For example `b` = 512, `kB` = 1000, `K` = 1024, `MB` = 1000x1000, `M` = 1024x1024, `GB` = 1000x1000x1000, `G` = 1024x1024x1024, and so on for `T`, `P`, `E`, `Z` and `Y`.

The following example will therefore print out the first 5000 characters of the `/var/log/dmesg` file.

```
head -c5kB /var/log/dmesg
```



## Arguments

If we want to display the start of more than one file we can simply list their file names one after the other. For example, to list the first 5 lines of the files `dmesg` and `syslog` (both of which are in `/var/log`) we can use the following command;

```
head -5 dmesg syslog
```

The results will look a little like the following;

```
pi@raspberrypi /var/log $ head -5 dmesg syslog
==> dmesg <==
[ 0.000000] Booting Linux on physical CPU 0xf00
[ 0.000000] Initializing cgroup subsys cpu
[ 0.000000] Initializing cgroup subsys cpuacct
[ 0.000000] Linux version 3.18.7-v7+ (dc4@dc4-XPS13-9333) (gcc version 4.8
[ 0.000000] CPU: ARMv7 Processor [410fc075] revision 5 (ARMv7), cr=10c5387d

==> syslog <==
Aug 28 06:25:03 raspberrypi rsyslogd: [origin software="rsyslogd"]
Aug 28 06:39:01 raspberrypi /USR/SBIN/CRON[1490]: (root) CMD
Aug 28 07:00:01 raspberrypi /USR/SBIN/CRON[1505]: (pi) CMD (/usr/bin/php)
Aug 28 07:00:41 raspberrypi /USR/SBIN/CRON[1504]: (CRON) info
Aug 28 07:09:01 raspberrypi /USR/SBIN/CRON[1516]: (root) CMD (/maxlifetime)
pi@raspberrypi /var/log $
```

We can see that at the start of each file the name of the file is printed out and there are indeed 5 lines of text each!

## Test yourself

1. What is the difference between the following commands `head -5 foo.txt`, `head -n5 foo.txt` and `head - 5 foo.txt`?

## less

The `less` command is used to display text files on the screen one page at a time. This is a useful command that has to some degree superseded the command `more` in that it provides a similar but expanded functionality (Get it? `less` is `more`).

- `less [options] filename` : Display text files

For example, to show the first page in the text file `/var/log/dmesg` that contains the text 'cpu-capacity' we could use the command as follows;

```
less +/"cpu_capacity" /var/log/dmesg
```

This would produce an output similar to the following;

```
[ 0.053503] CPU0: update cpu_capacity 1024
[ 0.053577] CPU0: thread -1, cpu 0, socket 15, mpidr 80000f00
[ 0.053615] [bcm2709_smp_prepare_cpus] enter
[ 0.053779] Setting up static identity map for 0x536c78 - 0x536cd0
[ 0.113483] [bcm2709_boot_secondary] cpu:1 started (0) 17
[ 0.113791] CPU1: Booted secondary processor
[ 0.113799] [bcm2709_secondary_init] enter cpu:1
[ 0.113851] CPU1: update cpu_capacity 1024
[ 0.113860] CPU1: thread -1, cpu 1, socket 15, mpidr 80000f01
[ 0.133451] [bcm2709_boot_secondary] cpu:2 started (0) 18
[ 0.133710] CPU2: Booted secondary processor
[ 0.133716] [bcm2709_secondary_init] enter cpu:2
[ 0.133746] CPU2: update cpu_capacity 1024
[ 0.133755] CPU2: thread -1, cpu 2, socket 15, mpidr 80000f02
[ 0.153508] [bcm2709_boot_secondary] cpu:3 started (0) 17
[ 0.153750] CPU3: Booted secondary processor
[ 0.153757] [bcm2709_secondary_init] enter cpu:3
[ 0.153788] CPU3: update cpu_capacity 1024
[ 0.153797] CPU3: thread -1, cpu 3, socket 15, mpidr 80000f03
[ 0.153891] Brought up 4 CPUs
[ 0.154012] SMP: Total of 4 processors activated (153.60 BogoMIPS).
[ 0.154045] CPU: All CPU(s) started in SVC mode.
[ 0.155051] devtmpfs: initialized
/var/log/dmesg
```

While the above command is identical to the initial command invoked in the `more` example, the output differs in that the searched for text is highlighted in the output text (although this is not shown in the printed page above).

The other major difference is in the interaction of the page while it is being displayed. Where `more` is a ‘pager’ in the sense that it is designed from the outset to work with a file one page at a time, if we use the up and down arrow keys we will find ourselves able to scroll up and down the file on the screen. Indeed, if some of our lines are long enough we can press the right or left arrow keys to move left and right on the file content.

## The `less` command

The `less` command is arguably best known for the fact that it is an improved method of viewing a file over the `more` command. The most noticable improvement is the ability to scroll forwards and backwards through the file although there are a *huge* range of additional features in addition to those found in `more`. In spite of what might look like obvious reasons to use the more modern and extensively featured command, there are still Linux distributions that will include one over the other (instead of both) as the smaller size of `more` over `less` (approximately 2000 lines of code compared to 27,000 lines respectively) makes `more` more attractive for lighter embedded systems or similar.

The `less` command is used to display text files one page at a time and as such it can be considered a member of the family of ‘pagers’. It is a program designed to emulate and improve on `more` and as such if you know the basics of `more` you will know the basics of `less`. The name is a nod to the phrase “*less is more*”.

The main function that `less` provides is to display as much text as will fit on a screen at a time and to subsequently move forwards and backwards in the file. The file can also scroll one line at a time by using the up and down arrow keys.

## Commands

`less` includes a huge raft of commands that will allow us to control the navigation through a text file. For the following commands, where we see `[k]`, `k` represents an (optional) number that we type as the argument before the command (the following is a subset showing only commonly used commands, there are a LOT more).

- `h` or `H` : Help. Display a summary of these commands. If you forget all the other commands, remember this one.
- `[k] <space>` or `f` : Display the next `k` lines of text [default = current screen size]
- `[k] b` : Display the previous `k` lines of text [default = current screen size]
- `[k] z` : Display next `k` lines of text [default = current screen size = `k`]
- *down arrow* : Scroll down the file one line at a time
- *up arrow* : Scroll up the file one line at a time
- `[k] <return>` : Display next `k` lines of text [default = 1]
- `q` or `Q` : Exit from `more`
- `F` : Scroll forward to the end of the file (equivalent of `tail -f`)
- `[k] b` or `<ctrl>B` : Skip backwards `k` screenfuls of text [default = 1]
- `=` : Display current file, lines displayed, byte position and percentage displayed
- `[k] /<pattern>` : Search forward for `k`th occurrence of *pattern* [default = 1]
- `[k] ?<pattern>` : Search backward for `k`th occurrence of *pattern* [default = 1]

- [k] n : Search forwards for kth occurrence of last *pattern* [default = 1]
- [k] N : Search backwards for kth occurrence of last *pattern* [default = 1]
- &<*pattern*> : Search and display only matching lines of *pattern*
- [k] g or < : Go to first line in file (or line k)
- [k] G or > : Go to last line in file (or line k)
- [k] p or % : Go to beginning of file (or k percent into file)

For example, executing the following command will begin paging through the `/var/log/dmesg` file

```
less /var/log/dmesg
```

While part way through displaying the file pressing the = character will show a range of information about the file similar to the following;

```
[ 0.000000] Normal zone: 780799 pages, LIFO batch:31
[ 0.000000] ACPI: PM-Timer IO Port: 0x408
[ 0.000000] ACPI: Local APIC address 0xfee00000
[ 0.000000] ACPI: LAPIC (acpi_id[0x01] lapic_id[0x00] enabled)
[ 0.000000] ACPI: LAPIC (acpi_id[0x02] lapic_id[0x01] enabled)
[ 0.000000] ACPI: LAPIC (acpi_id[0x03] lapic_id[0x02] enabled)
[ 0.000000] ACPI: LAPIC (acpi_id[0x04] lapic_id[0x03] enabled)
/var/log/dmesg lines 117-139/938 byte 9130/64465 14% (press RETURN)
```

Here we can see the name of the file `/var/log/dmesg`, the lines being displayed 117-139, the total number of lines 938, the byte position in the file in relation to the total number of bytes 9130/64465 and the percentage through the file 14%.

While part way through displaying the file pressing the & character followed by a pattern we want to search on will show only lines with the matching pattern. So if we pressed `&cpu`, something similar to the following would be displayed;

```
[ 0.053615] [bcm2709_smp_prepare_cpus] enter
[ 0.113483] [bcm2709_boot_secondary] cpu:1 started (0) 17
[ 0.113799] [bcm2709_secondary_init] enter cpu:1
[ 0.113851] CPU1: update cpu_capacity 1024
[ 0.113860] CPU1: thread -1, cpu 1, socket 15, mpidr 80000f01
[ 0.133451] [bcm2709_boot_secondary] cpu:2 started (0) 18
[ 0.133716] [bcm2709_secondary_init] enter cpu:2
[ 0.133746] CPU2: update cpu_capacity 1024
[ 0.133755] CPU2: thread -1, cpu 2, socket 15, mpidr 80000f02
[ 0.153508] [bcm2709_boot_secondary] cpu:3 started (0) 17
[ 0.153757] [bcm2709_secondary_init] enter cpu:3
```

```
[ 0.153788] CPU3: update cpu_capacity 1024
[ 0.153797] CPU3: thread -1, cpu 3, socket 15, mpidr 80000f03
& :
```

In the live example, every instance of the pattern `cpu` is highlighted.

## Options

There are several options available to use with the `less` command. Some of the most commonly used are;

- `-i` will ignore case in searches that do not contain uppercase.
- `-I` will ignore case in all searches.
- `-s` will squeeze multiple blank lines into one
- `-p string` specifies a *string* that will be searched for in the file.
- `+num` will start displaying the file at line **number** *num*
- `-N` will display line Numbers next to the lines

For example, we can tell the command to search for the text `'cpu'` when executing the command `less` as follows;

```
less -p cpu /var/log/dmesg
```

The output will present the first instance of the string `'cpu'` and highlight it and any other instances on the screen as follows;

```
[ 0.000000] Normal zone: 241664 pages, LIFO batch:31
[ 0.000000] [bcm2709_smp_init_cpus] enter (8620->f3003010)
[ 0.000000] [bcm2709_smp_init_cpus] ncores=4
[ 0.000000] PERCPU: Embedded 11 pages/cpu @ba05d000 s12864 r8192 d24000
[ 0.000000] pcpu-alloc: s12864 r8192 d24000 u45056 alloc=11*4096
[ 0.000000] pcpu-alloc: [0] 0 [0] 1 [0] 2 [0] 3
/var/log/dmesg
```

To get a clearer idea of the line numbers in a displayed file we can use the `less` command with the `-N` option as follows;

```
less -N /var/log/dmesg
```

The output will display the file similar to the following;

```
12 [ 0.000000] Normal zone: 1888 pages used for memmap
13 [ 0.000000] Normal zone: 0 pages reserved
14 [ 0.000000] Normal zone: 241664 pages, LIFO batch:31
15 [ 0.000000] [bcm2709_smp_init_cpus] enter (8620->f3003010)
16 [ 0.000000] [bcm2709_smp_init_cpus] ncores=4
17 [ 0.000000] PERCPU: Embedded 11 pages/cpu @ba05d000 s12864
/var/log/dmesg
```

## Test yourself

1. Show a page at random. Press `h` to display the help screen and count the number of commands and options available with `less`. See if you can guess how many there are for each first.
2. What three ways could be used to search for and display a particular string of text?

## more

The `more` command is used to display text files on the screen one page at a time. This is a useful command that has to some degree been superseded by the command `less` that provides a similar but expanded functionality.

- `more [options] filename` : Display text files

For example, to show the first page in the text file `var/log/dmesg` that contains the text 'cpu-capacity' we could use the command as follows;

```
more +/"cpu_capacity" /var/log/dmesg
```

This would produce an output similar to the following;

```
[ 0.004116] CPU: Testing write buffer coherency: ok
[ 0.004231] ftrace: allocating 19969 entries in 59 pages
[ 0.053503] CPU0: update cpu_capacity 1024
[ 0.053577] CPU0: thread -1, cpu 0, socket 15, mpidr 80000f00
[ 0.053615] [bcm2709_smp_prepare_cpus] enter
[ 0.053779] Setting up static identity map for 0x536c78 - 0x536cd0
[ 0.113483] [bcm2709_boot_secondary] cpu:1 started (0) 17
[ 0.113791] CPU1: Booted secondary processor
[ 0.113799] [bcm2709_secondary_init] enter cpu:1
[ 0.113851] CPU1: update cpu_capacity 1024
[ 0.113860] CPU1: thread -1, cpu 1, socket 15, mpidr 80000f01
[ 0.133451] [bcm2709_boot_secondary] cpu:2 started (0) 18
[ 0.133710] CPU2: Booted secondary processor
[ 0.133716] [bcm2709_secondary_init] enter cpu:2
[ 0.133746] CPU2: update cpu_capacity 1024
[ 0.133755] CPU2: thread -1, cpu 2, socket 15, mpidr 80000f02
[ 0.153508] [bcm2709_boot_secondary] cpu:3 started (0) 17
[ 0.153750] CPU3: Booted secondary processor
[ 0.153757] [bcm2709_secondary_init] enter cpu:3
[ 0.153788] CPU3: update cpu_capacity 1024
[ 0.153797] CPU3: thread -1, cpu 3, socket 15, mpidr 80000f03
[ 0.153891] Brought up 4 CPUs
[ 0.154012] SMP: Total of 4 processors activated (153.60 BogoMIPS).
--More--(35%)
```

Notice that the percentage shown at the bottom of the screen is an indication of how far through the file the contents shown are.

## The `more` command

The `more` command is used to display text files one page at a time and as such it can be considered a member of the family of ‘pagers’. It is a fairly mature program and as such there have been improvements that have been identified as desirable and subsequently included into the program `less` (in a clever nod to the phrase “*less is more*”). `more` will only move forward through a file (`less` will move backward as well).

The main function that `more` provides is to display as much text as will fit on a screen at a time and to subsequently move to the next page when the space bar is pressed or to advance a line at a time when the return key is pressed.

### Commands

`more` includes a raft of commands that will allow us to control the navigation through a text file. This means that while we are in the middle of displaying our file we can control how it scrolls (how many lines at a time for instance) or what text to search for. For the following commands, where we see `[k]`, `k` represents an (optional) number that we type as the argument before the command (the following is a subset showing only commonly used commands).

- `h` or `?` : Help. Display a summary of these commands. If you forget all the other commands, remember this one.
- `[k] <space>` : Display the next `k` lines of text [default = current screen size]
- `[k] z` : Display next `k` lines of text [default = current screen size]
- `[k] <return>` : Display next `k` lines of text [default = 1]
- `[k] d` or `<ctrl>D` : Scroll `k` lines [current scroll size, initially 11]
- `q` or `Q` : Exit from `more`
- `[k] s` : Skip forward `k` lines of text [default = 1]
- `[k] f` : Skip forward `k` screenfuls of text [default = 1]
- `[k] b` or `<ctrl>B` : Skip backwards `k` screenfuls of text [default = 1]
- `=` : Display current line number
- `[k] /<regular expression>` : Search for `k`th occurrence of [regular expression](#) [default = 1]
- `[k] n` : Search for `k`th occurrence of last [regular expression](#) [default = 1]
- `v` : Start up `/usr/bin/vi` at current line
- `<ctrl>L` : Redraw screen
- `[k] :n` : Go to `k`th next file [default = 1]
- `[k] :p` : Go to `k`th previous file [default = 1]
- `:f` : Display current file name and line number
- `.` : Repeat previous command

To explain further, if we run the following command to start paging through our file...

```
more /var/log/dmesg
```

... and then (while part way through viewing the file) press `6z` the display will show the next 6 lines of the file.



## Options

There are several options available to use with the `more` command. Some of the most commonly used are;

- `-num` will set the **number** of lines that the screen size corresponds to.
- `-s` will squeeze multiple blank lines into one
- `+/"string"` specifies a **string** that will be searched for in the file.
- `+num` will start displaying the file at line **number** *num*

For example, if we set the number of lines for the screen to 3 using the `-num` option as follows;

```
more -3 /var/log/dmesg
```

The output will assume that the screen is 3 lines deep and therefore show 3 lines for each additional page displayed with the `<space>` bar. The first 'page' displayed will look like the following;

```
pi@raspberrypi ~ $ more -3 /var/log/dmesg
[  0.000000] Booting Linux on physical CPU 0xf00
[  0.000000] Initializing cgroup subsys cpu
[  0.000000] Initializing cgroup subsys cpuacct
--More--(1%)
```

## Test yourself

1. What command would be used to show the first instance of the text 'hello world' from a file and then to show 5 lines at a time every time the space bar is pressed?
2. What is the alternative (more modern) command commonly used instead of `more`?
3. Use `more` to list the file `/var/log/dmesg`. Press the `s` key to skip forward a random number of lines. use a command to find the line that is being displayed then quit the command and run it again, but with an option that takes you straight to that same random line number.

## tail

The `tail` command is designed to print the last part of a file to the screen. By default it will print out the last 10 lines. It is a very handy program for quickly checking out the end of a file.

- `tail [options] filename(s)` : List the last part of a specified file.

For example if we want to list the last 10 lines of the file `/var/log/dmesg` we could do so as follows;

```
tail /var/log/dmesg
```

This will result in an output similar to the following;

```
pi@raspberrypi /var/log $ tail /var/log/dmesg
[  4.723858] EXT4-fs (mmcblk0p2): mounted filesystem with ordered data mode.
[  4.735464] VFS: Mounted root (ext4 filesystem) readonly on device 179:2.
[  4.745366] devtmpfs: mounted
[  4.750800] Freeing unused kernel memory: 396K (80767000 - 807ca000)
[  5.896875] udevd[175]: starting version 175
[  8.192851] EXT4-fs (mmcblk0p2): re-mounted. Opts: (null)
[  8.447207] EXT4-fs (mmcblk0p2): re-mounted. Opts: (null)
[ 11.411899] random: nonblocking pool is initialized
[ 12.843008] smsc95xx 1-1.1:1.0 eth0: hardware isn't capable of remote
[ 14.455026] smsc95xx 1-1.1:1.0 eth0: link up, 100Mbps, full-duplex, lpa
```

## The tail command

The `tail` command is designed to carry out the simple task of reading the last few lines of a file and presenting them to an output (by default, the screen). The default number of lines that the command will print out is 10, but that can be changed using the hyphen (`-n`) option.

It is complimented by the [head](#) command which will print out the *first* several lines of a file.

## Options

There are a small number of options available for `tail`. There are three that are more commonly used than others;

- `-c[num]` prints the last *num* number of bytes of the file (where 8 bytes is a character)
- `-n[num]` prints the last *num* number of lines of the file
- `-f` prints the last 10 lines of the file and if new lines are added it will output those as well

At first glance the `-c` option might look a little unusual, but what it does is provide a very exact method for printing out a specific number of letters of a file. For example if we wanted to print out the last 50 characters of the file `/var/log/dmesg` we would use the following;

```
tail -c50 /var/log/dmesg
```

The subsequent output would look as follows;

```
pi@raspberrypi /var/log $ tail -c50 /var/log/dmesg
0 eth0: link up, 100Mbps, full-duplex, lpa 0xC5E1
```

We can see that text returned is 41 characters long.

With the `-n` option we can print a specific number of lines. So, the following will print out the last 5 lines of the `/var/log/dmesg` file;

```
tail -n5 /var/log/dmesg
```

With a result that looks as follows;

```
pi@raspberrypi /var/log $ tail -n5 /var/log/dmesg
[  8.192851] EXT4-fs (mmcblk0p2): re-mounted. Opts: (null)
[  8.447207] EXT4-fs (mmcblk0p2): re-mounted. Opts: (null)
[ 11.411899] random: nonblocking pool is initialized
[ 12.843008] smsc95xx 1-1.1:1.0 eth0: hardware isn't capable of remote
[ 14.455026] smsc95xx 1-1.1:1.0 eth0: link up, 100Mbps, full-duplex, lpa
```

For either the `-c` or `-n` options we can specify a multiplier adjacent to the *num* number that will increase the number of returned characters or lines. For example `b` = 512, `kB` = 1000, `K` = 1024, `MB` = 1000x1000, `M` = 1024x1024, `GB` = 1000x1000x1000, `G` = 1024x1024x1024, and so on for `T`, `P`, `E`, `Z` and `Y`.

The following example will therefore print out the last 5000 characters of the `/var/log/dmesg` file.

```
tail -c5kB /var/log/dmesg
```

The `-f` option for `tail` provides a powerful tool for real-time monitoring of activity on a changing file. Using `-f` on a file, it will maintain a watch on the file and if additional lines are appended to the file they will be printed to the screen.

## Arguments

If we want to display the end of more than one file we can simply list their file names one after the other. For example, to list the last 5 lines of the files `dmesg` and `syslog` (both of which are in `/var/log`) we can use the following command;

```
tail -n5 dmesg syslog
```

The results will look a little like the following;

```
pi@raspberrypi /var/log $ tail -n5 dmesg syslog
==> dmesg <==
[  8.192851] EXT4-fs (mmcblk0p2): re-mounted. Opts: (null)
[  8.447207] EXT4-fs (mmcblk0p2): re-mounted. Opts: (null)
[ 11.411899] random: nonblocking pool is initialized
[ 12.843008] smsc95xx 1-1.1:1.0 eth0: hardware isn't capable of remote
[ 14.455026] smsc95xx 1-1.1:1.0 eth0: link up, 100Mbps, full-duplex, lpa

==> syslog <==
Aug 29 02:17:01 raspberrypi /USR/SBIN/CRON[2856]: (root) CMD (run-parts)
Aug 29 02:39:02 raspberrypi /USR/SBIN/CRON[2870]: (root) CMD (maxlifetime)
Aug 29 03:00:01 raspberrypi /USR/SBIN/CRON[2900]: (pi) CMD (/usr/bin/php)
Aug 29 03:00:39 raspberrypi /USR/SBIN/CRON[2899]: (CRON) info (No MTA)
Aug 29 03:09:01 raspberrypi /USR/SBIN/CRON[2917]: (root) CMD (sessionclean)
```

We can see that at the start of each file listed the name of the file is printed out and there are indeed 5 lines of text each!

## Test yourself

1. Why are the multipliers K and kB different?

# File Systems

## fdisk

`fdisk` is a command designed to manage disk partitions. This means that it allows us to view, create, resize, delete, change, copy and move partitions on a hard drive.



Let's start this off with a bit of a warning. `fdisk` is a command that will re-organise your storage and as such it carries some significant risk if done incorrectly. `fdisk` will require that it is run by a user with administrator permissions and for good reason. Please don't create, delete or modify partitions unless you know what you are doing! incorrect usage of `fdisk` may result in loss of data or trash the system.

While we will outline some of the functions of `fdisk` here we will restrict the description to allow an understanding of what `fdisk` can show us and if you are wanting to change your partitions I recommend that you seek specific advice before doing so.

- `fdisk [options] [device]` : manipulate partition tables.

The only command / option combination that we will look at in depth incorporates the `-l` option to list the disk partition tables.

```
sudo fdisk -l
```

The program will then present the information that it has on the existing partitions;

```
Disk /dev/mmcblk0: 7.4 GiB, 7948206080 bytes, 15523840 sectors
```

```
Units: sectors of 1 * 512 = 512 bytes
```

```
Sector size (logical/physical): 512 bytes / 512 bytes
```

```
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

```
Disklabel type: dos
```

```
Disk identifier: 0xa3a4d77a
```

Device	Boot	Start	End	Sectors	Size	Id	Type
/dev/mmcblk0p1		8192	131071	122880	60M	c	W95 FAT32 (LBA)
/dev/mmcblk0p2		131072	15523839	15392768	7.3G	83	Linux

```
Disk /dev/sda: 29.5 GiB, 31614566400 bytes, 61747200 sectors
```

```
Units: sectors of 1 * 512 = 512 bytes
```

```
Sector size (logical/physical): 512 bytes / 512 bytes
```

```
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

```
Disklabel type: dos
```

Disk identifier: 0xb1832c48

Device	Boot	Start	End	Sectors	Size	Id	Type
/dev/sda1		96	61747199	61747104	29.5G	c W95	FAT32 (LBA)

The information above shows that we have two different storage devices connected to the system. /dev/mmcb1k0 and /dev/sda. There is a great deal of information presented about the disks themselves in addition to information on how they are partitioned.

We can see that the device (disk) /dev/mmcb1k0 has two partitions set on it. We're told that the disk has 7.4 GiB of storage (The 'i' in GiB is an indication that the storage size is reported using factors of 1024 rather than 1000 (which would be a GB). Do not panic, this is normal.). The information on sectors, is a way of representing storage capacity and is something of a hold over from when storage was always a spinning disk of something (up to recently we would also be talking about cylinders and blocks).

/dev/mmcb1k0 is reported to be divided into two partitions (/dev/mmcb1k0p1 and /dev/mmcb1k0p2). The storage allocated to each partition is allocated to specific sectors which correspond to a particular size. The Id of the partition corresponds to *system indicators* or 'types' for the partitions. The type is also represented by a human readable name. The various 'Types include (but are not limited to);

0	Empty	24	NEC DOS	81	Minix / old Lin	bf	Solaris
1	FAT12	39	Plan 9	82	Linux swap / So	c1	DRDOS/sec (FAT
2	XENIX root	3c	PartitionMagic	83	Linux	c4	DRDOS/sec (FAT
3	XENIX usr	40	Venix 80286	84	OS/2 hidden C:	c6	DRDOS/sec (FAT
4	FAT16 <32M	41	PPC PReP Boot	85	Linux extended	c7	Syrinx
5	Extended	42	SFS	86	NTFS volume set	da	Non-FS data
6	FAT16	4d	QNX4.x	87	NTFS volume set	db	CP/M / CTOS /
7	HPFS/NTFS	4e	QNX4.x 2nd part	88	Linux plaintext	de	Dell Utility
8	AIX	4f	QNX4.x 3rd part	8e	Linux LVM	df	BootIt
9	AIX bootable	50	OnTrack DM	93	Amoeba	e1	DOS access
a	OS/2 Boot Manag	51	OnTrack DM6 Aux	94	Amoeba BBT	e3	DOS R/O
b	W95 FAT32	52	CP/M	9f	BSD/OS	e4	SpeedStor
c	W95 FAT32 (LBA)	53	OnTrack DM6 Aux	a0	IBM Thinkpad hi	eb	BeOS fs
e	W95 FAT16 (LBA)	54	OnTrackDM6	a5	FreeBSD	ee	GPT
f	W95 Ext'd (LBA)	55	EZ-Drive	a6	OpenBSD	ef	EFI (FAT-12/16
10	OPUS	56	Golden Bow	a7	NeXTSTEP	f0	Linux/PA-RISC
11	Hidden FAT12	5c	Priam Edisk	a8	Darwin UFS	f1	SpeedStor
12	Compaq diagnost	61	SpeedStor	a9	NetBSD	f4	SpeedStor
14	Hidden FAT16 <3	63	GNU HURD or Sys	ab	Darwin boot	f2	DOS secondary
16	Hidden FAT16	64	Novell Netware	af	HFS / HFS+	fb	VMware VMFS
17	Hidden HPFS/NTF	65	Novell Netware	b7	BSDI fs	fc	VMware VMKCORE
18	AST SmartSleep	70	DiskSecure Mult	b8	BSDI swap	fd	Linux raid aut
1b	Hidden W95 FAT3	75	PC/IX	bb	Boot Wizard hid	fe	LANstep
1c	Hidden W95 FAT3	80	Old Minix	be	Solaris boot	ff	BBT
1e	Hidden W95 FAT1						

Yes, there are conservatively a metric meaga-load of types there. For our very simplistic overview of `fdisk` we shouldn't be too concerned about the variety. There are quite a few specialised and some semi-historical types there so in the 'Just Enough' way of thinking we can expect to see some Type '7', 'b' and 'c' on removable media and '82' / '83' for standard storage (be prepared for some flexibility there).

## The `fdisk` command

Hard disks (or more commonly nowadays with a wide range of options available, 'storage' devices) can be divided into one or more logical disks called partitions. These divisions are described in the 'partition table' found in sector 0 of a disk. The table lists information about the start and end of each partition, information about its type, and whether it is marked bootable or not. the `fdisk` command allows us to edit the partition table and as such it has the potential to significantly affect the operation of the storage medium. As a result, the `fdisk` command is only executable by a user with administrator privileges and **we risk losing data** on the disk if we execute the command incorrectly.

Partitions can be different sizes, and different partitions may have different filesystems on them, so a single disk can be used for many purposes. Traditional hard drives have a structure defined by the terms of cylinders, heads, and sectors. Modern drives use logical block addressing (LBA) which renders this structure largely irrelevant however, the standard allocation unit for partitioning purposes is usually still the cylinder.

Linux needs a minimum of one partition to support its root file system. It can also take advantage of swap files and/or swap partitions, but as a swap partition is more efficient we will usually have a second Linux partition dedicated for swap.

On Intel compatible hardware, the Basic Input / Output System (BIOS) that boots the computer can often only access the first 1024 cylinders of the disk. As a result there can often be a third partition of a few MB (typically mounted on `/boot`), to store the kernel image and a few auxiliary files used while booting.

`fdisk` allows us to view, create, resize, delete, change, copy and move partitions on a hard drive. It is an essential tool for creating space for new partitions, organising space for new drives, re-organising old drives and copying or moving data to new disks. While `fdisk` can manipulate the partition table, this does not make the space available for use. To do this we need to format the partition with a specific filesystem using `mkfs`.

As already described in the original example we can view our partition details with the `-l` option.

To go further down the rabbit hole of manipulating partitions is something that I am hesitant to describe because it may provide the impression that it is a trivial task that anyone should try. It is not something that should be avoided, but it is something that we should learn about and practise in a safe environment before attempting it for the first time. This can be done in a controlled way using the interactive commands but without saving the changes.

Once we have identified the device that we want to partition, we can start `fdisk` as a command driven interactive utility with the `fdisk` command and the device;

```
sudo fdisk /dev/sda
```

The response is a welcome message and a warning;

```
Welcome to fdisk (util-linux 2.25.2).
```

```
Changes will remain in memory only, until you decide to write them.
```

```
Be careful before using the write command.
```

```
Command (m for help):
```



We should note that we can make plenty of *potential* changes using `fdisk`'s commands, but these do not become operative until we save them using 'w'. This is a good way to practice using the commands without worrying about how they might affect the storage (so long as we don't enact the changes).

Pressing 'm' will show the range of possible commands;

Help:

DOS (MBR)

- a toggle a bootable flag
- b edit nested BSD disklabel
- c toggle the dos compatibility flag

Generic

- d delete a partition
- l list known partition types
- n add a new partition
- p print the partition table
- t change a partition type
- v verify the partition table

Misc

- m print this menu
- u change display/entry units
- x extra functionality (experts only)

Save & Exit

- w write table to disk and exit
- q quit without saving changes

Create a new label



```
g   create a new empty GPT partition table
G   create a new empty SGI (IRIX) partition table
o   create a new empty DOS partition table
s   create a new empty Sun partition table
```

To add a new partition we would press 'n' which will ask what type of partition we want to set up;

Partition type

```
p   primary (0 primary, 0 extended, 4 free)
e   extended (container for logical partitions)
```

Select (default p):



The partition table is located in the master boot record (MBR) of a disk. The MBR is the first sector on the disk, so the partition table is quite small. This limits the number of primary partitions on a disk to four. When more than four partitions are required (which is pretty normal) one of the primary partitions must become an extended partition. An extended partition is a container for one or more *logical* partitions. In this way, you can have more than four partitions on a drive using the MBR layout.

Making the assumption (in this case) that we will add a primary partition we can enter 'p' and we are asked which partition number we want to create;

Partition number (1-4, default 1):

Then we are asked where the first sector of our partition should start from;

First sector (2048-61747199, default 2048):

Then we are asked what the last sector will be;

Last sector, +sectors or +size{K,M,G,T,P} (2048-61747199, default 61747199):

Once complete, `fdisk` will tell us the details of the partition that it has set up;

Created a new partition 1 of type 'Linux' and of size 29.5 GiB.

If this was our desired result we might write the changes and the configuration would be stored in the partition table. Again, this is something to be studied and understood before trying for real.

## Test yourself

1. How many primary partitions can be created on a device?
2. What 'type' of partition is 83?
3. How can you practice using the `fdisk` command without implementing the changes?

## mkfs

The `mkfs` command is used to create a file system on a Linux partition. It is the process of applying *high level* formatting to a partition such that a particular file system is set up and the storage is made ready for use. This is an important step in adding storage but carries with it a degree of caution which should accompany any command that is making changes to storage. Incorrect usage could result in **loss of data**. As expected with this type of command we would need to be an administrative user to execute it.

- `mkfs [options] device` : format a partition with a file system

The usage indication above is slightly simplified from that shown in the [man](#) pages, but it represents the expected usage scenarios for 'Just Enough'.

Having identified or created a partition using `fdisk` we can create a formatted partition as follows (assuming that the partition we want to format is `/dev/sda1` and this will use the default 'ext2' file system);

```
sudo mkfs /dev/sda1
```

In our usage case we already have a formatted partition on `/dev/sda1` and we receive the following notification;

```
mke2fs 1.42.12 (29-Aug-2014)
/dev/sda1 contains a vfat file system labelled 'SP UFD U2'
Proceed anyway? (y,n)
```



Once we proceed past this point we will be making some serious changes to our storage!

Having done due deliberation if we decide to proceed and press 'y' the process begins by printing out details of the filesystem and then writing the inode tables;

```
Creating filesystem with 7718388 4k blocks and 1933312 inodes
Filesystem UUID: 5d930bad-3024-476b-8673-4b450855f537
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
    4096000
```

```
Allocating group tables: done
Writing inode tables: 177/236
```

Once complete it will start writing the superblock and filesystem information;

Writing superblocks and filesystem accounting information: 10/236

Be warned, this output can sometimes look like it has frozen. There are certainly reports of it online and I have seen it myself. However, be a little patient (perhaps 10 minutes or so) and it should complete.

Once complete all that remains is to [mount](#) the partition and we can use the freshly formatted storage.

## The `mkfs` command

File systems are used to control how data is stored on our drives and partitions, what other information is attached to the data itself and how access to data is controlled. File systems are being improved all the time to include more functionality and efficiency.

The `mkfs` command is used to format partitions as a particular file system type. The command is short for **make file system**. The `mkfs` command is actually a front end to several file system-specific commands. Not all commands for all file systems are supported on all distributions. We can find which file systems are available on our system by checking in our `/sbin` directory using `ls` as follows;

```
ls /sbin/mk* -l
```

This should show something similar to the following;

```
lrwxrwxrwx 1 root root      8 Nov 21  2014 /sbin/mkdosfs -> mkfs.fat
-rwxr-xr-x 1 root root 100688 Mar  2  2015 /sbin/mke2fs
-rwxr-xr-x 1 root root   9704 Apr  5  2015 /sbin/mkfs
-rwxr-xr-x 1 root root  22084 Apr  5  2015 /sbin/mkfs.bfs
-rwxr-xr-x 1 root root  30316 Apr  5  2015 /sbin/mkfs.cramfs
lrwxrwxrwx 1 root root      6 Mar  2  2015 /sbin/mkfs.ext2 -> mke2fs
lrwxrwxrwx 1 root root      6 Mar  2  2015 /sbin/mkfs.ext3 -> mke2fs
lrwxrwxrwx 1 root root      6 Mar  2  2015 /sbin/mkfs.ext4 -> mke2fs
lrwxrwxrwx 1 root root      6 Mar  2  2015 /sbin/mkfs.ext4dev -> mke2fs
-rwxr-xr-x 1 root root  30648 Nov 21  2014 /sbin/mkfs.fat
-rwxr-xr-x 1 root root  30364 Apr  5  2015 /sbin/mkfs.minix
lrwxrwxrwx 1 root root      8 Nov 21  2014 /sbin/mkfs.msdos -> mkfs.fat
lrwxrwxrwx 1 root root      8 Nov 21  2014 /sbin/mkfs.vfat -> mkfs.fat
-rwxr-xr-x 1 root root  17912 Jan 24 11:30 /sbin/mkhomedir_helper
-rwxr-xr-x 1 root root  55164 Apr  5  2015 /sbin/mkswap
```

Here we can see that there are a range of different file system types supported and in some cases there are [links](#) established to ensure that a specific command will direct to an appropriate program.

The main option that we should therefore be aware of when using `mkfs` is the `-t` option which allows us to specify the file system 'type'.

While the default use of `mkfs` will implement the `ext2` file system, we can specify others as follows;

- `ext3`: `sudo mkfs -t ext3 device`
- `ext4`: `sudo mkfs -t ext4 device`
- `msdos`: `sudo mkfs -t msdos device`
- `bfs`: `sudo mkfs -t bfs device`

Some quick points of interest on the three main file system players that are used in Linux;

- `ext2` :
  - can handle up to 4TB
  - the superblock feature increase file system performance
  - `ext2` reserves 5% of disk space for root
  - `ext2` is popular on USB and other solid-state devices because it does not have a journaling function so it typically makes fewer reads and writes to the drive extending the life of the device.
- `ext3` :
  - provides all the feature of `ext 2` but includes journaling and backward compatibility
  - can upgrade `ext2` to `ext3` without loss of data.
  - journaling feature speeds up the system to recover state after power-failure or improper mount unmount etc.
- `ext4`:
  - supports larger filesystem, faster checking, nanosecond timestamps, and verification of the journal through checksums.
  - backwards and forwards compatible with versions 2 and 3
  - faster time-stamping
  - faster file system checking
  - journaling check-sums
  - automatic space allocation to avoid fragmentation

## Test yourself

1. How can you find out what file systems are supported on your operating system?
2. What is the default file system used with `mkfs`?

## mount

Once a storage device (CD Rom, Hard drive, USB Stick, etc) has been [partitioned](#) for use we need to use the `mount` command to mount it to a filesystem, making it accessible and attaching it to an existing directory structure. The command is seldom required for GUI based systems that automate the process, but for manually setting up file systems or troubleshooting, an understanding of the `mount` command is highly useful.

- `mount [options] type device directory` : mount storage onto the file system

For example, after plugging in a USB memory stick we can run the following commands to mount the storage;

```
sudo mkdir /mnt/usbddata
sudo mount /dev/sda1 /mnt/usbddata
```

The process has proceeded in two steps.

1. First we created an empty directory using `mkdir` called `usbddata` which gives us a place to mount the storage
2. Then we mount the storage which is represented as a [device](#) in `/dev/sda1` onto our new directory (`/mnt/usbddata`).

From here we can `cd` into the directory and utilise the storage to our hearts content.



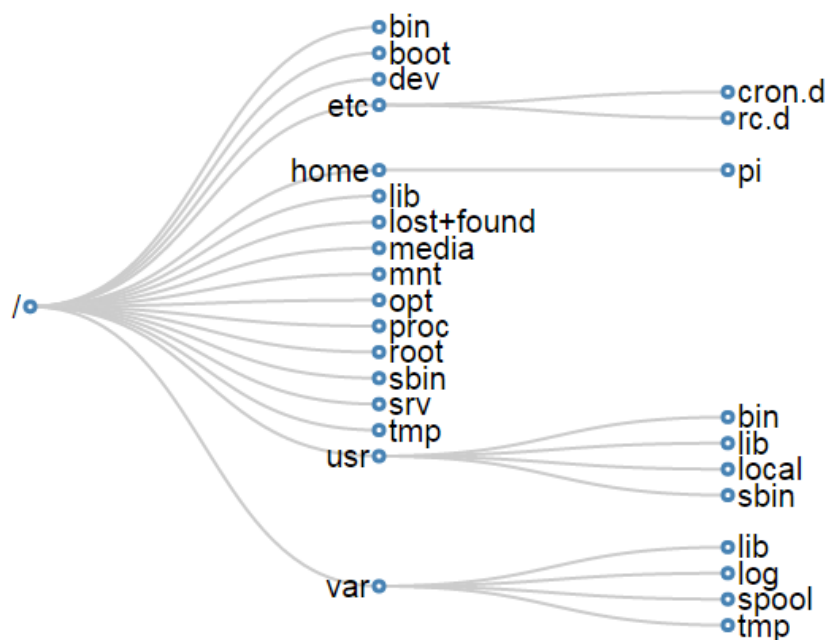
Things to note:

This is not persistent storage. In other words it will not automatically reappear if the computer is turned off and restarted (or rebooted). This is a pretty simple example and there is a wide range of different storage types that can be mounted in a wide range of ways.

## The `mount` command

The `mount` command is one of those that when you start using it confidently on a regular basis you will probably be in a pretty comfortable place administering Linux.

The Linux filesystem is a [hierarchy of directories](#) that is used to organize our files in a logical order similar to the following;



Directory Hierarchy

When we mount our new storage we create a directory to associate it with its mounting point. Convention has us putting this in the '/mnt' directory, but we can [link](#) to that position from another point if we want. In this way we can make the storage appear in a very flexible way.

To mount new storage we should know what type of storage it is, where it is presented to the operating system as a device and where we are going to mount it.

There is a long list storage types which can be supported. These can be found on the mount man page. They could include; adfs, affs, autofs, btrfs, cifs, coda, coherent, cramfs, debugfs, devpts, efs, ext, ext2, ext3, ext4, hfs, hfsplus, hpfs, iso9660, jfs, minix, msdos, ncfs, nfs, nfs4, ntfs, proc, qnx4, ramfs, reiserfs, romfs, squashfs, smbfs, sysv, tmpfs, ubifs, udf, ufs, umsdos, usbfs, vfat, xenix, xfs or xiafs. The range is dependant on the support for the type in the Linux kernel that is being run (in other words, it varies).

Bear in mind that we're not deciding *what* type to use when we implement the mount command. The type is set by the configuration of the storage medium.

The example we used earlier in this section didn't specify a type. The good news is that mount can often auto-detect the type making our lives easier. Some of the most common and most likely to be detected automatically are;

- ext4 - this is probably the most common, recent Linux filesystem type
- ext3 - the previous most common Linux filesystem type
- ntfs - a common Windows filesystem for larger external hard drives
- vfat - a common Windows filesystem for smaller external hard drives

If a filesystem type doesn't auto-detect this could be as a result of;

- Not having filesystem tools installed for the chosen filesystem. For example, to mount ntfs drives in older versions of Ubuntu we would have needed to install the ntfs-3g package.

- We may have selected the wrong partition.
- The storage partition may be corrupt or unformatted.

When new storage is introduced to the computer, it should appear in the `/dev` directory. To the uninitiated, it will not be immediately obvious which device is the one to use since there will be quite a few in there.

Removable drives and standard hard drives are most likely to be represented as 'SCSI disks' which is abbreviated to 'sd'. The first device that fits the 'SCSI disk' criteria will be designated 'sda' the second 'sdb' and so on. The drive should be partitioned for use as part of the formatting process and as such it will have at least one partition on it. This first partition is designated by a numerical value so that our device could be 'sda1'. Additional partitions would be 'sda2', 'sda3' etc.

It would be possible to list (`ls`) the contents of the `/dev` directory before adding the storage and after adding it which could indicate the device name, but a far more dependable method is to use the `fdisk` command to list the available partitions. This we can do with the following command;

```
sudo fdisk -l
```

The output can be relatively lengthy depending on the installed storage, but here we can see the resulting information from plugging in a 32GB USB stick (just the USB stick information is shown);

```
Disk /dev/sda: 29.5 GiB, 31614566400 bytes, 61747200 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xc3072e18
```

Device	Boot	Start	End	Sectors	Size	Id	Type
/dev/sda1		96	61747199	61747104	29.5G	c	W95 FAT32 (LBA)

Sure enough we can see the `/dev/sda1` identifier that is specific to the device.

The last step before mounting is to create a mount point for the storage. We do this by simply creating an empty directory. By convention the `/mnt` directory is used for mounting storage devices like CDRoms and USB drives and the ultimate location of the mount point will need to be left to the users needs.

Looking again at our earlier example;

```
sudo mkdir /mnt/usldata  
sudo mount /dev/sda1 /mnt/usldata
```

We make a directory that will serve as our mount point in the /mnt directory called usldata. We then instruct the computer to mount the device at /dev/sda1 onto the directory /mnt/usldata. The command does not specify a type and therefore we will be using the default.

Once done we can navigate to the directory to explore it. We can also confirm its existence by using the mount command on its own. If we just run the command mount as follows;

```
mount
```

We will be presented with a list of mounts present on the system. The reply that is specific to our recently mounted drive will look something like the following;

```
/dev/sda1 on /mnt/data type vfat
```

This tells us the device, the mount point and the type (vfat). There may be additional information that will describe how the mounted storage is configured. While interesting, we will decline to explore this as the book is called '*Just Enough Linux*', not '*In Depth Linux*'.

No description of the mount command could fail to mention the /etc/fstab file. fstab stands for file system **table** and the fstab file contains the information that automates the process of mounting partitions. Typically this file will contain entries for internal devices such as CDROMs or network shares. Ones that have an expectation for mounting on boot. It can however contain information that will describe how a mount is to be made, but specify 'noauto' that will prevent it occurring automatically at boot. This then allows the use of the mount command with the -a option which will mount all the file systems specified in fstab. (assuming that the directory mount point exists).

```
mount -a
```

## Test yourself

1. If you plugged in two external USB drives, what could be their likely device designations and where could this be checked?
2. If you use the mount command to mount a drive, will it appear mounted again after a reboot?



## umount

Once a storage device (CD Rom, Hard drive, USB Stick, etc) has been mounted for use we need to use the `umount` command to unmount it from a filesystem. Unmounting a filesystem cleanly is good practice and could possibly reduce the chance of data not being saved correctly to the storage if a process hasn't completed. The command would typically only be required after mounting a drive using `mount`.

- `umount [options] device and/or directory` : unmount storage from the file system

For example, after plugging in a USB memory stick and `mounting` it from the device `/dev/sda1` onto `/mnt/usbdata` we can run the following command to unmount the storage;

```
sudo umount /mnt/usbdata
```

## The `umount` command

The `umount` command is used to manually unmount filesystems on Linux operating systems. This is the reverse process to that carried out with the `mount` command.

If we recall the `mount` command we specified the type of storage, the device name (in the `/dev` directory) and where we wanted to mount it.

A typical usage might look as follows;

```
sudo mkdir /mnt/usbdata
sudo mount /dev/sda1 /mnt/usbdata
```

Here we have made a directory that will serve as our `mount` point in the `/mnt` directory called `usbdata`. We then instruct the computer to `mount` the device at `/dev/sda1` onto `/mnt/usbdata`. We have **not** provided a type since typically the `mount` can automatically work it out for us.

Unmounting a file system can be substantially easier since the process of `mounting` a device has done the hard work in terms of specifying the configuration details. That means that we only need to supply the device name (`/dev/sda1` in this case) or the mount point (`/mnt/usbdata`). We can carry out the command with this little information since both the device name and `mount` point are both unique to the mounted storage.

Therefore all three of the following commands are the equivalent of each other;

```
sudo umount /dev/sda1
sudo umount /mnt/usbdata
```

```
sudo umount /dev/sda1 /mnt/usbdata
```

Mounted filesystems are automatically unmounted when a computer is shut down in a normal manner. However, there are times when an individual filesystem needs to be unmounted while a computer is still running. A typical example is when removing an external device such as a USB stick. If we remove this type of storage before the filesystem on it is properly unmounted, any data recently written to it might not be saved correctly.

Unmounting a filesystem can sometimes result in errors being presented on the screen. The most common report is that the filesystem is busy. This would be an indication that it is currently being used by some process. This could occur if a file that was on the filesystem was open or even something as simple as a window in a desktop GUI showing files being displayed. This can be remedied by closing any open files or windows.

## Test yourself

1. Will the directory that the filesystem was mounted on remain in place if the filesystem is unmounted?

# System Information

## date

The `date` command can be used to display the date / time that the computer is set to or to set the systems date / time. This command can be useful simply to tell the date or time or for a more serious purpose to set the systems configuration.

- `date [options] +format` : display or set the date / time

The simplest usage of the `date` command is found by executing the command by itself;

```
date
```

This will result in something like the following being output;

```
Sat Jan 23 15:26:14 NZDT 2016
```

At heart the `date` command has a simple usage model, but it is accompanied by some considerable formatting options.

## The `date` command

The `date` command is so named because of its association with **date** and time. As such its usage is centred around a couple of main functions. The first is that it can be invoked to report the time. This can be formatted in a user specified manner. The second major function is to allow the user to set the date / time of the system.

Associated with the reporting and setting date / time is the formatting of each in the command. The display formatting can be affected by use of the plus (+) symbol and then some formatting directions in the form of a string of options. For example to print out only the current month using its full name we can use the format control `%B` as follows;

```
date +"%B"
```

This will result in something like the following being output;

```
January
```

These display options can be combined to create meaningful context. For example;

```
date +"This is week %U of %Y"
```

Will display the text “This is week ” followed by the week number of the year and then the text “ of ” and then the full year. Similar to the following;

```
This is week 04 of 2016
```

The list of format controls is as follows;

- %% a literal %
- %a locale’s abbreviated weekday name (e.g., Sun)
- %A locale’s full weekday name (e.g., Sunday)
- %b locale’s abbreviated month name (e.g., Jan)
- %B locale’s full month name (e.g., January)
- %c locale’s date and time (e.g., Thu Mar 3 23:05:25 2005)
- %C century; like %Y, except omit last two digits (e.g., 20)
- %d day of month (e.g., 01)
- %D date; same as %m/%d/%y
- %e day of month, space padded; same as %\_d
- %F full date; same as %Y-%m-%d
- %g last two digits of year of ISO week number (see %G)
- %G year of ISO week number (see %V); normally useful only with %V
- %h same as %b
- %H hour (00..23)
- %I hour (01..12)
- %j day of year (001..366)
- %k hour, space padded ( 0..23); same as %\_H
- %l hour, space padded ( 1..12); same as %\_I
- %m month (01..12)
- %M minute (00..59)
- %n a newline
- %N nanoseconds (000000000..999999999)
- %p locale’s equivalent of either AM or PM; blank if not known
- %P like %p, but lower case
- %r locale’s 12-hour clock time (e.g., 11:11:04 PM)
- %R 24-hour hour and minute; same as %H:%M
- %s seconds since 1970-01-01 00:00:00 UTC
- %S second (00..60)
- %t a tab
- %T time; same as %H:%M:%S

- %u day of week (1..7); 1 is Monday
- %U week number of year, with Sunday as first day of week (00..53)
- %V ISO week number, with Monday as first day of week (01..53)
- %w day of week (0..6); 0 is Sunday
- %W week number of year, with Monday as first day of week (00..53)
- %x locale's date representation (e.g., 12/31/99)
- %X locale's time representation (e.g., 23:13:48)
- %y last two digits of year (00..99)
- %Y year
- %Z +hhmm numeric time zone (e.g., -0400)
- %:z +hh:mm numeric time zone (e.g., -04:00)
- %::z +hh:mm:ss numeric time zone (e.g., -04:00:00)
- %:::z numeric time zone with : to necessary precision (e.g., -04, +05:30)
- %Z alphabetic time zone abbreviation (e.g., EDT)

Because the default action of `date` is to pad numeric fields with zeros, we also have a range of optional flags which can follow the % sign to modify the format controls. Those optional flags are;

- - (hyphen) do not pad the field
- \_ (underscore) pad with spaces
- 0 (zero) pad with zeros
- ^ use upper case if possible
- # use opposite case if possible

For example in the earlier output the week number of the year was returned as 04. If we wanted to eliminate any leading zeros in that figure we could use the command as follows;

```
date +"This is week %-U of %Y"
```

Which will display the following;

```
This is week 4 of 2016
```

## Options

While there are a few options available, the most useful option when using `date` are;

- -d display time as specified by an accompanying string
- -s sets the time
- -u sets or prints Coordinated Universal Time (UTC)

By default the use of `date` with no options will report back whatever the current time is. However, the `-d` option allows us to request that the computer display a date specified by a user defined string. For example to display the date for the fourth of July 2001 we could enter the command;

```
date -d "4 JUL 2001"
```

Which will display the following;

```
Wed Jul 4 00:00:00 NZST 2001
```

We could be forgiven for thinking that this is a neat trick, but unless we really wanted to know that the fourth of July 2001 was a Wednesday, it had little use. However, the user defined string can include more human readable descriptions that could be useful. For example we could enter any one of the following strings and get a valid output;

```
date -d "now"  
date -d "next thursday"  
date -d "yesterday"  
date -d "last year"  
date -d "4 days ago"  
date -d "today - 1 year"  
date -d "yesterday - 1 year"
```

Setting the time is accomplished with the `-s` option. Because this is a function that can have some pretty far reaching consequences for the system as a whole it is restricted to those with administrator privileges. Therefore we need to prefix the date command with `sudo` or to execute the command as the root user.

The following command will set the date and time to the 24th of January 2016 at 7:12;

```
sudo date -s "24 Jan 2016 07:12:00"
```

There is quite a degree of flexibility in the formatting of the time that is specified, but because the command can be clever enough to adjust some things and not others or to take the command at it's literal face value, it is better to be as precise as possible.

Coordinated Universal Time (which is abbreviated as UTC), is the primary time standard by which the world regulates clocks and time. It was formerly known as Greenwich Mean Time (GMT) and while UTC is considered interchangeable with GMT, it is no longer precisely defined by the scientific community. Likewise 'Zulu' time is a description of GMT using the phonetic representation of 'Z'. The use of UTC is common in aviation and military circles since it represents a common reference point for time on a global scale.

It is also possible to display UTC time using the `-u` option. (While it can also be set using this option this would not be something that someone reading a book called ‘Just Enough Linux’ should probably do).

## Test yourself

1. Format a `date` command to report back the date time showing the full name of the day of the week, the number of the week in the year (without leading zeros) and the year in the format “Sunday of week 4 of 2016”
2. Use the `date` command to report back Zulu time in seconds since the Unix Epoch (you will probably need to Google this to find out what it is).

## df

The `df` command is used to report the amount of free space available on file systems. This is a vital command for knowing about the resources available on a system and one of the ‘go-to’ commands for troubleshooting systems where storage volume might be limited.

- `df [options] file` : display the amount of free space on filesystems.

For example, using the `df` command on its own (without any options) might produce something like the following;

```
df
```

Produces ...

Filesystem	1K-blocks	Used	Available	Use%	Mounted on
/dev/root	7549084	1055236	6159348	15%	/
devtmpfs	469748	0	469748	0%	/dev
tmpfs	474052	0	474052	0%	/dev/shm
tmpfs	474052	6356	467696	2%	/run
tmpfs	5120	4	5116	1%	/run/lock
tmpfs	474052	0	474052	0%	/sys/fs/cgroup
/dev/mmcblk0p1	61384	20296	41088	34%	/boot

The first column shows the name of the disk partition as it appears in the `/dev` directory. The following columns show total space, blocks allocated and blocks available. The capacity column indicates the amount used as a percentage of total file system capacity.

The final column shows the [mount](#) point of the [file system](#). This is the directory where the file system is mounted within the [file system](#) tree. Note that the root partition will always show a mount point of `/`.

## The `df` command

The `df` command is named for the abbreviation of the words ‘disk free’ and it carries out this exact role by reporting the amount of used and available space on [mounted](#) file systems.

It is a truism of computing that storage needs increase to fit the space available, so with that in mind it pays us to have a set of tools available that can determine the amount of volume we have on any particular piece of connected storage. For this reason alone, having a good working knowledge of `df` and it’s erstwhile companion `du` is highly recommended.

While there are several options available, the most useful option when using `df` is `-h` which will report the space in human readable format so that ...



```
df -h
```

... will produce ...

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/root	7.2G	1.1G	5.9G	15%	/
devtmpfs	459M	0	459M	0%	/dev
tmpfs	463M	0	463M	0%	/dev/shm
tmpfs	463M	6.3M	457M	2%	/run
tmpfs	5.0M	4.0K	5.0M	1%	/run/lock
tmpfs	463M	0	463M	0%	/sys/fs/cgroup
/dev/mmcblk0p1	60M	20M	41M	34%	/boot

Here we can see that the sizes for the file systems are reported as kilobytes (K), megabytes (M) and gigabytes (G).

We can also be specific in reporting the free space on a particular device by providing the path to the directory in the command;

```
df /
```

... will produce ...

Filesystem	1K-blocks	Used	Available	Use%	Mounted on
/dev/root	7549084	1055592	6158992	15%	/

We can also determine the space available on a filesystem by specifying a particular directory and the command will report back the space remaining in the device that contains the particular directory;

```
df /home
```

... will therefore produce ...

Filesystem	1K-blocks	Used	Available	Use%	Mounted on
/dev/root	7549084	1055592	6158992	15%	/

Note that the returned values for this command are the same as for `df /`. That's because the `/` and the `/home` directory exist on the same mounted device (`/dev/root`)

`df` gets most of its information from a filesystem's superblock (which is a record of the characteristics of a filesystem) and as a result it takes this information at face value and doesn't include open files on disk, but will include open files in memory and data / index files used for data management. As a result, `df` cannot be relied on to show an entirely precise indication of free space.

### Test yourself

1. What would an available value of space given as 659348 (using the standard `df` command) be if the `-h` switch was used?

## du

The `du` command is used to report the amount of space used by files and directories. This is a vital command for knowing about the resources available on a system and one of the ‘go-to’ commands for troubleshooting systems where storage volume might be limited.

- `du [options] file` : display the amount space used in files and directories

For example, using the `du` command to examine the space used in the `/var/log` directory and its subdirectories as follows...

```
du /var/log
```

... produces ...

```
12      /var/log/fsck
4        /var/log/ntpstats
48      /var/log/apt
4        /var/log/samba
2932    /var/log
```

The first column shows the number of disk blocks used in each directory (remembering of course that `/var/log` is the lowest directory and the others are subdirectories of it). The second column is the directory being reported on.

## The `du` command

The `du` command is named for the abbreviation of the words ‘disk usage’ and it reports the sizes of directories, directory trees and files.

It is a truism of computing that storage needs increase to fit the space available, so with that in mind it pays us to have a set of tools available that can determine the amount of volume we have on any particular piece of connected storage. For this reason alone, having a good working knowledge of `du` and its erstwhile companion `df` is highly recommended.

While there are several options available, the most useful option when using `du` are;

- `-h` provides a **human** readable format of volume
- `-s` provides a **s**umarised total of disk usage
- `--time` includes the last modification time in the output

Entering the same command that we executed for the `/var/log` directory, but with the `-h` option we are presented with a different reporting of the sizes of the directories;

```
du -h /var/log
```

... produces ...

```
12K    /var/log/fsck
4.0K   /var/log/ntpstats
48K    /var/log/apt
4.0K   /var/log/samba
3.1M   /var/log
```

Here we can see that the sizes for the directories are reported as kilobytes (K), megabytes (M). They can also be represented as gigabytes (G), terabytes (T), petabytes (P), exabytes (E), zettabytes (Z) and yottabytes (Y).



For the purposes of illustration, it would seem unlikely to see yottabytes commonly used in the immediate future given that they equate to 1,000,000,000,000,000,000,000 bytes. Although I'd be comfortable being proved wrong on this point.

Producing a summarised reading of disk used with the `-s` option allows us to return only a specific reading. For example;

```
du -s /var/log
```

... produces ...

```
3152    /var/log
```

Since this command is enormously useful when fault finding it is also useful to get an indication of the last modification time of a directory. This helps because if we have a process that is writing to a directory and filling it up, sometimes the largest directory won't be the scene of the crime, but the most recently written to one is. This check can be accomplished with the `--time` option;

```
df --time /var/log
```

... will produce ...

```
pi@raspberrypi:~ $ du --time /var/log
12      2015-11-22 07:51      /var/log/fsck
4        2015-11-02 17:31      /var/log/ntpstats
48       2016-01-16 16:41      /var/log/apt
4        2015-03-08 01:13      /var/log/samba
3152     2016-01-23 07:52      /var/log
```

A restriction of `du` is that the sizes of directories and files it reports are not exact numbers and are only approximations. However, this does not detract from its role in determining where problems might be occurring. The other restriction is that `du` can only be used to estimate used space for directories and files for which the user has reading permission. We may in some cases need to use `sudo` or be a user with system administrator privileges to execute the command successfully (in some cases).

## Test yourself

1. What options would be used with `du` to return a summarised indication of space used with modification time displayed?

## free

The `free` command shows the total amount of free and used physical and swap memory in the system, as well as the buffers used by the kernel and shared memory. In the same way that `df` and `du` can report how storage space on physical storage media is being utilised, the `free` command can show how the memory (ram) is being used. This is another useful command for troubleshooting and understanding system performance.

- `free [options]` : displays information about free and used memory on the system

For example, if we type in the command as follows we will see an interesting array of information;

```
free
```

... produces ...

	total	used	free	shared	buffers	cached
Mem:	948108	104116	843992	6364	9720	56040
-/+ buffers/cache:		38356	909752			
Swap:	102396	0	102396			

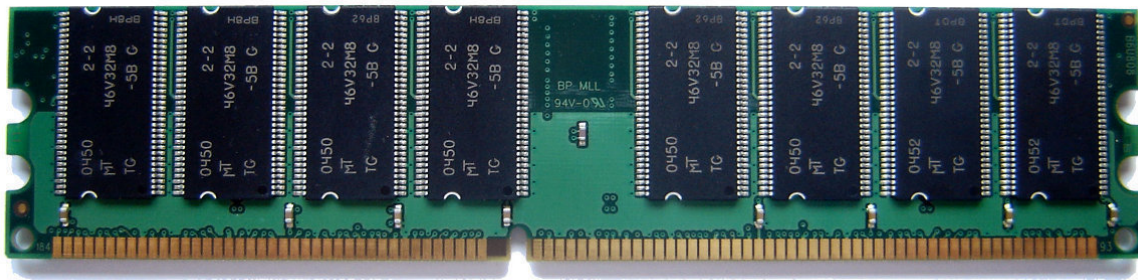
This is displaying the total amount of free and used physical and swap memory in the system, as well as the buffers used by the kernel. The row labelled 'Mem', also displays the amount of memory allocated to buffers, caches and how much memory is shared.

The `free` command output can be easily misinterpreted as it is not immediately obvious what memory 'activity' is being reported on, so it is useful to take a little time to get a good grounding in the terms to form a better understanding.

## The free command

The `free` command is named for its ability to report on the amount of **free** memory is available on a system. But what should be obvious from the example output above, is that memory is used in many different ways.

Firstly, let's be clear about what we're referring to when we talk about memory. Specifically this is memory used when the computer is running applications / operating system and generally processing data. It is concerned mainly with the allocation of **Random Access Memory (RAM)** as opposed to memory in the form of hard drives or similar that will persistently store information. However, it includes some memory that will exist on hard drives to supplement RAM.



RAM Stick

To best understand the output we should consider it line by line and examine the various row / column combinations.

### Mem

The Mem line is referring to the physical memory available on the system

### Buffers / Cache

A buffer, also called buffer memory, is usually defined as a portion of memory that is set aside as a temporary holding place for data that is being sent to or received from an external device, such as a HDD, keyboard, printer or network. Cache is a memory location to store frequently used data for faster access. Cache data can be used multiple times whereas buffer memory is used only once. And both are temporary stores for your data processing.

This line is especially relevant with regard to applications, as data accessed from files pass through the cache. Hence, the cache can speed up access to data by reducing the requirement to read/write from/to the hard drive or other storage (given that accessing hard drives and similar is slower than RAM).

### Swap

Swap refers to memory that has been allocated on a hard drive or similar storage to help the system cope in situations where it has run out of RAM. Using swap space can be an indication that a system could require more resources, although sometimes this need can be extremely infrequent and the use could be thought of as an acceptable trade off to adding more RAM that might not get used very often.



Be aware, that when swap memory is used, the amount shown by free will increase, but even when it is finished, the 'used' amount will still show that higher value (since the space that the memory is referring to has technically been used). This can be confusing and could possibly lead us to think that there is a constant usage of swap memory when in fact it was momentary.

### Total

The total column refers to the total amount of memory available in either the RAM or from the hard drive (or similar) as shared memory.

### Used

The 'used' column is potentially the most confusing. This is a mixture of memory in use for applications and other 'temporarily' (buffer + cache) used memory that is available if needed. So while the memory is being used for the buffers / cache area, much of this memory is available if required by an application. The 'temporarily' used memory is borrowed if available to help improve performance, otherwise the system would have to read from disk more often. Much of this type of memory is shown under the 'cached' column.

## Free

This is free memory ready to be used when required.

## Shared

Shared memory is a method of interprocess communication whereby several processes share a single chunk of memory to communicate. The shared column lists the amount of memory shared between those multiple processes. It is often listed as obsolete, but the shared memory column represents either the MemShared value (in 2.4 series kernels) or the Shmem value (in 2.6 series kernels and later) taken from the /proc/meminfo file. The value is zero if none of the entries is exported by the kernel.

## Buffers

The buffers column shows the amount of memory used by the kernel buffer cache. The buffer cache is used to speed up disk operations, by allowing disk reads and writes to be serviced directly from memory. The buffer cache size will increase or decrease as memory usage on the system changes. This memory is reclaimed if it is needed by applications.

## Cached

The cache column tells us how many memory pages the kernel has cached for faster access later. Since the memory used for buffers and cache can easily be reclaimed for use by applications, the -/+ buffers/cache row provides an indication of the memory *actually* used by applications (the used column) or available to applications (the free column). The sum of the memory used by buffers and cache reported in the Mem row is subtracted from the total used memory and added to the total free memory to give the used and free values on the two figures on the -/+ buffers/cache row.

	total	used	free	shared	buffers	cached
Mem:	948108	104116	843992	6364	9720	56040
-/+ buffers/cache:		38356	909752			
Swap:	102396	0	102396			

$$\begin{array}{r}
 104116 \\
 - (9720 + 56040) \\
 \hline
 = 38356
 \end{array}$$

Calculation of used buffers/cached memory



	total	used	free	shared	buffers	cached
Mem:	948108	104116	843992	6364	9720	56040
-/+ buffers/cache:		38356	909752			
Swap:	102396	0	102396			

$$\begin{array}{r}
 843992 \\
 + (9720 + 56040) \\
 \hline
 = 909752
 \end{array}$$

Calculation of free buffers/cached memory

## Options

While there are several options available, the most useful option when using `free` is `-h` which will report the space in **human readable format** so that ...

```
free -h
```

... will produce ...

	total	used	free	shared	buffers	cached
Mem:	925M	101M	823M	6.2M	10M	54M
-/+ buffers/cache:		36M	888M			
Swap:	99M	0B	99M			

Here we can see that the sizes for the amount of free memory is reported in megabytes (M). They can also be represented as bytes (B), kilobytes (K), gigabytes (G) and terabytes (T).

`free` will report slightly less memory being in a computer than the *actual* total. This is predominantly because the kernel always remains in main memory while the computer is in operation, and thus the memory that it occupies can never be freed. There can also be regions of memory that are reserved for other purposes, according to the specific system architecture.

## Test yourself

1. The total amount of used swap is high with very little free. Is this bad? What if you saw exactly the same figure 24 hours after first noticing it?
2. Is 'shared' memory even relevant nowadays?

# Processes

## crontab

The `crontab` command give the user the ability to schedule tasks to be run at a specific time or with a specific interval. If you want to move beyond using Linux from a graphical user interface, you will most likely want to schedule a task to run at a particular time or interval. Even just learning about it might give you ideas of what you might do.

- `crontab [-u user] [-l | -r | -e]` : Schedule a task to run at a particular time or interval

For example, you could schedule a script to run every day to carry out a backup process in the middle of the night. or capture some data every hour to store in a database.

## The `crontab` command

The command `crontab` is a concatenation of ‘cron table’ because it uses the job scheduler `cron` to execute tasks which are stored in a ‘table’ of sorts in the users `crontab` file. `cron` is named after ‘Khronos’, the Greek personification of time.

While each user who sets up a job to run using the `crontab` creates a `crontab` file, the file is not intended to be edited by hand. It is in different locations in different flavour of Linux distributions and the most reliable mechanism for editing it is by running the `crontab -e` command. Each user has their own `crontab` file and the root user can edit another users `crontab` file. This would be the situation where we would use the `-u` option, but honestly once we get to that stage it can probably be assumed that we know a fair bit about Linux.

There are only three main options that are used with `crontab`.

## Options

The first option that we should examine is the `-l` option which allows us to list the `crontab` file;

```
crontab -l
```

Once run it will list the contents of the `crontab` file directly to the screen. The output will look something like;

```
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command

*/10 * * * * /usr/bin/php /home/pi/scrape-books.php
```

Here we can see that the main part of the file (in fact everything except the final line) is comments that explain how to include an entry into the crontab file.

The entry in this case is specified to run every 10 minutes and when it does, it will run the PHP script `scrape-books.php` (we'll explain how this is encoded later in the examples section).

If we want to remove the current crontab we can use the `-r` option. Probably not something that we would do on a regular basis, as it would be more likely to be editing the content rather than just removing it wholesale.

Lastly there is the option to edit the crontab file which is initiated using `-e`. This is the main option that would be used and the one we will cover in detail in the examples below.

## Examples

As an example, consider that we wish to run a Python script every day at 6am. The following command will let us edit the crontab;

```
crontab -e
```

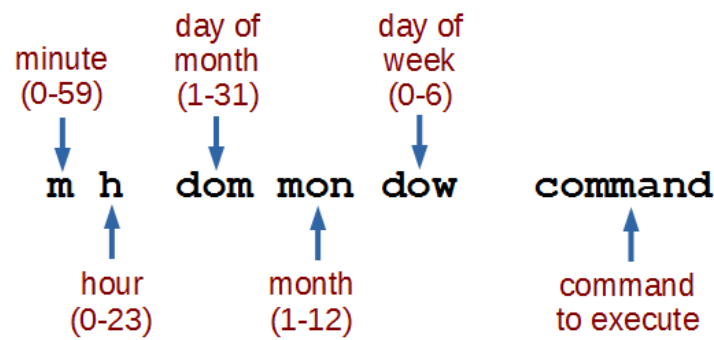
Once run it will open the crontab in the default [editor](#) on your system (most likely `'vi'`, `'vim'` or `'nano'`). The file will look as follows;

```
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
```

As stated earlier, the default file obviously includes some explanation of how to format an entry in the crontab. In our case we wish to add in an entry that tells the script to start at 6 hours and 0 minutes each day. The crontab accepts six pieces of information that will allow that action to be performed. each of those pieces is separated by a space.

1. A number (or range of numbers), m, that represents the minute of the hour (valid values 0-59);
2. A number (or range of numbers), h, that represents the hour of the day (valid values 0-23);
3. A number (or range of numbers), dom, that represents the day of the month (valid values 0-31);
4. A number (or list, or range), or name (or list of names), mon, that represents the month of the year (valid values 1-12 or Jan-Dec);
5. A number (or list, or range), or name (or list of names), dow, that represents the day of the week (valid values 0-6 or Sun-Sat); and
6. command, which is the command to be run, exactly as it would appear on the command line.

The layout is therefore as follows;



Linux permissions as rwx

Assuming that we want to run a Python script called 'm\_temp.py' which was in the 'pi' home directory the line that we would want to add would be as follows;

```
0 6 * * * /usr/bin/python /home/pi/m_temp.py
```

So at minute 0, hour 6, every day of the month (where the asterisk denotes 'everything'), every month, every day of the week we run the command `/usr/bin/python /home/pi/m_temp.py` (which, if we were at the command line in the pi home directory we would run as `python m_temp.py`, but since we can't guarantee where we will be when running the script, we are supplying the full path to the python command and the `m_temp.py` script.

If we wanted to run the command twice a day (6am and 6pm (1800hrs)) we can supply a comma separated value in the hours (h) field as follows;

```
0 6,18 * * * /usr/bin/python /home/pi/m_temp.py
```

If we wanted to run the command at 6am but only on weekdays (Monday through Friday) we can supply a range in the `dow` field as follows (remembering that 0 = Sunday);

```
0 6 * * 1-5 /usr/bin/python /home/pi/m_temp.py
```

If we want to run the same command every 2 hours we can use the `*/2` notation, so that our line in the crontab would look like the following;

```
0 */2 * * * /usr/bin/python /home/pi/m_temp.py
```

It's important to note that we need to include the `0` at the start (instead of the `*`) so that it doesn't run every minute every 2 hours (every minute in other words)

## Test yourself

1. How could you set up a schedule job in crontab that ran every second?
2. Create a crontab line to run a command on the 20th of July every year at 2 minutes past midnight.

## kill

The `kill` command allows us to terminate a process by sending it a signal. It can send different signals depending on how difficult the process is to stop. It relies on knowing the **Process IDentification number (PID)** of the process. The `kill` command is an important administrative tool for the purposes of ensuring that a system is running correctly.

- `kill [options] PID(s)` : sends a signal to terminate a process or processes by PID

Assuming that we have already established that the PID that we wish to terminate is 798, all we need to do is execute the following command;

```
kill 798
```

There is no feedback at the command line to indicate success or failure, but this can be accomplished using a separate command such as `ps`.

## The 'kill' command

The stated aim of the `kill` command is right there in the name. The object is to **kill** a process to stop it running. That might be because it needs to be shut down in an orderly way as a natural function of whatever is being carried out by the computer, or it could be required to try to stop a program that has become unresponsive. In this respect it is very similar to the `killall` command which will kill a process based on the originating command that ran it.

One of the great misunderstandings with the `kill` and `killall` commands is that their role is not to simply pull the plug on a process. Their job is to send a signal to a process asking it to shut down. Sometimes a process is resistant to the instruction and a more serious signal needs to be sent. The `kill` command can handle that too.

One of the most important aspects of using the `kill` command is that we will need to determine the **Process IDentification number (PID)** of the process to be terminated before we enact the `kill` command. To do this a couple of good options are the `ps` and / or the `top` commands. Either of these can be used with their various options to determine the appropriate PID and armed with that we can look at using `kill`.

As mentioned, the function of the `kill` command is to send a signal to a process. The range of possible signals that can be sent can be found by executing the `kill` command with the `-l` option (just for clarity, that's a **lower case L**, not the number 1). We can therefore execute the command as follows;

```
kill -l
```

... which will result in the following list of possible signals;

```

1) SIGHUP      2) SIGINT      3) SIGQUIT      4) SIGILL      5) SIGTRAP
6) SIGABRT     7) SIGBUS      8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV    12) SIGUSR2     13) SIGPIPE     14) SIGALRM    15) SIGTERM
16) SIGSTKFLT  17) SIGCHLD     18) SIGCONT     19) SIGSTOP    20) SIGTSTP
21) SIGTTIN    22) SIGTTOU     23) SIGURG      24) SIGXCPU    25) SIGXFSZ
26) SIGVTALRM  27) SIGPROF     28) SIGWINCH    29) SIGIO      30) SIGPWR
31) SIGSYS     34) SIGRTMIN     35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4 39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+\
13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-\
12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5 60) SIGRTMAX-4  61) SIGRTMAX-3 62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX

```

By default (without specifying a signal) the `kill` command will use signal 15 ‘SIGTERM’. If a process resists this effort to shut it down it is common practice to resort to signal 9 which is ‘SIGKILL’. This would be accomplished as follows (assuming that we had already established that the PID that we wanted to terminate was 798);

```
kill -9 798
```



It is an unusual situation to find a process that cannot be terminated with a `kill -9` command. The only time this shouldn’t be effective is if the process is making a request to the core of the operating system (the kernel). Even then, when it has completed the request, it should die.

If we want to terminate more than one process we can simply separate them with a space as follows;

```
kill 798 1234 666 999
```

## Test yourself

1. Which signal should be used first when executing the `kill` command, -9 or -15?

## killall

The `killall` command allows us to terminate a process or processes by sending it a signal. It can send different signals depending on how difficult the process is to stop. It is extremely similar to the `kill` command in its function, but does not rely on knowing the Process IDentification number (PID) of the process. Instead it can use the originating processes command name and as a result, when executed it can terminate *all* the processes started with that command. The `killall` command is an important administrative tool for the purposes of ensuring that a system is running correctly.

- `killall [options] command name` : sends a signal to terminate a process or processes by name

Assuming that we have already have a process named `top` running, all we need to do is execute the following command;

```
killall top
```

There is no feedback at the command line to indicate success or failure, but this can be accomplished using a separate command such as `ps`.

## The 'killall' command

The stated aim of the `killall` command is in the name. The object is to **kill all** the processes with a specific name to stop them running. That might be because they need to be shut down in an orderly way as a natural function of whatever is being carried out by the computer, or it could be required to try to stop programs that have become unresponsive.

One of the great misunderstandings with the `kill` and `killall` commands is that their role is not to simply pull the plug on a process. Their job is to send a signal to a process or processes asking them to shut down. Sometimes a process is resistant to the instruction and a more serious signal needs to be sent. The `killall` command can handle that to.

It might seem obvious but one of the most important aspects of using the `killall` command is that we will need to determine the *name* of the process to be terminated before we enact the command. To do this a couple of good options are the `ps` and `/` or the `top` commands. Either of these can be used with their various options to determine the appropriate name and armed with that we can look at using `killall`.

As mentioned the function of the `killall` command is to send a signal to a process. The range of possible signals that can be sent can be found by executing the `killall` command with the `-l` option (just for clarity, that's a **lower case L**, not the number 1). We can therefore execute the command as follows;



```
killall -l
```

... which will result in the following list of possible signals;

```
HUP INT QUIT ILL TRAP ABRT IOT BUS FPE KILL USR1 SEGV USR2 PIPE ALRM TERM
STKFLT CHLD CONT STOP TSTP TTIN TTOU URG XCPU XFSZ VTALRM PROF WINCH IO PWR S\
YS
UNUSED
```

By default (without specifying a signal) the `kill` command will use signal 'SIGTERM'. We should note at this stage that this signal is not on the list of signals listed by issuing the `killall` command with the `-l` option. That's because the signals listed there do not have the 'SIG' letters in front of them. So if we wanted to send a 'SIGKILL' signal, we would have to execute the command with the `-s` option and the full name of the signal like so (assuming that we had already established that the process `top` was the one that we wanted to terminate);

```
killall -s SIGKILL top
```



It is an unusual situation to find processes that cannot be terminated with a `killall -s SIGKILL` command. The only time this shouldn't be effective is if the processes are making a request to the core of the operating system (the kernel). Even then, when they have completed the request, they should die.

If we want to terminate more than one named process we can simply separate them with a space as follows;

```
killall top nano
```

This should terminate the processes `top` and `nano`.

## Test yourself

1. What are the differences between the `kill` command and the `killall` command?

## ps

The `ps` command shows a snapshot of the running processes. Similar to the `top` command, the `ps` command will report processes with associated characteristics and there are a wide range of different ways that they can be presented. However, where `top` will continually update itself, `ps` provides a fixed report. This is another useful command for troubleshooting and understanding system operation.

- `ps [options]` : displays a snapshot of the current processes

To view a list of running processes we can execute the following command;

```
ps -aux | less
```

Its actually two commands where the output of `ps` is `piped` into the `less` command so that the resulting long list can be examined one page at a time. The first page of the output will look a little like this;

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.4	5376	3944	?	Ss	06:08	0:07	/sbin/init
root	2	0.0	0.0	0	0	?	S	06:08	0:00	[kthreadd]
root	3	0.0	0.0	0	0	?	S	06:08	0:01	[ksoftirqd/0]
root	5	0.0	0.0	0	0	?	S<	06:08	0:00	[kworker/0:0H]
root	6	0.0	0.0	0	0	?	S	06:08	0:03	[kworker/u8:0]
root	7	0.0	0.0	0	0	?	S	06:08	0:04	[rcu_preempt]
root	8	0.0	0.0	0	0	?	S	06:08	0:00	[rcu_sched]
root	9	0.0	0.0	0	0	?	S	06:08	0:00	[rcu_bh]
root	10	0.0	0.0	0	0	?	S	06:08	0:00	[migration/0]
root	11	0.0	0.0	0	0	?	S	06:08	0:00	[migration/1]
root	12	0.0	0.0	0	0	?	S	06:08	0:00	[ksoftirqd/1]
root	14	0.0	0.0	0	0	?	S<	06:08	0:00	[kworker/1:0H]
root	15	0.0	0.0	0	0	?	S	06:08	0:00	[migration/2]
root	16	0.0	0.0	0	0	?	S	06:08	0:00	[ksoftirqd/2]

The output shows a similar feel to the `top` command in the sense that we see a list of processes and some information about them, but the information presented here is slightly different. The options used with the command told it to list the processes of all users (`-a`), to provide some detail about the processes (`u`) and to include processes that have no controlling terminal (`x`). This is a very common command for administering a Linux system.

## The `ps` command

The `ps` command is intended to report process status of currently running processes.

If it is executed without options it returns only four pieces of information on at least the two process that will be running. The shell and the `ps` command itself.

```
ps
```

... produces ...

PID	TTY	TIME	CMD
788	pts/0	00:00:01	bash
1642	pts/0	00:00:00	ps

Here we can see the following;

- The **P**rocess **I**D of the process (788)
- The **T**erminal **T**Ype or name of the terminal that the user logged into (pts/0)
- The total CPU **T**IME the process has used since it started (00:00.01)
- The **C**o**M**man**D** that was used to start the process (bash)

Obviously this amount of information is slightly limited, which is why the `ps` command is normally used with a combination of options.

## Options

While there are several options available, and they are most often used in combination and [piped](#) to [less](#) or [grep](#) to control the returned information. The most commonly used options are;

- `-a` : list the processes of all users
- `-u` : display results in a user orientated format
- `-x` : include processes without controlling terminals
- `-e` : include every process that is currently running
- `-f` : do a full listing of details for processes

Revisiting the initial example that we showed earlier;

```
ps -aux | less
```

With the (abridged) output;

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.4	5376	3944	?	Ss	06:08	0:07	/sbin/init
root	2	0.0	0.0	0	0	?	S	06:08	0:00	[kthreadd]
root	3	0.0	0.0	0	0	?	S	06:08	0:01	[ksoftirqd/0]
root	5	0.0	0.0	0	0	?	S<	06:08	0:00	[kworker/0:0H]

When done at the terminal the `-a` option will ensure that all the processes are listed (although `less` will control their release). The `-x` option includes those processes that don't have controlling terminals and the `-u` option has presented the information in a more detailed format. The various columns in the more detailed format correspond to;

- **USER** : The user that the process is running under.
- **%CPU** : The percentage of time the process has used the CPU since the process started
- **%MEM** : The percentage of real memory used by this process.
- **VSZ** : The size (in kilobytes) of the process in virtual memory.
- **RSS** : The real-memory size of the process (in kilobytes).
- **STAT** : The status of the process.
- **START** : When the process was started.
- **TIME** : The time the process has been running for
- **COMMAND** : The command the initiated the process

The **STAT** column includes a set of codes to describe the status of the processes. These can be interpreted with the following characters;

- **D** uninterruptible sleep (usually IO)
- **R** running or runnable (on run queue)
- **S** interruptible sleep (waiting for an event to complete)
- **T** stopped, either by a job control signal or because it is being traced
- **W** paging (not valid since the 2.6.xx kernel)
- **X** dead (should never be seen)
- **Z** defunct ("zombie") process, terminated but not reaped by its parent
- **<** high-priority (not nice to other users)
- **N** low-priority (nice to other users)
- **L** has pages locked into memory (for real-time and custom IO)
- **s** is a session leader
- **l** is multi-threaded
- **+** is in the foreground process group

The other commonly used combination of options is as follows;

```
ps -ef | less
```

With the (slightly edited for clarity) output;

```
avahi    354      1  0 06:22 ?        00:00:00 avahi-daemon: running
ntp      627      1  0 06:22 ?        00:00:00 /usr/sbin/ntpd -p /var/run/ntpd
pi       646      1  0 06:22 ?        00:00:00 Xtightvnc :1 -desktop X -auth
pi       730      1  0 06:22 ?        00:00:00 /bin/sh /home/pi/.vnc/xstartup
pi       750      1  0 06:22 ?        00:00:00 autocutsel -fork
pi       753     730  0 06:22 ?        00:00:00 /bin/sh /etc/X11/Xsession
pi       773     753  0 06:22 ?        00:00:00 xmessage -center -file -
root     774     615  0 06:23 ?        00:00:00 sshd: pi [priv]
pi       784     774  0 06:23 ?        00:00:00 sshd: pi@pts/0
pi       787     784  0 06:23 pts/0    00:00:00 -bash
pi       980     787  0 06:56 pts/0    00:00:00 ps -ef
pi       981     787  0 06:56 pts/0    00:00:00 grep --color=auto pi
```

Here we are including all the running processes with the `-e` option and `-f` to output additional details. In addition to this we have [piped](#) the output to [grep](#) which only returns lines that include 'pi' in them (with the thought that we are looking for processes associated with the 'pi' user. There is a small fly in this ointment in the form that the top line that is normally included with the `-f` option forms the column headers for each column. So for the sake of explaining the columns, the following is the (abridged) output with the column headers included;

UID	PID	PPID	C	STIME	TTY	TIME	CMD
avahi	354	1	0	06:22	?	00:00:00	avahi-daemon: running
ntp	627	1	0	06:22	?	00:00:00	/usr/sbin/ntpd -p /var/run/ntpd
pi	646	1	0	06:22	?	00:00:00	Xtightvnc :1 -desktop X -auth
pi	730	1	0	06:22	?	00:00:00	/bin/sh /home/pi/.vnc/xstartup
pi	750	1	0	06:22	?	00:00:00	autocutsel -fork
pi	753	730	0	06:22	?	00:00:00	/bin/sh /etc/X11/Xsession
pi	773	753	0	06:22	?	00:00:00	xmessage -center -file -
root	774	615	0	06:23	?	00:00:00	sshd: pi [priv]

The the columns relate to;

- UID : user ID
- PID : process ID
- PPID : parent process ID.
- C : processor utilization. (the integer value of the percent usage over the lifetime of the process)
- STIME : the start time of the process
- TTY : controlling tty (terminal).
- TIME : cumulative CPU time, '[DD-]HH:MM:SS' format
- CMD : the command that has been run

**Test yourself**

1. Is it better to pipe the output of `ps` into `less` or `grep`?
2. Which of the options `-f` or `-u` shows memory usage?

## top

The top command is used to display a real-time summary of system information and processes running on the computer. top is interactive and can be controlled and configured while operating to help determine different aspects of the systems operation. This is an extremely useful command to keep track of changing demands on the system and the changes as processes carry out their tasks.

- top [options] : display system and process information in real time

For example, just executing top from the command line as follows;

```
top
```

... will produce an output that looks something like this;

```
top - 08:50:22 up 2:41, 1 user, load average: 0.00, 0.01, 0.05
Tasks: 86 total, 1 running, 85 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.1 us, 0.2 sy, 0.0 ni, 99.7 id, 0.0 wa, 0.0 hi, 0.1 si, 0.0 st
KiB Mem: 948108 total, 139600 used, 808508 free, 14908 buffers
KiB Swap: 102396 total, 0 used, 102396 free. 82084 cached Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1231	pi	20	0	5092	2480	2148	R	0.7	0.3	0:00.15	top
1026	root	20	0	0	0	0	S	0.3	0.0	0:02.66	kworker/0:0
1	root	20	0	5376	3944	2792	S	0.0	0.4	0:05.79	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd
3	root	20	0	0	0	0	S	0.0	0.0	0:00.18	ksoftirqd/0
5	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/0:+
6	root	20	0	0	0	0	S	0.0	0.0	0:00.77	kworker/u8+
7	root	20	0	0	0	0	S	0.0	0.0	0:00.32	rcu_preempt
8	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcu_sched
9	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcu_bh
10	root	rt	0	0	0	0	S	0.0	0.0	0:00.00	migration/0
11	root	rt	0	0	0	0	S	0.0	0.0	0:00.00	migration/1
12	root	20	0	0	0	0	S	0.0	0.0	0:00.01	ksoftirqd/1
14	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/1:+
15	root	rt	0	0	0	0	S	0.0	0.0	0:00.01	migration/2
16	root	20	0	0	0	0	S	0.0	0.0	0:00.00	ksoftirqd/2
18	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/2:+

The top of top line displays an output which tells us how long the system has been operating for how many users there are and what our load is. The second line shows summary of tasks or

processes. The third line gives an indication of the state of the CPUs. The fourth and fifth lines show an indication of memory usage similar to that reported with the `free` command. Under that is a table showing processes, their properties and states.

The information displayed will update at a regular interval (defaults can vary, but about every 3 seconds). The `top` command can be stopped by pressing `q` (for `quit`).

## The `top` command

The `top` command is one of the most frequently used commands for monitoring system performance primarily because of the wide range of parameters that can be seen and the options available for configuring the viewed information. It's named '`top`' since it is useful in identifying the `top` users (processes) of resources in a system.

We should start with a description of the various displayed parameters in the default view and then work through the interactive commands available.

### The 1st line

The first line from our example looks like this;

```
top - 08:50:22 up 2:41, 1 user, load average: 0.00, 0.01, 0.05
```

The information presented here (in order) is;

- The current time (08:50:22)
- The length of time the system has been 'up' (operating) (2 hours and 41 minutes)
- The number of users logged in (1)
- The average load on the system in the last minute (0.00), five minutes (0.01) and 15 minutes (0.05)

### The 2nd line

The second line from our example looks like this;

```
Tasks: 86 total, 1 running, 85 sleeping, 0 stopped, 0 zombie
```

The information presented here (in order) is;

- The total number of processes that have been started on the system (86)
- The number of running processes (1)
- The number of sleeping processes (85)
- The number of zombie processes (0)



A zombie process is a process that has completed execution but still has an entry in the process table: it is a process in the 'Terminated state'. Having a zombie process is not necessarily a bad thing. It usually appears because the parent process may still need to read the exit status of the zombied / defunct child process



## The 3rd line

The third line from our example looks like this;

```
%Cpu(s): 0.1 us, 0.2 sy, 0.0 ni, 99.7 id, 0.0 wa, 0.0 hi, 0.1 si, 0.0 st
```

The information presented here (in order) is;

- The percentage of CPU allocated to **user** processes (0.1)
- The percentage of CPU allocated to **system** processes (0.2)
- The percentage of CPU processes with a priority upgrade (**ni**ce) (0.0)
- The percentage of CPU not used (**id**le) (99.7)
- The percentage of CPU processes **w**aiting for I/O operations (0.0)
- The percentage of CPU serving **h**ardware **i**nterrupts (0.0)
- The percentage of CPU serving **s**oftware interrupts (0.1)
- The percentage of CPU **s**tolen by the hypervisor for other tasks (0.0)

## The 4th and 5th lines

The fourth and fifth lines from our example looks like this;

```
KiB Mem:    948108 total,    139600 used,    808508 free,    14908 buffers
KiB Swap:   102396 total,      0 used,    102396 free.    82084 cached Mem
```

The fourth row shows how the physical memory (RAM) is being utilised and the fifth row shows the same type of utilisation for the swap memory.

### Mem

The Mem line is referring to the physical memory available on the system. Specifically this is memory used when the computer is running applications / operating system and generally processing data. It is the allocation of **R**andom **A**ccess **M**emory (RAM) as opposed to memory in the form of hard drives or similar that will persistently store information.

### Swap

Swap refers to memory that has been allocated on a hard drive or similar storage to help the system cope in situations where it has run out of RAM. Using swap space can be an indication that a system could require more resources, although sometimes this need can be extremely infrequent and the use could be thought of as an acceptable trade off to adding more RAM that might not get used very often.



Be aware, that when swap memory is used, the amount shown *as* used will increase, but even when it is finished, the 'used' amount will still show that higher value (since the space that the memory is referring to has technically been used). This can be confusing and could possibly lead us to think that there is a constant usage of swap memory when in fact it was momentary.

## Buffers / Cache

A buffer, also called buffer memory, is usually defined as a portion of memory that is set aside as a temporary holding place for data that is being sent to or received from an external device, such as a HDD, keyboard, printer or network. Cache is a memory location to store frequently used data for faster access. Cache data can be used multiple times where as buffer memory is used only once. And both are temporary stores for your data processing.

The fourth and fifth rows show their respective properties of;

- Total memory available
- Amount of memory used
- Amount of memory free
- Amount of memory used as a memory buffer for temporary holding (14908)
- Amount of memory used as a memory cache for frequently used data (82084)

## The process table

The process table from our example looks a little like the following (slightly truncated);

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1231	pi	20	0	5092	2480	2148	R	0.7	0.3	0:00.15	top
1026	root	20	0	0	0	0	S	0.3	0.0	0:02.66	kworker/0:0
1	root	20	0	5376	3944	2792	S	0.0	0.4	0:05.79	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd

This table shows the following properties of processes;

- The **Process ID** of the process (1231)
- The **USER** that is the owner of the process (pi)
- The **P**riority of the process (20)
- The **NI**ce value of the process (0)
- The **VIR**tual memory used by the process (5092)
- The physical (or **RES**ident) memory used from the process (2480)
- The **SHa**Red memory that the process is using (2148)
- The **S**tatus of the process (S=sleeping R=running Z=zombie) (R)
- The percentage of **CPU** used by the process (0.7)
- The percentage of Random Access **MEM**ory used by the process (0.3)
- The total CPU **TIME** the process has used since it started (0:00.15)
- The **COMMAND** that was used to start the process (top)

## Interactive commands

Once running, aside from 'q' to quit, the most useful command is 'h' to display the **h**elp screen. This will then display the available commands for top;

Help for Interactive Commands - procs-ng version 3.3.9

Window 1:Def: Cumulative mode Off. System: Delay 3.0 secs; Secure mode Off.

Z,B,E,e Global: 'Z' colors; 'B' bold; 'E'/'e' summary/task memory scale  
 l,t,m Toggle Summary: 'l' load avg; 't' task/cpu stats; 'm' memory info  
 0,1,2,3,I Toggle: '0' zeros; '1/2/3' cpus or numa node views; 'I' Irix mode  
 f,F,X Fields: 'f'/'F' add/remove/order/sort; 'X' increase fixed-width

L,&,<,> . Locate: 'L'/'&' find/again; Move sort column: '<'/'>' left/right  
 R,H,V,J . Toggle: 'R' Sort; 'H' Threads; 'V' Forest view; 'J' Num justify  
 c,i,S,j . Toggle: 'c' Cmd name/line; 'i' Idle; 'S' Time; 'j' Str justify  
 x,y . Toggle highlights: 'x' sort field; 'y' running tasks  
 z,b . Toggle: 'z' color/mono; 'b' bold/reverse (only if 'x' or 'y')  
 u,U,o,O . Filter by: 'u'/'U' effective/any user; 'o'/'O' other criteria  
 n,#,^O . Set: 'n'/'#' max tasks displayed; Show: Ctrl+'O' other filter(s)  
 C,... . Toggle scroll coordinates msg for: up,down,left,right,home,end

k,r Manipulate tasks: 'k' kill; 'r' renice

d or s Set update interval

W,Y Write configuration file 'W'; Inspect other output 'Y'

q Quit

( commands shown with '.' require a visible task display window )

Press 'h' or '?' for help with Windows,

Type 'q' or <Esc> to continue

Are there a lot of options there? Yes there are. We're not going to work through them since I think that there might be another book in describing them all. However, it may be worth mentioning one that isn't there. 'A' will toggle another screen that displays even more information! Yes. top is the gift that just keeps on giving. This will display something that looks a bit like this;

2:Job - 17:26:58 up 11:18, 1 user, load average: 0.00, 0.01, 0.05

Tasks: 88 total, 1 running, 87 sleeping, 0 stopped, 0 zombie

%Cpu(s): 0.4 us, 0.6 sy, 0.0 ni, 99.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st

KiB Mem: 948108 total, 140676 used, 807432 free, 15740 buffers

KiB Swap: 102396 total, 0 used, 102396 free. 82108 cached Mem

1	PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
	1	root	20	0	5376	3944	2792	S	0.0	0.4	0:06.71	systemd
	2	root	20	0	0	0	0	S	0.0	0.0	0:00.01	kthreadd
	5	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/0+

2	PID	PPID	TIME+	%CPU	%MEM	PR	NI	S	VIRT	SWAP	RES	UID	COMMAND
	1530	2	0:00.00	0.0	0.0	20	0	S	0	0	0	0	kworker+
	1529	2	0:00.00	0.0	0.0	20	0	S	0	0	0	0	kworker+
	1528	2	0:00.00	0.0	0.0	20	0	S	0	0	0	0	kworker+

3	PID	%MEM	VIRT	SWAP	RES	CODE	DATA	SHR	nMaj	nDRT	S	PR	NI
	646	0.7	8772	0	6916	1212	4960	2428	10	0	S	20	0

```

775  0.5  11072      0  4804    724    640   4180     0    0 S   20    0
614  0.5    7808      0  4288    724    592   3832    13    0 S   20    0
774  0.4   6888      0  4136     12    916   3272     1    0 S   20    0
4  PID  PPID   UID USER      RUSER   TTY      TIME+  %CPU %MEM S COMMAND
    1     0     0 root      root    ?        0:06.71  0.0  0.4 S systemd
    2     0     0 root      root    ?        0:00.01  0.0  0.0 S kthreadd
    3     2     0 root      root    ?        0:01.09  0.8  0.0 S ksoftirqd+
    5     2     0 root      root    ?        0:00.00  0.0  0.0 S kworker/0+
```

Yes... There's even more information and parameters to explain. But not right now. In the mean time, bask in the wonder that is the initial top screen and look out for the next book 'Just Enough top'.

## Test yourself

1. What is the process table in top sorted on?

# Network

## curl

The `curl` command is designed to send or receive files to or from a remote server. It works with a range of protocols and runs non-interactively. It is similar in function to [wget](#) but it supports a wider range of protocols and has a greater range of options for usage. However, it does not support recursive downloads.

- `curl [options] [URL]` : download or upload files to remote servers.

Using an example of downloading a file from the web, it is only necessary to use the `-O` option and provide the URL of the file that is required and the download will begin;

```
curl -O https://d3js.org/preview.png
```

The program will then connect to the remote server and start downloading the file `preview.png`;

% Total	% Received	% Xferd	Average Speed	Time	Time	Time	Current
			Dload Upload	Total	Spent	Left	Speed
100	288k	100	288k	0	0	148k	0
				0:00:01	0:00:01	--:--:--	148k

There is an updating progress display while the download continues and then the program finishes and we are left with an indication of the statistics involved in the downloading.

The file is downloaded into the current working directory.

## The `curl` command

`curl` is a utility that allows us to download and upload files over a network. While it is similar in function to [wget](#), there are enough differences to allow both to co-exist. For 95% of the downloading tasks accomplished by an ordinary user, either option will suffice. The name originated from the phrase “See URL” where the phonetic “See” was replaced with the character ‘c’. The reason for the name is inherent in the program’s use. Without any options, `curl` will echo the contents of a web page to the console. It only starts being able to write files to storage with the use of options.

`curl` is renowned for having an extremely wide range of options and flexibility for use. These are not all going to be of use to someone who just wants to move a file, but when we’re faced with a more complex download / upload task, `curl` is a viable option.

The range of supported protocols is wide and includes DICT, FILE, FTP, FTPS, GOPHER, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMTP, SMTPS, TELNET and TFTP. The different protocols used are specified by the URL that we pass as part

of the command. It uses these protocols in the sense that if you could paste a URL in a browser and have it subsequently download a file, the same file could be downloaded from the command line using `curl`. `curl` is not the only file downloading utility that is commonly used in Linux. `wget` is also widely used for similar functions. However both programs have different strengths and in the case of `curl` that strength is in support of a wider range of protocols and options / flexibility. There are other differences as well, but these would be the major ones.

There is a large range of options that can be used to ensure that downloads are configured correctly. We will examine a few of the more basic examples below.

- `--limit-rate` : **limit** the download speed / download **rate**.
- `-o` : download and store with a different file name
- `-u` : specify username and password authentication

### Rate limit the bandwidth

There will be times when we will be somewhere that the bandwidth is limited or we want to prioritise the bandwidth in some way. We can restrict the download speed with the option `--limit-rate` as follows;

```
curl -O --limit-rate 5k https://d3js.org/preview.png
```

Here we've limited the download speed to 5 kilobytes per second. The amount may be expressed in bytes, kilobytes with the `k` suffix, or megabytes with the `m` suffix.

% Total	% Received	% Xferd	Average Speed	Time	Time	Time	Current
			Dload Upload	Total	Spent	Left	Speed
100	288k	100	288k	0	0	5142	0
0:00:57	0:00:57	--:--:--	5425				

The assigned rate is the *average* speed experienced during the transfer. `curl` might use greater speeds in bursts, but over time it approximates the given rate. In the output example above, the overall download speed was just above the desired speed at 5142bps.

### Rename the downloaded file

If we try to download a file with the same name into the working directory it will overwrite the incumbent file. However, we can give the file a different name when downloaded using the `-o` option (that's a lower-case 'o' by the way). For example to save the file with the name `newpreview.png` we would do the following;

```
curl -o newpreview.png https://d3js.org/preview.png
```

### Download multiple files

We can download multiple files by simply including them one after the other in the command as follows;

```
curl -O https://d3js.org/forkme.png -O https://d3js.org/preview.png
```

While that is good, it can start to get a little confusing if a large number of URL's are included. The output provides a progress display and then a summary of the transfers as follows;

% Total	% Received	% Xferd	Average Speed	Time	Time	Time	Current
			Dload Upload	Total	Spent	Left	Speed
100	9267	100	9267	0	0	930	0
0:00:09	0:00:09	--:--:--	2103				
100	288k	100	288k	0	0	36503	0
0:00:08	0:00:08	--:--:--	47431				

### Download files that require a username and password

The examples shown thus far have been able to be downloaded without providing any form of authentication (no user / password). However this will be a requirement for some downloads. To include a username and password in the command we include the `-u` option. For example if we needed to use the username 'adam' and the password '1234' there are a couple of ways that we could proceed. We could form our command as follows;

```
curl -u adam:1234 ftp://website.org/file.zip
```

You may be thinking to yourself "Is this secure?". To which the answer should probably be "No". It is one step above anonymous access, but not a lot more. This is not a method by which things that should remain private should be secured, but it does provide a method of restricting anonymous access.

Alternatively (so long as it isn't being run as part of a script) we can simply include the username as follows and `curl` will prompt for a password;

```
curl -u adam ftp://website.org/file.zip
```

We still shouldn't pretend that this is entirely secure, since while we have avoided putting the password in the command history (i.e. we could find it by up-arrowing), there are still issues to be dealt with to numerous and in some cases to esoteric to mention. When in doubt, stay away from the Internet.

### Test yourself

1. Craft a `curl` command that downloads a file to a different name, limiting the download rate to 10 kilobytes per second.
2. Once question 1 above is carried out, where do we find the output of the downloads progress?

## host

The `host` command is used to determine the IP address of a computer from its host name or vice versa. The `host` command is simple to use and provides a useful tool to understanding details about a network.

- `host [options] server` : checks the IP address of a computer from its host-name or vice versa.

The simplest example employs just the command and a host-name similar to the following;

```
host nasa.org
```

This will produce the server(s) whose IP addresses resolve from the given host-name;

```
nasa.org has address 173.244.177.114
```

## The `host` command

The `host` command was named because it is effectively searching for information about a '**host**' where the term is referring to a networked server that is connected and referred to by an IP address and a domain name.

We could ask the question why we need to be able to carry out this function at all, but the simple answer to this is that the Internet is a complicated place and networks are complicated things. To be able to go places on the internet we rely on a Domain Name System (DNS) which can ensure that when we refer to a server on the internet by name it can find the correct IP address that the data needs to go to. So when we find that we're having problems resolving the correct address for a server, we need to have a mechanism by which we can test the DNS settings to ensure that everything is set as it should be.

## Options

There is a wide range of options that can be easily displayed by running the `host` command without any options or host-names;

- `-a` is equivalent to `-v -t ANY`
- `-c` specifies query class for non-IN data
- `-C` compares SOA records on authoritative nameservers
- `-d` is equivalent to `-v`
- `-l` lists all hosts in a domain, using AXFR
- `-i` IP6.INT reverse lookups
- `-N` changes the number of dots allowed before root lookup is done



- -r disables recursive processing
- -R specifies number of retries for UDP packets
- -s a SERVFAIL response should stop query
- -t specifies the query type
- -T enables TCP/IP mode
- -v enables verbose output
- -w specifies to wait forever for a reply
- -W specifies how long to wait for a reply
- -4 use IPv4 query transport only
- -6 use IPv6 query transport only
- -m set memory debugging flag (trace|record|usage)

This is a great list and for administrators and network engineers they provide a wealth of testing functionality. For mere mortals, I would suggest that the requirement is to have an understanding of the command without using options and perhaps the -t option.

The -t option allows us to be selective as to the type of DNS records we get reported. There are several different resource types that can be specified;

- a IPv4 IP address
- aaa IPv6 IP address
- cname **canonical name** record (Alias)
- mx **email exchange** server host names
- ns **name** (DNS) server names
- ptr **pointer** to a canonical name
- soa Authoritative information about a DNS zone
- txt **text** record

For example, to show only the email servers for google.com we could execute the following;

```
host -t mx google.com
```

This will report the servers responsible for managing email for google.com;

```
google.com mail is handled by 40 alt3.aspmx.1.google.com.  
google.com mail is handled by 20 alt1.aspmx.1.google.com.  
google.com mail is handled by 10 aspmx.1.google.com.  
google.com mail is handled by 50 alt4.aspmx.1.google.com.  
google.com mail is handled by 30 alt2.aspmx.1.google.com.
```

**Test yourself**

1. What would happen if you entered the URL `www.google.com` into a browser if there wasn't a DNS service available?
2. Is executing the command `host nasa.org` or `host 173.244.177.114` referred to as a 'reverse' lookup?

## ifconfig

The `ifconfig` command can be used to view the configuration of, or to configure a network interface. Networking is a fundamental function of modern computers. `ifconfig` allows us to configure the network interfaces to allow that connection.

- `ifconfig [arguments] [interface]`

or

- `ifconfig [arguments] interface [options]`

Used with no 'interface' declared `ifconfig` will display information about all the operational network interfaces. For example running;

```
ifconfig
```

... produces something similar to the following on a simple Raspberry Pi.

```
eth0      Link encap:Ethernet  HWaddr 76:12:45:56:47:53
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

wlan0     Link encap:Ethernet  HWaddr 09:87:65:54:43:32
          inet addr:10.1.1.8  Bcast:10.1.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:3978 errors:0 dropped:898 overruns:0 frame:0
          TX packets:347 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:859773 (839.6 KiB)  TX bytes:39625 (38.6 KiB)
```

The output above is broken into three sections; `eth0`, `lo` and `wlan0`.

- `eth0` is the first Ethernet interface and in our case represents the RJ45 network port on the Raspberry Pi (in this specific case on a B+ model). If we had more than one Ethernet interface, they would be named `eth1`, `eth2`, etc.
- `lo` is the loopback interface. This is a special network interface that the system uses to communicate with itself. You can notice that it has the IP address 127.0.0.1 assigned to it. This is described as designating the 'localhost'.
- `wlan0` is the name of the first wireless network interface on the computer. This reflects a wireless interface (if installed). Any additional wireless interfaces would be named `wlan1`, `wlan2`, etc.

## The `ifconfig` command

The `ifconfig` command is used to read and manage a servers network interface configuration (hence `ifconfig` = interface **configuration**).

We can use the `ifconfig` command to display the current network configuration information, set up an ip address, netmask or broadcast address on an network interface, create an alias for network interface, set up hardware addresses and enable or disable network interfaces.



`ifconfig` has been 'deprecated' in some Linux distributions, which means that the software has been superseded and where practical an alternative used. Although deprecated, the command is still available, although its use may raise warning messages recommending alternative practices, and deprecation may indicate that the feature will be removed in the future. Features are deprecated rather than immediately removed in order to provide backward compatibility. In the case of `ifconfig` the alternative (which we will discuss in a separate section) is [ip](#).

To view the details of a specific interface we can specify that interface as an argument;

```
ifconfig eth0
```

Which will produce something similar to the following;

```
eth0      Link encap:Ethernet  HWaddr b8:27:eb:2c:bc:62
          inet addr:10.1.1.8  Bcast:10.1.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:119833 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8279 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:8895891 (8.4 MiB)  TX bytes:879127 (858.5 KiB)
```

The configuration details being displayed above can be interpreted as follows;

- `Link encap:Ethernet` - This tells us that the interface is an Ethernet related device.

- `HWaddr b8:27:eb:2c:bc:62` - This is the hardware address or Media Access Control (MAC) address which is unique to each Ethernet card. Kind of like a serial number.
- `inet addr:10.1.1.8` - indicates the interfaces IP address.
- `Bcast:10.1.1.255` - denotes the interfaces broadcast address
- `Mask:255.255.255.0` - is the network mask for that interface.
- `UP` - Indicates that the kernel modules for the Ethernet interface have been loaded.
- `BROADCAST` - Tells us that the Ethernet device supports broadcasting (used to obtain IP address via DHCP).
- `RUNNING` - Lets us know that the interface is ready to accept data.
- `MULTICAST` - Indicates that the Ethernet interface supports multicasting.
- `MTU:1500` - Short for **Maximum Transmission Unit** is the size of each packet received by the Ethernet card.
- `Metric:1` - The value for the Metric of an interface decides the priority of the device (to designate which of more than one devices should be used for routing packets).
- `RX packets:119833 errors:0 dropped:0 overruns:0 frame:0` and `TX packets:8279 errors:0 dropped:0 overruns:0 carrier:0` - Show the total number of packets received and transmitted with their respective errors, number of dropped packets and overruns respectively.
- `collisions:0` - Shows the number of packets which are colliding while traversing the network.
- `txqueuelen:1000` - Tells us the length of the transmit queue of the device.
- `RX bytes:8895891 (8.4 MiB)` and `TX bytes:879127 (858.5 KiB)` - Indicates the total amount of data that has passed through the Ethernet interface in transmit and receive.

## Options

The main option that would be used with `ifconfig` is `-a` which will display all of the interfaces on the system available (ones that are 'up' (active) and 'down' (shut down)). The default use of the `ifconfig` command without any arguments or options will display only the active interfaces.

```
ifconfig -a
```

## Arguments

We can disable an interface (turn it down) by specifying the interface name and using the suffix 'down' as follows;

```
ifconfig eth0 down
```

Or we can make it active (bring it up) by specifying the interface name and using the suffix 'up' as follows;

```
ifconfig eth0 up
```

To assign a IP address to a specific interface we can specify the interface name and use the IP address as the suffix;

```
ifconfig eth0 10.1.1.8
```

To add a netmask to a a specific interface we can specify the interface name and use the netmask argument followed by the netmask value;

```
ifconfig eth0 netmask 255.255.255.0
```

To assign an IP address and a netmask at the same time we can combine the arguments into the same command;

```
ifconfig eth0 10.1.1.8 netmask 255.255.255.0
```

## Test yourself

1. List all the network interfaces on your server.
2. Why might it be a bad idea to turn down a network interface while working on a server remotely?
3. Display the information about a specific interface, turn it down, display the information about it again then turn it up. What differences do you see?

## ip

The `ip` command is a relatively recent network configuration command that is stepping in to replace the `ifconfig` command that has been deprecated (but is still widely supported). Like `ifconfig` it can be used to view the configuration of, or to configure a network interface, but it has a significant range of additional functions which can be enacted using its flexible command architecture. Networking is a fundamental function of modern computers. `ip` allows us to configure network interfaces to allow that connection.

- `ip [options] object sub-command [parameters]` : network configuration tool

To display the information about all the operational network interfaces on the current system we can execute the command as follows;

```
ip address show
```

... produces something similar to the following on a simple Raspberry Pi.

```
1: lo: <LOOPBACK,UP,LOWER_UP>
    mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP>
    mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether b8:45:e6:8c:9c:61 brd ff:ff:ff:ff:ff:ff
    inet 10.1.1.33/24 brd 10.1.1.255 scope global eth0
        valid_lft forever preferred_lft forever
    inet6 fa80::1627:9266:d35e:705f/64 scope link
        valid_lft forever preferred_lft forever
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP>
    mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 83:2f:62:e4:d4:64 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.199/24 brd 192.168.1.255 scope global wlan0
        valid_lft forever preferred_lft forever
    inet6 fe80::44cc:e626:be2b:62a4/64 scope link
        valid_lft forever preferred_lft forever
```

The output above is broken into three sections; `lo`, `eth0` and `wlan0`.

- `lo` is the loopback interface. This is a special network interface that the system uses to communicate with itself. We can see that it has the IP address 127.0.0.1 assigned to it. This is described as designating the 'localhost'.

- `eth0` is the first Ethernet interface and in our case represents the RJ45 network port on the Raspberry Pi (in this specific case on a B+ model). If we had more than one Ethernet interface, they would be named `eth1`, `eth2`, etc.
- `wlan0` is the name of the first wireless network interface on the computer. This reflects a wireless USB adapter (if installed). Any additional wireless interfaces would be named `wlan1`, `wlan2`, etc.

There is a considerable amount of information packed into the response to the command. We won't explain all of it, but parts that could be considered important from a basic stand point could be;

- `UP` - Indicates that the kernel modules for the Ethernet interface have been loaded.
- `LOWER_UP` tells us the state of the Ethernet link. In this context it's an indication that it's connected (the cable is plugged in or the wireless system correctly set up) and it can see a device on the other end of the connection.
- `BROADCAST` - Tells us that the Ethernet device supports broadcasting (used to obtain IP address via DHCP).
- `MULTICAST` - Indicates that the Ethernet interface supports multicasting.
- `mtu 1500` - Short for **m**aximum **t**ransmission **u**nit is the maximum size of packets used by the Ethernet card.
- The `link/ether` is the hardware address or **M**edia **A**ccess **C**ontrol (MAC) address which is unique to each Ethernet card. Kind of like a serial number. In the case of `eth0` it is reported as `b8:45:e6:8c:9c:61`.
- The `inet` section shows us the IP address of our interface combined with the netmask in what's known as CIDR notation. In the case of `eth0` with `10.1.1.33/24` this tells us that it's the address `10.1.1.33` with a netmask `255.255.255.0`.
- on the same line as the `inet` section, `brd` denotes the interfaces broadcast address as `10.1.1.255`



An alternative to specifying a netmask in the format of '255.255.255.0' is to use a system called Classless Inter-Domain Routing, or CIDR. The concept is that you can add a specification in the IP address itself that indicates the number of significant bits that make up the netmask.

For example, we could designate the IP address `10.1.1.33` as associated with the netmask `255.255.255.0` by using the CIDR notation of `10.1.1.33/24`. This means that the first 24 bits of the IP address given are considered significant for the network routing.

Using CIDR notation allows us to do some very clever things to organise our network, but at the same time it can have the effect of freaking people out by introducing a pretty complex topic when all they want to do is get their network going :-). So for the sake of this explanation we can assume that if we wanted to specify an IP address and a netmask, it could be accomplished by either specifying each separately (IP address = `10.1.1.33` and netmask = `255.255.255.0`) or in CIDR format (`10.1.1.33/24`)

It should be noted that the `ip` command has a range of abbreviation options for its objects, commands and parameters. This means that the example above could just as easily be carried out with;



```
ip addr show
```

... or even more aggressively ...

```
ip a s
```

## The `ip` command

The `ifconfig` command has been deprecated and its function has been replaced by several different commands. One of those commands is `ip`. It has been named as a result of its association with network routing and the subsequent association with the **internet protocol** which is the principal communications protocol in the Internet protocol suite for relaying datagrams across network boundaries.

The `ip` command is designed to be flexible enough to perform a wide range of network tasks. It gains this flexibility through the design of the syntax that it employs where it can combine an *object* a *sub-command* and *parameters*.

For example, objects that it can apply itself to include;

- `address` : a protocol (IP or IPv6) address on a device.
- `link` : a network device.
- `maddress` : a multicast address.
- `mrout` : a multicast routing cache entry.
- `route` : a routing table entry.
- `rule` : a rule in routing policy database.
- `tunnel` : a tunnel over IP.

Sub-commands that be requested include (their actions are relatively self-explanatory);

- `add`
- `change`
- `delete`
- `flush`
- `get`
- `show`
- `save`
- `set`
- `update`

And the parameters that can be configured include;

- `dev interface_name` : a network interface (the `dev` keyword is often optional)
- `up` : to turn something on
- `down` : to turn something off
- `primary` : to set a priority to number 1
- `secondary` : to set a priority to number 2
- `type` : a type of route
- `via` : a gateway to route to
- `default` : a default route

`ip` is therefore able to use this combination of objects, sub-commands and parameters to form a flexible ‘language’ of sorts from their combination.

## Examples of use

The following commands are a simple snapshot of the possible combinations, but they serve to reinforce the possibilities and provide a reference for some common use.



Some of the following examples will require administrator privileges and you may need to prefix them with `sudo` to execute them successfully.

To show a specific network interface (`wlan0`);

```
ip address show wlan0
```

This will show the details for the `wlan0` interface on the screen;

```
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP>  
    mtu 1500 qdisc mq state UP group default qlen 1000  
    link/ether 80:1f:02:f4:54:68 brd ff:ff:ff:ff:ff:ff  
    inet 192.168.1.199/24 brd 192.168.1.255 scope global wlan0  
        valid_lft forever preferred_lft forever  
    inet6 fe80::257f:586d:65c3:b83b/64 scope link  
        valid_lft forever preferred_lft forever
```

To disable the interface `wlan0` (to turn it down) and then to show the details for it;

```
ip link set wlan0 down  
ip address show wlan0
```

```
3: wlan0: <BROADCAST,MULTICAST>  
    mtu 1500 qdisc mq state DOWN group default qlen 1000  
    link/ether 80:1f:02:f4:54:68 brd ff:ff:ff:ff:ff:ff
```

To enable the interface `wlan0` (to turn it up) and then to show the details for it;

```
ip link set wlan0 up  
ip address show wlan0
```

```
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP>  
    mtu 1500 qdisc mq state UP group default qlen 1000  
    link/ether 80:1f:02:f4:54:68 brd ff:ff:ff:ff:ff:ff  
    inet6 fe80::257f:586d:65c3:b83b/64 scope link tentative  
        valid_lft forever preferred_lft forever
```

To replace an IP address and a netmask in CIDR format we have to carry out two steps to achieve our goal. Firstly we have to add our new address which will add a new address to the interface as a secondary address and then we need to remove the original address. This might seem like one step too many, and the question has been asked “Why not use the replace or change sub-commands?”. They are valid questions and the answer that I believe is the most logical is that there could be more than one address for the interface already in which case, which one would get replaced / changed? Whatever the case to replace an address and netmask on an interface we must first add a new one (and here we also echo it to the screen to see the resulting secondary address);

```
ip address add 192.168.1.198/24 dev wlan0  
ip address show wlan0
```

```
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP>
    mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 80:1f:02:f4:54:68 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.199/24 brd 192.168.1.255 scope global wlan0
        valid_lft forever preferred_lft forever
    inet 192.168.1.198/24 scope global secondary wlan0
        valid_lft forever preferred_lft forever
    inet6 fe80::257f:586d:65c3:b83b/64 scope link
        valid_lft forever preferred_lft forever
```

... Then we remove the original and show the details for the interface;

```
ip address delete 192.168.1.199/24 dev wlan0
ip address show wlan0
```

```
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP>
    mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 80:1f:02:f4:54:68 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.198/24 scope global wlan0
        valid_lft forever preferred_lft forever
    inet6 fe80::257f:586d:65c3:b83b/64 scope link
        valid_lft forever preferred_lft forever
```

The `ip` command is a very powerful command with a wide range of options and a flexible syntax. There are elements in the command that are more squarely focussed at power users, but there are uses that a beginner will find essential.

## Test yourself

1. Show how the `replace` sub-command can be used to change the address of a network interface.

## netstat

The `netstat` command allows us to display network connection information including incoming and outgoing traffic and a range of network interface statistics. It is a commonly used tool for administering and troubleshooting network connection problems and is a great source of information on how our computer is connected to the outside world.

- `netstat [options]` : network connection monitoring tool

The key to using `netstat` is in the use of the options to control the output. There is a lot of information that can be displayed using the command and understanding the options is important to either filter down to the data we're interested in or to display the type of information we want. In the following example using the `-a` option we will show all the Internet connections and domain sockets that are associated with the network connections we have set up (this is a shortened version to save paper);

```
netstat -a
```

Active Internet connections (servers and established)

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
tcp	0	0	*:5901	*:*	LISTEN
tcp	0	0	*:ssh	*:*	LISTEN
tcp	0	64	10.1.1.33:ssh	10.1.1.225:52535	ESTABLISHED
tcp	0	0	10.1.1.33:ssh	10.1.1.225:53168	ESTABLISHED
tcp6	0	0	:::ssh	:::*	LISTEN
udp	0	0	*:56800	*:*	
udp	0	0	10.1.1.33:ntp	*:*	
udp	0	0	192.168.1.199:ntp	*:*	
udp	0	0	localhost:ntp	*:*	
udp	0	0	*:ntp	*:*	
udp6	0	0	:::mdns	:::*	

Active UNIX domain sockets (servers and established)

Proto	RefCnt	Flags	Type	State	I-Node	Path
unix	2	[ ]	DGRAM		6919	/var/run/wpa_supplicant/wlan0
unix	2	[ ]	DGRAM		4720	/run/systemd/notify
unix	2	[ ACC ]	STREAM	LISTENING	4722	/run/systemd/private
unix	2	[ ]	DGRAM		4738	/run/systemd/shutdown
unix	8	[ ]	DGRAM		4740	/run/systemd/journal/dev-log
unix	2	[ ACC ]	SEQPACKET	LISTENING	4744	/run/udev/control
unix	2	[ ACC ]	STREAM	LISTENING	4748	/run/systemd/journal/stdout
unix	5	[ ]	DGRAM		4750	/run/systemd/journal/socket
unix	2	[ ACC ]	STREAM	LISTENING	5775	/var/run/avahi-daemon/socket
unix	2	[ ]	DGRAM		1447	/run/systemd/journal/syslog

```

unix  2      [  ]          DGRAM                    5800    /var/run/thd.socket
unix  2      [ ACC ]     STREAM        LISTENING  7338    /var/run/dhcpd.sock
unix  2      [  ]          DGRAM                    7342
unix  3      [  ]          STREAM        CONNECTED  7995

```

This is obviously a lot of information (and I have reduced it by about half for reproduction here). The key is to know how to parse out the information we're looking for.

The output is divided into two parts. Firstly the Internet connections where the respective columns refer to;

- Proto : The protocol name
- Recv-Q : The count of bytes not copied by the user program connected to this socket
- Send-Q : The count of bytes not acknowledged by the remote host
- Local Address : Address and port number of the local end of the socket.
- Foreign Address : Address and port number of the remote end of the socket.
- State : The state of the socket. Normally this can be one of several values:
  - ESTABLISHED : The socket has an established connection.
  - SYN\_SENT : The socket is actively attempting to establish a connection.
  - SYN\_RECV : A connection request has been received from the network.
  - FIN\_WAIT1 : The socket is closed, and the connection is shutting down.
  - FIN\_WAIT2 : Connection is closed, and the socket is waiting for a shut-down from the remote end.
  - TIME\_WAIT : The socket is waiting after close to handle packets still in the network.
  - CLOSE : The socket is not being used.
  - CLOSE\_WAIT : The remote end has shut down, waiting for the socket to close.
  - LAST\_ACK : The remote end has shut down, and the socket is closed. Waiting for acknowledgement.
  - LISTEN : The socket is listening for incoming connections. Such sockets are not included in the output unless we specify the `-listening (-l)` or `-all (-a)` option.
  - CLOSING : Both sockets are shut down but we still don't have all our data sent.
  - UNKNOWN : The state of the socket is unknown.

Secondly the active domain sockets where the columns refer to;

- Proto : The protocol (usually unix) used by the socket.
- RefCnt : The reference count (i.e. attached processes via this socket)
- Flags : The flag normally displayed is ACC (rarely also W for wait data or N for no space). ACC is used on unconnected sockets if their corresponding processes are waiting for a connect request.
- Type : There are several types of socket access including;
  - DGRAM : The socket is used in Datagram (connectionless) mode.
  - STREAM : This is a stream (connection) socket.
  - RAW : The socket is used as a raw socket.
  - RDM : This one serves reliably-delivered messages.

- SEQPACKET: This is a sequential packet socket.
  - PACKET : Raw interface access socket.
  - UNKNOWN : For unknown or future types
- State : This field will contain one of the following Keywords;
  - FREE : The socket is not allocated
  - LISTENING : The socket is listening for a connection request. (only included in the output if we specify the -l or -a option.
  - CONNECTING : The socket is about to establish a connection.
  - CONNECTED : The socket is connected.
  - DISCONNECTING : The socket is disconnecting.
  - (empty) : The socket is not connected to another one.
- I-Node : The unique [inode](#) number
- Path : This is the path name by which the corresponding processes attached to the socket.

As we have already observed, there is a lot of information here and the key is to understand the methods of using netstat to filter and discover that information. Examples of this follow.

## The netstat command

netstat is a command that can be used to monitor a range of information about the Linux networking subsystem. The type of information displayed is controlled by the first option, as follows:

- (none) : By default, netstat displays a list of open sockets.
- -a : Display the Internet connections and domain sockets associated with network connections.
- -i : Display a table of all network interfaces.
- -M : Display a list of Masqueraded connections.
- -s : Display summary statistics for each protocol.

When displaying the Internet connections and domain sockets with the -a option we can further modify the returned information by specifying the protocol with -t for TCP and -u for UDP connections. Therefore the following command with the options -at will show all the Internet connections and domain sockets using the TCP protocol as follows;

```
netstat -at
```

Active Internet connections (servers and established)

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
tcp	0	0	*:5901	*:*	LISTEN
tcp	0	0	*:x11-1	*:*	LISTEN
tcp	0	0	*:ssh	*:*	LISTEN
tcp	0	0	10.1.1.8:ssh	10.1.1.25:52535	ESTABLISHED
tcp	0	256	10.1.1.8:ssh	10.1.1.25:53168	ESTABLISHED
tcp6	0	0	:::ssh	:::*	LISTEN

In the same way we can display a summary of network statistics for just the UDP protocol with the following command;

```
netstat -su
```

Udp:

```
738 packets received
0 packets to unknown port received.
0 packet receive errors
617 packets sent
IgnoredMulti: 9610
```

UdpLite:

IpExt:

```
InMcastPkts: 159
OutMcastPkts: 22
InBcastPkts: 9610
InOctets: 3576306
OutOctets: 543235
InMcastOctets: 17183
OutMcastOctets: 5198
InBcastOctets: 1712994
InNoECTPkts: 17471
```

When we want to show a summary of the network interfaces we can use the -i option as follows;

```
netstat -i
```

... to produce an output as follows;



Iface	MTU	Met	RX-OK	RX-ERR	RX-DRP	RX-OVR	TX-OK	TX-ERR	TX-DRP	TX-OVR	Flg
eth0	1500	0	18296	0	0	0	3222	0	0	0	BMRU
lo	65536	0	0	0	0	0	0	0	0	0	LRU
wlan0	1500	0	6561	0	1	0	108	0	0	0	BMRU

But to get a extended report on the interfaces we can combine the `-i` option with `e` as follows;

```
netstat -ie
```

... to produce;

#### Kernel Interface table

```
eth0      Link encap:Ethernet  HWaddr bd:37:a1:8c:8c:22
          inet addr:10.1.1.33  Bcast:10.1.1.255  Mask:255.255.255.0
          inet6 addr: fe80::1627:9566:445e:907f/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:18412 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3296 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2265470 (2.1 MiB)  TX bytes:688861 (672.7 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

wlan0     Link encap:Ethernet  HWaddr 45:12:05:54:27:88
          inet addr:192.168.1.199  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::2c3c:7626:232b:67a8/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:6576 errors:0 dropped:1 overruns:0 frame:0
          TX packets:108 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2208189 (2.1 MiB)  TX bytes:16428 (16.0 KiB)
```

It is also worth mentioning that for a command like this with a relatively verbose output, [piping](#) the command to [grep](#) would be an efficient mechanism to reduce the output if required. For example to show the hardware addresses for the interfaces we could execute the following command;

```
netstat -ie | grep HWaddr
```

... which would produce;

```
eth0      Link encap:Ethernet  HWaddr bd:37:a1:8c:8c:22  
wlan0     Link encap:Ethernet  HWaddr 45:12:05:54:27:88
```

## Test yourself

1. What would be the options to use to display a summary of the statistics for the TCP protocol.

## ping

The `ping` command allows us to check the network connection between the local computer and a remote server. It does this by sending a request to the remote server to reply to a message (kind of like a read-request in email). This allows us to test network connectivity to the remote server and to see if the server is operating. The `ping` command is a simple and commonly used network troubleshooting tool.

- `ping [options] remote server` : checks the connection to a remote server.

To check the connection to the server at CNN for example we can simply execute the following command (assuming that we have a connection to the internet);

```
ping cnn.com
```

Which will return something like the following;

```
PING cnn.com (157.166.226.25) 56(84) bytes of data.  
64 bytes from www.cnn.com (157.166.226.25): icmp_seq=1 ttl=111 time=265 ms  
64 bytes from www.cnn.com (157.166.226.25): icmp_seq=2 ttl=111 time=257 ms  
64 bytes from www.cnn.com (157.166.226.25): icmp_seq=3 ttl=111 time=257 ms  
64 bytes from www.cnn.com (157.166.226.25): icmp_seq=4 ttl=111 time=258 ms  
64 bytes from www.cnn.com (157.166.226.25): icmp_seq=5 ttl=111 time=257 ms  
64 bytes from www.cnn.com (157.166.226.25): icmp_seq=6 ttl=111 time=257 ms  
^C  
--- cnn.com ping statistics ---  
14 packets transmitted, 13 received, 7% packet loss, time 13016ms  
rtt min/avg/max/mdev = 257.199/267.169/351.320/24.502 ms
```

The first thing to note is that by default the `ping` command will just keep running. When we want to stop it we need to press CTRL-c to get it to stop.

The information presented is extremely useful and tells us that `www.cnn.com`'s IP address is 157.166.226.25 and that the time taken for a ping send and return message took about 250 milliseconds.



As an aside, the ping command is one of those commands that has illustrated nicely for me the need to write a book that has some simpler language in it to explain things. For example, the following is the description of the ping command from the man pages;

“ping uses the ICMP protocol’s mandatory ECHO\_REQUEST datagram to elicit an ICMP ECHO\_RESPONSE from a host or gateway. ECHO\_REQUEST datagrams (‘pings’) have an IP and ICMP header, followed by a struct timeval and then an arbitrary number of ‘pad’ bytes used to fill out the packet.”

Don’t get me wrong. It’s a great description and I’m not suggesting that it be changed, but it’s not entirely friendly to a new user.

## The ping command

The ping command is a very simple network / connectivity checking tool that is one of the default ‘go-to’ commands for system administrators. You might be wondering about how the name has come about. It is reminiscent of the echo-location technique used by dolphins, whales and bats to send out a sound and to judge their surroundings by the returned echo. In the dramatised world of the submariner, a ping is the sound emitted by a submarine in the same way to judge the distance and direction to an object. It was illustrated to best effect in the book by Tom Clancy and the subsequent movie “[The Hunt for Red October](https://en.wikipedia.org/wiki/The_Hunt_for_Red_October)<sup>21</sup>” where the submarine commander makes the request for “One Ping Only”.



One Ping Only

It works by sending message called an ‘Echo Request’ to a specific network location (which we specify as part of the command). When (or if) the server receives the request it sends an ‘Echo Reply’ to the originator that includes the exact payload received in the request. The command will continue to send and (hopefully) receive these echoes until the command completes its requisite number of attempts or the command is stopped by the user (with a CTRL-c). Once complete, the command summarises the effort.

---

<sup>21</sup>[https://en.wikipedia.org/wiki/The\\_Hunt\\_for\\_Red\\_October](https://en.wikipedia.org/wiki/The_Hunt_for_Red_October)

From the example used above we can see the output as follows;

```
PING cnn.com (157.166.226.25) 56(84) bytes of data.
64 bytes from www.cnn.com (157.166.226.25): icmp_seq=1 ttl=111 time=265 ms
64 bytes from www.cnn.com (157.166.226.25): icmp_seq=2 ttl=111 time=257 ms
64 bytes from www.cnn.com (157.166.226.25): icmp_seq=3 ttl=111 time=257 ms
64 bytes from www.cnn.com (157.166.226.25): icmp_seq=4 ttl=111 time=258 ms
64 bytes from www.cnn.com (157.166.226.25): icmp_seq=5 ttl=111 time=257 ms
64 bytes from www.cnn.com (157.166.226.25): icmp_seq=6 ttl=111 time=257 ms
^C
--- cnn.com ping statistics ---
14 packets transmitted, 13 received, 7% packet loss, time 13016ms
rtt min/avg/max/mdev = 257.199/267.169/351.320/24.502 ms
```

We can see from the returned pings that the IP address of the server that is designated as 'www.cnn.com' is '157.166.226.25' (The resolution of the IP address would be made possible by DNS, but using a straight IP address is perfectly fine). The `icmp_seq=` column tells us the sequence of the returned replies and `ttl` indicates how many IP routers the packet can go through before being thrown away. The time provides the measured return trip of the request and reply.

The summary at completion tells us how many packets were sent and how many received back. This forms a percentage of lost packets which is established over the specified time. The final line provides a minimum, average maximum and standard deviation from the mean.

## Options

There are a few different options for use, but the more useful are as follows;

- `-c` only ping the connection a certain number (count) of times
- `-i` change the time interval between pings

It's really useful to have ping running continuously so that we can make changes to networking while watching the results, but it's also useful to run the command for a limited amount of time. This is where the `-c` option comes in. This will simply restrict the number of pings that are sent out and will then cease and summarise the effort. This can be used as follows;

```
ping -c 4 cnn.com
```

Which will return something like the following;

```
PING cnn.com (157.166.226.25) 56(84) bytes of data.  
64 bytes from www.cnn.com (157.166.226.25): icmp_seq=1 ttl=111 time=266 ms  
64 bytes from www.cnn.com (157.166.226.25): icmp_seq=2 ttl=111 time=263 ms  
64 bytes from www.cnn.com (157.166.226.25): icmp_seq=3 ttl=111 time=288 ms  
64 bytes from www.cnn.com (157.166.226.25): icmp_seq=4 ttl=111 time=280 ms  
  
--- cnn.com ping statistics ---  
4 packets transmitted, 4 received, 0% packet loss, time 3003ms  
rtt min/avg/max/mdev = 263.283/274.586/288.637/10.386 ms
```

Sometimes it can be convenient to set our own time interval between pings. This can be accomplished with the `-i` option which will let us vary the repeat time. The default is 1 second, however the value cannot be set below 0.2 seconds without doing so as the superuser. Interestingly there is an option to flood the network with pings (flood mode) to test the network infrastructure. However, this would be something typically left to research *carefully* when you really need it.

## Test yourself

1. How does the ping command to a server name know how to return an IP address?
2. What does 'ttl' stand for?

## scp

The `scp` command is used to securely copy files using a [ssh](#) connection between two computers. The copy process is highly flexible and can be accomplished from a local machine to a remote machine or vice versa. It can also be used to copy from one remote computer to another remote computer. The `scp` command is simple to use and is a vital tool for transferring files between computers that are connected on a network.

- `scp [options] user1@sourcehost:directory/filename user2@destinationhost:directory/filename`  
: securely copy files between computers.

For example, if we had a file named 'foo.txt' in our current (source) directory and wanted to transfer it to the home directory of the user 'pi' on the destination computer with the IP address 10.1.1.33, we could execute the following command;

```
scp foo.txt pi@10.1.1.33:~/
```

If we have never tried to securely login (or `scp` files) to that computer before, we are presented with a warning asking if we're sure and is this *actually* the computer we're intending to connect to;

```
The authenticity of host '10.1.1.33 (10.1.1.33)' can't be established.  
ECDSA key fingerprint is 35:2c:ea:8e:b2:87:19:0a:40:b2:bb:81:22:01:e3:02.  
Are you sure you want to continue connecting (yes/no)? yes
```

Assuming that this is correct we can respond with 'y' and the process will continue and will ask for the password for the 'pi' user;

```
Warning: Permanently added '10.1.1.33' (ECDSA) to the list of known hosts.  
pi@10.1.1.33's password:
```

Since we have confirmed this as the correct host it gets added to a master list of known hosts so that we don't get asked if we know about it in the future. Then we are asked for the 'pi' user's password on the destination host. Once entered the transfer takes place and we are presented with details of the process;

```
foo.txt                                100% 129      0.1KB/s   00:00
```

At this point the remote host has a copy of the file 'foo.txt' in the home directory of the 'pi' user. Since we didn't specify the name of the file on the destination computer the name remained the same.

The command we used in the example employed some shortcuts and relied on defaults to shorten the amount we needed to type, but we could have also typed in the full command specifying all the information as such;

```
scp pi@10.1.1.200:/home/pi/foo.txt pi@10.1.1.33:/home/pi/foo.txt
```

The next time we perform a scp (or ssh) to the same machine (from the same local machine) we won't be asked if we're sure about the connection and we will be directed straight to the password request.

## The scp command

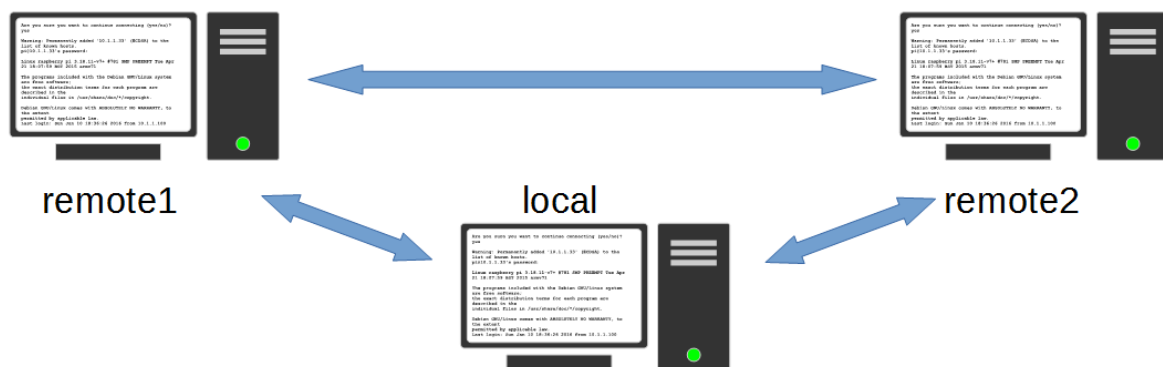
The scp command is designed to provide a user with mechanism to securely copy files between computers. There are alternative mechanisms for copying files, but scp was designed to ensure that the security of the information as it is transferred is maintained.

One of the increasingly prevailing themes in modern times is the need to provide data security. While in many cases having transparency while copying files seems acceptable, situations where privacy and / or security are warranted (medical, legal or financial records) demand a greater protection against misuse. Without going into the mechanics of it (this would be a book in itself) the scp command provides a connection between the user and a remote host that has a high degree of security against someone intercepting the information being transmitted. It provides this security by utilising the ssh command as a baseline for the connection, so understanding this command would also be useful.

The best way to illustrate the utility of scp is to provide examples of common usage. We will therefore demonstrate the scp commands for copying a file (/home/pi/foo.txt) in the following circumstances;

- From a remote host to a local host
- From a local host to a remote host
- From one remote host to another

We will assume a configuration similar to that shown below with two remote hosts ('remote1' and 'remote2') and a local host named 'local'. Likewise we will assume that the user 'pi' has accounts on all three machines.



The connected hosts for scp

### From a remote host to a local host



```
scp pi@remote1:/home/pi/foo.txt pi@local:/home/pi/foo.txt
```

### From a local host to a remote host

```
scp pi@local:/home/pi/foo.txt pi@remote1:/home/pi/foo.txt
```

### From a remote host (remote1) to a remote host (remote2)

```
scp pi@remote1:/home/pi/foo.txt pi@remote2:/home/pi/foo.txt
```

We can copy multiple files at once by separating them with a space. The following command copies `foo.txt` and `bar.txt` from the current (local) directory to the directory `/home/pi` on the server `remote1`

```
scp foo.txt bar.txt pi@remote1:/home/pi/
```

We can copy multiple files of the same type using [wildcards](#). The following command copies all the files ending in `.txt` from the current (local) directory to the directory `/home/pi` on the server `remote1`

```
scp *.txt pi@remote1:/home/pi/
```

## Options

There are a few options that can be used with `scp`. The following are particularly useful;

- `-r` will recursively copy files/directories
- `-p` will preserve file properties

To copy directories we need to employ the `-r` option which will recursively copy directories.



The word ‘recursively’ tells us that the the command will look at a directory and it will also copy any subdirectories of that directory and any subdirectories of that subdirectory ad infinitum.... (so it will get all the files and subdirectories in the specified directory).

So to copy the directory foodir from a remote machine to our local (current) directory

```
scp -r pi@remote1:/home/pi/foodir .
```

Notice in the above command we have used the period (.) symbol to tell the command to use the current directory location on the local machine.

When we copy a file we do not transfer the file properties with it. To preserve the modification time, access time and modes of the originating file we need to include the -p option.

### Test yourself

1. How many levels of subdirectories does the -r option support?
2. Does the scp command support the use of [wildcards](#)?
3. Could the scp command be used to copy files between different directories on the same remote computer?
4. Could the scp command be used to copy files between different users on the same remote computer?

## sftp

The `sftp` command is used to start an interactive program designed to provide a way of securely transferring files between two computers. The command is based on an `ssh` connection. The feature that distinguishes `sftp` from `scp` is its ability to provide an interactive option for moving files between two hosts. The `sftp` command is a secure replacement for the `ftp` command and is a useful tool for interactively transferring files between computers that are connected on a network.

- `sftp [options] [user@]host-name[:directory]` : interactively copy files between computers securely.

To start the program we can do so simply using the user and host name that we want on the remote host. The example we will use is a very simple one and assumes that we execute it as the 'pi' user and that there is an account for the 'pi' user on the remote computer;

```
sftp 10.1.1.33
```

If we have never tried to `sftp`, `ssh` or `scp` to that computer before, we are presented with a warning asking if we're sure and is this *actually* the computer we're intending to connect to;

```
The authenticity of host '10.1.1.33 (10.1.1.33)' can't be established.  
ECDSA key fingerprint is 35:2c:ea:8e:b2:87:19:0a:40:b2:bb:81:22:01:e3:02.  
Are you sure you want to continue connecting (yes/no)? yes
```

Assuming that this is correct we can respond with 'y' and the process will continue and will ask for the password for the 'pi' user;

```
Warning: Permanently added '10.1.1.33' (ECDSA) to the list of known hosts.  
pi@10.1.1.33's password:
```

Since we have confirmed this as the correct host it gets added to a master list of known hosts so that we don't get asked if we know about it in the future. Then we are asked for the 'pi' user's password on the destination host. Once entered the connection is made, interactive session is started and we are presented with an `sftp` command prompt;

```
Connected to 10.1.1.33  
sftp>
```

The command prompt allows us to interactively navigate the directory structure using common commands like `pwd`, `cd` and `ls`.

So from here we can list the contents of the directory we are in using the `ls` command;

```
sftp> ls
Desktop                               dir1
dir2                                 bar.txt
foo.txt                             foobar
foodir                             temp
```

It may not be immediately obvious, but we are in the 'pi' users home directory on 10.1.1.33. Assuming that we are interested in retrieving the file 'foo.txt', we can use the `get` command to retrieve it and to copy it back to our local working directory (the directory where we first executed the `sftp` command);

```
sftp> get foo.txt
Fetching /home/pi/foo.txt to foo.txt
/home/pi/foo.txt                100% 129    0.1KB/s   00:00
```

Having completed what we wanted to do we can now exit the interactive `sftp` program using the `bye` command;

```
sftp>bye
pi@raspberrypi:~ $
```

The next time we perform a `sftp`, `scp` or `ssh` to the same machine (from the same local machine) we won't be asked if we're sure about the connection and we will be directed straight to the password request.

The example we used assumed starting as the 'pi' user, having an account for the 'pi' user on the remote host and wanting to start in the 'pi' users home directory. Of course this will not always be the case and we could have run the command with more explicit instructions about the user to login as and the directory to go to as follows;

```
sftp pi@10.1.1.33:/home/pi
```

## The `scp` command

The File Transfer Protocol (FTP) was a widely used protocol for transferring files between computers. However, with an increasing emphasis on security and because FTP sends authentication information and file contents over the network unencrypted, it was recognised that there needed to be a better way to accomplish the same job. The `sftp` command was developed to provide a secure alternative to **FTP**. As such it mitigates security concerns by providing data transfer over a fully encrypted channel.

`sftp` is a file transfer protocol that implements all the operations found in FTP and includes some that `scp` doesn't handle, such as renaming and deleting remote files.

As shown in the example above, once the program is initiated, it is controlled by a range of commands that allow navigation through the directory structure and movement of the files. The list of available commands can be found by typing `help` or `?` at the `sftp` command prompt. The list of available commands is shown below;

Available commands:

- `bye` : Quit `sftp`
- `cd path` : Change remote directory to 'path'
- `chgrp grp path` : Change group of file 'path' to 'grp'
- `chmod mode path` : Change permissions of file 'path' to 'mode'
- `chown own path` : Change owner of file 'path' to 'own'
- `df [-hi] [path]` : Display statistics for current directory or filesystem containing 'path'
- `exit` : Quit `sftp`
- `get [-Ppr] remote [local]` : Download file
- `reget remote [local]` : Resume download file
- `reput [local] remote` : Resume upload file
- `help` : Display this help text
- `lcd path` : Change local directory to 'path'
- `lls [ls-options [path]]` : Display local directory listing
- `mkdir path` : Create local directory
- `ln [-s] oldpath newpath` : Link remote file (-s for symlink)
- `lpwd` : Print local working directory
- `ls [-lafhlNrSt] [path]` : Display remote directory listing
- `lumask umask` : Set local umask to 'umask'
- `mkdir path` : Create remote directory
- `progress` : Toggle display of progress meter
- `put [-Ppr] local [remote]` : Upload file
- `pwd` : Display remote working directory
- `quit` : Quit `sftp`
- `rename oldpath newpath` : Rename remote file
- `rm path` : Delete remote file
- `rmdir path` : Remove remote directory
- `symlink oldpath newpath` : Symlink remote file
- `version` : Show `sftp` version
- `!command` : Execute 'command' in local shell
- `!` : Escape to local shell
- `?` : Synonym for help

It's not difficult to use these commands to make sure that you can navigate and move files correctly on the remote server, but when we need to carry out functions on the local computer the `!` command is very useful to return to the local shell where we can run normal commands and then we can type `exit` to return to the `sftp` session.

**Test yourself**

1. What is the main advantage of using `sftp` over `ftp`?
2. What is the main advantage of using `sftp` over `scp`?

## ssh

The `ssh` command is used to securely access a remote machine so that a user can log on and execute commands. In spite of its significance, the `ssh` command is simple to use and is a vital tool for managing computers that are connected on a network. In situations where a computer is running 'headless' (without a screen or keyboard) it is especially useful.

- `ssh [options] [login name] server` : securely access a remote computer.

For example, operating as the user 'pi' we want to login to the computer at the network address 10.1.1.33

We initiate the process by running the command;

```
ssh 10.1.1.33
```

If we have never tried to securely login to that computer before, we are presented with a warning asking if we're sure and is this *actually* the computer we're intending to login to;

```
The authenticity of host '10.1.1.33 (10.1.1.33)' can't be established.  
ECDSA key fingerprint is 35:2c:ea:8e:b2:87:19:0a:40:b2:bb:81:22:01:e3:02.  
Are you sure you want to continue connecting (yes/no)? yes
```

Assuming that this is correct we can respond with 'y' and the process will continue and will ask for the password for the 'pi' user (since we executed the command as 'pi', the default action is to assume that we are trying to login to the remote computer as the user 'pi');

```
Warning: Permanently added '10.1.1.33' (ECDSA) to the list of known hosts.  
pi@10.1.1.33's password:
```

Since we have confirmed this as the correct host it gets added to a master list of known hosts so that we don't get asked if we know about it in the future. Then we are asked for the 'pi' user's password on the remote host. Once entered we are logged in and presented with some details of the machine we are connecting to;

```
Linux raspberrypi 3.18.11-v7+ #781 SMP PREEMPT Tue Apr 21 18:07:59 BST 2015 a\
rmv7l
```

The programs included with the Debian GNU/Linux system are free software; the exact distribution terms for each program are described in the individual files in `/usr/share/doc/*/copyright`.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent permitted by applicable law.

Last login: Sun Jan 10 18:36:26 2016 from 10.1.1.100

At this point we are logged into the computer at 10.1.1.33 and can execute commands as the user 'pi' on that machine.

To close this connection we can simply press the key combination CTRL-d and the connection will close with the following message;

```
pi@raspberrypi ~ $ logout
Connection to 10.1.1.33 closed.
```

The next time we login to the same machine we won't be asked if we're sure about the connection and we will be directed straight to the password request.

## The `ssh` command

The `ssh` command is designed to provide a user with secure access to a **shell** on a remote computer. There are a number of different ways that users can interact with networked computers, but one of the key functions that needs to be enabled is the ability to execute commands as if you were sitting at a terminal with a keyboard directly attached to the machine.

Often this isn't possible because the computer is located in another room or even another country. Sometimes the machine has no keyboard or monitor connected and is sitting in a rack or on a shelf. Sometimes the server will be a virtual machine with no 'real' hardware at all. For these instances we still want to be able to connect to the machine in some way and execute commands.

There are different commands and programs that will provide this remote access, but one of the increasingly prevailing themes is the need to provide a secure connection between the remote host and the user. This is because control of those remote machines needs to be limited to those who are authorised to carry out work on the machines for the safe operation of the functions they are carrying out (think of remotely controlling an electricity supply or traffic control computers). Without going into the mechanics of it (this would be a book in itself) the `ssh` command provides a connection between the user and a remote host that has a high degree of security against someone intercepting the information being transmitted.

The example above demonstrated the connection to a remote server as the currently logged in user. However, if we wanted to log in as a different user (let's call the user 'newpi') we would execute the command as follows;



```
ssh newpi@10.1.1.33
```

The two examples thus far have used IP addresses to designate the host-name, however, if the remote host had a designated name 'pinode' the same command could be entered as follows;

```
ssh newpi@pinode
```

There are a huge number of uses which the ssh command makes possible, but for a new user, the takeaway should be that access between two servers can be carried out easily and the information that is transferred can be done so securely using the ssh command.

### Test yourself

1. What are the two main functions of the ssh command?

## traceroute

The `traceroute` command is used to determine the route that IP packets originating from our system follow to reach a destination host. A large network will include routers to direct packets from one destination to another. `traceroute` will allow us to see the path that the packets take like a trail of breadcrumbs. `traceroute` is a network diagnosis program that has been developed to troubleshoot problems where connections are broken or to identify routers and / or devices in the network path.

- `traceroute [options] host-name` : trace the route packets take to network host

In it's simplest form, `traceroute` only requires the command and the name or IP address of the destination host;

```
traceroute 8.8.8.8
```

The IP address 8.8.8.8 is the address of Google's public Domain Name System (DNS)

The returned information shows a list of the different 'hops' that the packets took to reach the server 8.8.8.8.

```
traceroute to 8.8.8.8 (8.8.8.8), 30 hops max, 60 byte packets
 1  10.1.1.201 (10.1.1.201)  8.421 ms  8.243 ms  8.140 ms
 2  jetstream.xtra.co.nz (125.239.10.1)  41.724 ms  42.742 ms  43.633 ms
 3  * * *
 4  ae8-10.gateway.net.nz (122.56.116.5)  57.666 ms  58.964 ms  59.604 ms
 5  ae5-2.gateway.net.nz (210.55.202.213)  60.572 ms  61.427 ms  63.133 ms
 6  xe7-0-9.gateway.net.nz (202.50.232.182)  87.727 ms  76.504 ms  76.583 ms
 7  ae2-10.gateway.net.nz (202.50.232.246)  77.858 ms  75.211 ms  75.970 ms
 8  google.gateway.net.nz (202.50.237.198)  78.081 ms  77.741 ms  79.743 ms
 9  72.14.237.11 (72.14.237.11)  81.398 ms  81.768 ms  82.599 ms
10  216.239.41.81 (216.239.41.81)  82.984 ms  83.978 ms  75.562 ms
11  google-public-dns-a.google.com (8.8.8.8)  76.806 ms  76.423 ms  75.224 ms
```

Each line shows the Time To Live (aka TTL or the 'hop limit') which gets incremented as the path progresses, the IP address and if available the host name of the router it encounters along with the Round Trip Time (RTT) for that particular router. There are three RTTs per router as `traceroute` sends three packets per router.

The third position shows three asterisks (\* \* \*) which tells us that a particular reading could not be returned. This could be for any one of a number of reasons, but in this case we can see that the omission did not stop the packets from completing their journey.

## The `traceroute` command

The `traceroute` program is so named because it is attempting the **trace** the **route** of packets as they flow through a network.

Some networks and the Internet in particular are large and complex combinations of network hardware, connected together by gateways, routers and switches. Tracking the route of our network packets (or finding the gateway that's dropping our traffic) can be difficult. `traceroute` uses a 'time to live' field in the the IP protocol to attempt to prompt an ICMP `TIME_EXCEEDED` response from each gateway in the chain to a remote host.

In our modern network environments, sometimes `traceroute` may not work as expected. This is because of widespread use of firewalls. Some firewalls will filter 'unlikely' UDP ports, or even the ICMP `TIME_EXCEEDED` responses. To work around this, some additional `traceroute` methods can be used.

Each of them works slightly differently. But the overall concept behind each of them is the same. All of them use the TTL value.

- `U` : The default method using UDP.
- `I` : Uses ICMP ECHO for probes.
- `T` : Uses TCP SYN for probes

It is possible that running `traceroute` with the `-I` or `-T` options may require [sudo](#) permissions. In which case the command would look like this;

```
sudo -I traceroute 8.8.8.8
```

## Test yourself

1. What sort of problems can `traceroute` highlight?
2. What could you try if it appears that `traceroute`'s default test isn't working at some point?

## wget

The `wget` command (or perhaps a better description is ‘utility’) is a program designed to make downloading files from the Internet easy using the command line. It supports the HTTP, HTTPS and FTP protocols and is designed to be robust to accomplish its job even on a network connection that is slow or unstable. It is similar in function to `curl` for retrieving files, but there are some key differences between the two, with the main one for `wget` being that it is capable of downloading files recursively (where resources are linked from web pages).

- `wget [options] [URL]` : download or upload files from the web non-interactively.

In it's simplest example of use it is only necessary to provide the URL of the file that is required and the download will begin;

```
wget https://github.com/mbostock/d3/archive/master.zip
```

The program will then connect to the remote server, confirm the file details and start downloading the file;

```
--2016-02-07 09:08:47-- https://github.com/mbostock/d3/archive/master.zip
Resolving github.com (github.com)... 192.30.252.131
Connecting to github.com (github.com)|192.30.252.131|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://codeload.github.com/mbostock/d3/zip/master [following]
--2016-02-07 09:09:07-- https://codeload.github.com/mbostock/d3/zip/master
Resolving codeload.github.com (codeload.github.com)... 192.30.252.161
Connecting to codeload.github.com |192.30.252.161|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [application/zip]
Saving to: 'master.zip.1'
```

As the downloading process proceeds a simple text animation advises of the progress with an indication of the amount downloaded and the rate

```
master.zip.1          [          <=>          ] 990.35K  53.6KB/s
```

Once complete the successful download will be reported accompanied by some statistics of the transfer;

```
2016-02-07 09:10:27 (50.3 KB/s) - 'master.zip.1' saved [3204673]
```

The file is downloaded into the current working directory.

## The `wget` command

`wget` is a utility that exists slightly out of the scope of a pure command in the sense that it is an Open Source program that has been compiled to work on a range of operating systems. The name is a derivation of **web get** where the function of the program is to 'get' files from the world wide web.

It does this via support for the HTTP, HTTPS and FTP protocols such that if you could paste a URL in a browser and have it subsequently download a file, the same file could be downloaded from the command line using `wget`. `wget` is not the only file downloading utility that is commonly used in Linux. `curl` is also widely used for similar functions. However both programs have different strengths and in the case of `wget` that strength is in support of recursive downloading where an entire web site could be downloaded while maintaining its directory structure and links. There are other differences as well, but this would be the major one.

There is a large range of options that can be used to ensure that downloads are configured correctly. We will examine a few of the more basic examples below and after that we will check out the recursive function of `wget`.

- `--limit-rate` : **limit** the download speed / download **rate**.
- `-O` : download and store with a different file name
- `-b` : download in the background
- `-i` : download multiple files / URLs
- `--ftp-user` and `--ftp-password` : FTP download using `wget` with username and password authentication

### Rate limit the bandwidth

There will be times when we will be somewhere that the bandwidth is limited or we want to prioritise the bandwidth in some way. We can restrict the download speed with the option `--limit-rate` as follows;

```
wget --limit-rate=20k https://github.com/mbostock/d3/archive/master.zip
```

Here we've limited the download speed to 20 kilo bytes per second. The amount may be expressed in bytes, kilobytes with the `k` suffix, or megabytes with the `m` suffix.

### Rename the downloaded file

If we try to download a file with the same name into the working directory it will be saved with an incrementing numerical suffix (i.e. `.1`, `.2` etc). However, we can give the file a different name when downloaded using the `-O` option (that's a capital 'o' by the way). For example to save the file with the name `alpha.zip` we would do the following;

```
wget -O alpha.zip https://github.com/mbostock/d3/archive/master.zip
```

### Download in the background

Because it may take some considerable time to download a file we can tell the process to run in the background which will release the terminal to carry on working. This is accomplished with the `-b` option as follows;

```
wget -b https://github.com/mbostock/d3/archive/master.zip
```

While the download continues, the progress that would normally be echoed to the screen is passed to the file `wget-log` that will be in the working directory. We can check this file to determine progress as necessary.

### Download multiple files

While we can download multiple files by simply including them one after the other in the command as follows;

```
wget https://website.org/file1.zip https://website.org/file2.zip
```

While that is good, it can start to get a little confusing if a large number of URL's are included. To make things easier, we can create a text file with the URL's/names of the files we want to download and then we specify the file with the `-i` option.

For example, if we have a file named `files.txt` in the current working directory that has the following contents;

```
https://github.com/d3/d3-dispatch/blob/master/src/dispatch.js
https://github.com/d3/d3-selection/blob/master/src/select.js
https://github.com/d3/d3-dsv/blob/master/src/csv.js
https://github.com/d3/d3-scale/blob/master/src/time.js
https://github.com/d3/d3-color/blob/master/src/color.js
https://github.com/d3/d3-axis/blob/master/src/axis.js
https://github.com/d3/d3-time/blob/master/src/interval.js
```

Then we can run the command...

```
wget -i files.txt
```

... and it will work through each file and download it.

### Download files that require a username and password

The examples shown thus far have been able to be downloaded without providing any form of authentication (no user / password). However this will be a requirement for some downloads. To include a username and password in the command we include the `--ftp-user` and `--ftp-password` options. For example if we needed to use the username 'adam' and the password '1234' we would form our command as follows;

```
wget --ftp-user=adam --ftp-password=1234 ftp://website.org/file.zip
```

You may be thinking to yourself "Is this secure?". To which the answer should probably be "No". It is one step above anonymous access, but not a lot more. This is not a method by which things that should remain private should be secured, but it does provide a method of restricting anonymous access.

### Download files recursively

One of the main features of `wget` is it's ability to download a large complex directory structure that exists on many levels. The best example of this would be the structure of files and directories that exist to make up a web site. While there is a wide range of options that can be passed to make the process work properly in a wide variety of situations, it is still possible to use a fairly generic set to get us going.

For example, to download the contents of the web site at `dnoob.runkite.com` we can execute the following command;

```
wget -e robots=off -r -np -c -nc http://dnoob.runkite.com/
```

The options used here do the following;

- `-e robots=off` : the `execute` option allows us to run a separate command and in this case it's the command `robots=off` which tells the web site that we are visiting that it should ignore the fact that we're running a command that is acting like a robot and to allow us to download the files.
- `-r` : the `recursive` option enables recursive downloading.
- `-np` : the `no-parent` option makes sure that a recursive retrieval only works on pages that are *below* the specified directory.
- `-c` : the `continue` option ensures that any partially downloaded file continues from the place it left off.
- `-nc` : the `no-clobber` option ensures that duplicate files are not overridden

Once entered the program will display a running listing of the progress and a summary telling us how many file and the time taken at the end. The end result is a directory called

dnoob.runkite.com in the working directory that has the entire website including all the linked pages and files in it. If we examine the directory structure it will look a little like the following;

```
dnoob.runkite.com/
├── assets
│   ├── css
│   │   └── screen.css?v=bb61fe2
│   ├── fonts
│   │   ├── icons.svg
│   │   └── icons.woff
│   └── js
│       ├── index.js?v=bb61fe2
│       └── jquery.fitvids.js?v=bb61fe2
├── content
│   └── images
│       └── 2015
│           ├── 09
│           │   ├── BplusandB2.png
│           │   ├── piblog-01.png
│           │   └── RPiAlpha-1.png
│           └── 10
│               ├── board-02.png
│               ├── terminal.png
│               └── wheezy-raspbian-file.png
├── everything-is-a-file-in-linux
│   └── index.html
├── favicon.ico
├── index.html
├── linux-files-and-inodes
│   └── index.html
├── public
│   └── jquery.min.js?v=bb61fe2
├── regular-expressions-in-linux
│   └── index.html
├── un
│   └── index.html
└── welcome-to-raspbian-debian-wheezy
    └── index.html
```

Using `wget` for recursive downloading should be used appropriately. It would be considered poor manners to pillage a web site for anything other than good reason. When in doubt contact the person responsible for a site or a repository just to make sure there isn't a simpler way that you might be able to accomplish your task if it's something 'weird'.

## Test yourself

1. Craft a `wget` command that downloads a file to a different name, limiting the download



rate to 10 kilobytes per second and which operates in the background.

2. Once question 1 above is carried out, where do we find the output of the downloads progress?

## Miscellaneous

### apt-get

The `apt-get` command is a program, that is used with Debian based Linux distributions to install, remove or upgrade software packages. It's a vital tool for installing and managing software and should be used on a regular basis to ensure that software is up to date and security patching requirements are met.

There are a plethora of uses for `apt-get`, but we will consider the basics that will allow us to get by. These will include;

- Updating the database of available applications (`apt-get update`)
- Upgrading the applications on the system (`apt-get upgrade`)
- Installing an application (`apt-get install *package-name*`)
- Un-installing an application (`apt-get remove *package-name*`)

### The `apt-get` command

The `apt` part of `apt-get` stands for 'advanced packaging tool'. The program is a process for managing software packages installed on Linux machines, or more specifically [Debian](https://www.debian.org/)<sup>22</sup> based Linux machines (Since those based on '[redhat](http://www.redhat.com/)'<sup>23</sup> typically use their `rpm` (red hat package management (or more lately the recursively named 'rpm package management') system). As Raspbian is based on Debian, so the examples we will be using are based on `apt-get`.

APT simplifies the process of managing software on Unix-like computer systems by automating the retrieval, configuration and installation of software packages. This was historically a process best described as 'dependency hell' where the requirements for different packages could mean a manual installation of a simple software application could lead a user into a sink-hole of despair.

In common `apt-get` usage we will be prefixing the command with `sudo` to give ourselves the appropriate permissions;

#### `apt-get update`

```
sudo apt-get update
```

This will resynchronize our local list of packages files, updating information about new and recently changed packages. If an `apt-get upgrade` (see below) is planned, an `apt-get update` should always be performed first.

Once the command is executed, the computer will delve into the internet to source the lists of current packages and download them so that we will see a list of software sources similar to the following appear;

---

<sup>22</sup><https://www.debian.org/>

<sup>23</sup><http://www.redhat.com/>

```
pi@raspberrypi ~ $ sudo apt-get update
Hit http://raspberrypi.collabora.com wheezy Release.gpg
Get:1 http://mirrordirector.raspbian.org wheezy Release.gpg [490 B]
Get:2 http://archive.raspberrypi.org wheezy Release.gpg [473 B]
Hit http://raspberrypi.collabora.com wheezy Release
Get:3 http://mirrordirector.raspbian.org wheezy Release [14.4 kB]
Get:4 http://archive.raspberrypi.org wheezy Release [17.6 kB]
Hit http://raspberrypi.collabora.com wheezy/rpi armhf Packages
Get:5 http://mirrordirector.raspbian.org wheezy/main armhf Packages [6,904 kB]
Get:6 http://archive.raspberrypi.org wheezy/main armhf Packages [130 kB]
Ign http://raspberrypi.collabora.com wheezy/rpi Translation-en
Ign http://mirrordirector.raspbian.org wheezy/contrib Translation-en
Ign http://mirrordirector.raspbian.org wheezy/main Translation-en
Ign http://mirrordirector.raspbian.org wheezy/non-free Translation-en
Ign http://mirrordirector.raspbian.org wheezy/rpi Translation-en
Fetched 7,140 kB in 35s (200 kB/s)
Reading package lists... Done
```

### **apt-get upgrade**

```
sudo apt-get upgrade
```

The `apt-get upgrade` command will install the newest versions of all packages currently installed on the system. If a package is currently installed and a new version is available, it will be retrieved and upgraded. Any new versions of current packages that cannot be upgraded without changing the install status of another package will be left as they are.

As mentioned above, an `apt-get update` should always be performed first so that `apt-get upgrade` knows which new versions of packages are available.

Once the command is executed, the computer will consider its installed applications against the databases list of the most up to date packages and it will prompt us with a message that will let us know how many packages are available for upgrade, how much data will need to be downloaded and what impact this will have on our local storage. At this point we get to decide whether or not we want to continue;

```
pi@raspberrypi ~ $ sudo apt-get upgrade
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages will be upgraded:
  bind9-host cups-bsd cups-client cups-common libapache2-mod-php5 libbind9-80
  libisccc80 libisccfg82 liblwres80 libsd11.2debian libsqlite3-0 libssl1.0.0
  php5-mcrypt php5-mysql raspi-config
6 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
```

Need to get 10.7 MB of archives.  
After this operation, 556 kB disk space will be freed.  
Do you want to **continue** [Y/n]?

Once we say yes ('Y') the upgrade kicks off and we will see a list of the packages as they are downloaded unpacked and installed (what follows is an edited example);

```
Do you want to continue [Y/n]? y
Get:1 http://archive.raspberrypi.org/debian/wheezy/main libssl1.2debian
armhf 1.2.15-5+rpi1 [205 kB]
Get:2 http://archive.raspberrypi.org/debian/wheezy/main raspi-config all
20150131-5 [13.3 kB]
Get:3 http://mirrordirector.raspbian.org/raspbian/ wheezy/main libsqlite3-0
armhf 3.7.13-1+deb7u2 [414 kB]
Fetched 10.7 MB in 31s (343 kB/s)
Preconfiguring packages ...
(Reading database ... 80703 files and directories currently installed.)
Preparing to replace cups-common 1.5.3-5+deb7u5
(using .../cups-common_1.5.3-5+deb7u6_all.deb) ...
Unpacking replacement cups-common ...
Preparing to replace cups-bsd 1.5.3-5+deb7u5
(using .../cups-bsd_1.5.3-5+deb7u6_armhf.deb) ...
Unpacking replacement cups-bsd ...
Preparing to replace php5-gd 5.4.39-0+deb7u2
(using .../php5-gd_5.4.41-0+deb7u1_armhf.deb) ...
Unpacking replacement php5-gd ...
Processing triggers for man-db ...
Setting up libssl1.0.0:armhf (1.0.1e-2+rvt+deb7u17) ...
Setting up libsqlite3-0:armhf (3.7.13-1+deb7u2) ...
Setting up cups-common (1.5.3-5+deb7u6) ...
Setting up cups-client (1.5.3-5+deb7u6) ...
```

There can often be alerts as the process identifies different issues that it thinks the system might strike (different aliases, runtime levels or missing fully qualified domain names). This is not necessarily a sign of problems so much as an indication that the process had to take certain configurations into account when upgrading and these are worth noting. Whenever there is any doubt about what has occurred, Google will be your friend :-).

### **apt-get install**

The `apt-get install` command installs or upgrades one (or more) packages. All additional (dependency) packages required will also be retrieved and installed.

```
sudo apt-get install *package-name*
```

If we want to install multiple packages we can simply list each package separated by a space after the command as follows;

```
sudo apt-get install package1 package2 package3
```

### **apt-get remove**

```
sudo apt-get remove *package-name*
```

The `apt-get remove` command removes one (or more) packages.

### **Test yourself**

1. How could you install a range of packages with similar names (for example all packages starting with 'mysql')?
2. When removing a package, is the configuration for that package retained or deleted (Google will be your friend)?
3. How could you remove multiple packages with a single command?

## **clear**

The `clear` command is used to remove the commands and output from the terminal window and present a clear screen. This is a convenient way to clear away a ‘mess’ to show a blank screen (apart from a command prompt).

- `clear` : clears the terminal screen.

This is one of the simplest commands with no options or arguments.

```
clear
```

The result of which is a clear screen with just the command prompt in the top left corner

## **The `clear` command**

The `clear` command has no options or arguments to distract us. Its mission is to simply make our lives easier by **clearing** the screen.

Be aware that while it might clear the contents of the screen, our command history is still accessible, so it's not guarantee of concealment.

It also has a short-cut in the form of the key combination of ‘CTRL-L’ (that's a lower case ‘L’ by the way).

## **Test yourself**

1. What other methods might be available to determine what was on the screen before it was blanked?

## echo

The echo command is widely used in writing [scripts](#) and at the command line for displaying information on the screen. While advanced users will use this command extensively, it is also useful for users starting out so that they have a good grasp of the context of the commands they are using and as a building block for further advancement.

- echo [options] [string(s)] : display text on the screen.



Each shell has a slightly different interpretation of echo, so while the principles remain the same, be aware that there will be inevitable variations somewhere along the track. For the purposes of the explanation below we will assume the use of the 'bash' shell which is predominantly the default shell in Linux distributions.

echo can be simply demonstrated via running something like the following command;

```
echo My first echo!
```

This will produce the following on the next line;

```
My first echo!
```

This example of use might seem a little trite, but the real strength comes with the application of variables and formatting options.

## The echo command

While echo can be considered a command and can be used from the command line easily, its real power is revealed in running [scripts](#). As part of a [scripts](#) the echo command can allow us to present a user with feedback and even gather input to allow greater utility and control.

echo only has three options of note and they all concern the formatting of the output;

- -n : do not output a trailing newline character
- -e : enable the interpretation of backslash escape sequences
- -E : disable the interpretation of backslash escape sequences (the default)

The application of a trailing newline character means that whenever echo outputs something to the screen it immediately appends a new line to the output so that the next thing that appears on the screen will occur on a new line. For example running the following command with the -n option;

```
echo -n My first echo!
```

will see the following line appear as such;

```
My first echo!pi@raspberrypi:~ $
```

This shows the command prompt (pi@raspberrypi:~ \$) with the username and system name immediately trailing the echoed string;

Enabling backslash characters with the -e option allows us to include a range of formatting characters that will allow us to present information on the screen in a more flexible way. The characters are;

- \a : alert (bell)
- \b : backspace
- \c : suppress further output
- \e : escape character
- \f : form feed
- \n : new line
- \r : carriage return
- \t : horizontal tab
- \v : vertical tab
- \\ : backslash
- \0nnn : the character whose ASCII code is nnn (octal). nnn can be 0 to 3 octal digits
- \xHH : the eight-bit character whose value is HH (hexadecimal). HH can be one or two hex digits

As an example, the following command;

```
echo -e "My \nfirst \necho! \n\t(now \twith \ttabs)"
```

Will produce the following output;

```
My
first
echo!
      (now      with      tabs)
```





The string in the command is enclosed in speech-marks. Without these the backslash and parenthesis characters are not interpreted as we would like.

The second string to echo's bow is it's ability to report back variable values. we can easily set variable values in the shell (or a script) which can then be printed using echo as follows;

```
x=1234  
echo $x
```

The command will then report back the value in the variable 'x';

```
1234
```

Likewise we can use the built in environment variables to display more about our system;

```
x=1234  
echo $HOME
```

The command will then report back the current users home directory;

```
/home/pi
```

We can find a list of the environment variables using the env command.

## Test yourself

1. Why did we not have to use parenthesis in our first example command `echo My first echo!?`
2. Craft an echo command that combines a string, a local variable (x) and an environmental variable.

## groupadd

The `groupadd` command is used by a superuser (typically via `sudo`) to add a group account. It's a fundamental command in the sense that Linux as a complex multi user system required a mechanism for adding groups. Not that this command will get used every day, but it's important to know to enable us to administer users on the computer.

- `groupadd [options] group` : add a group account

The following example is the simplest application of the command;

```
sudo groupadd pigroup
```

This creates the group 'pigroup' using the default values from the system.

## The `groupadd` command

The `groupadd` command is not utilised on a regular basis but it is useful to recognise its role. It adds a group account using the default values from the system. The new group will be entered into the system files as needed.. It can only be used by the root user, so it is often prefixed by the `sudo` command. Group names may only be up to 32 characters long and if the group name already exists in an external group database such as NIS or LDAP, `groupadd` will deny the group creation request.

When `groupadd` is executed it does so utilising the following files;

- `/etc/group` : Group account information.
- `/etc/gshadow` : Secure group account information.
- `/etc/login.defs` : Shadow password suite configuration.

## Test yourself

1. What other command that we have looked at utilises the files above?

## man

The `man` command is used to format and display documentation (or manuals) on other commands. It is one of the most important commands from the perspective of providing an always available reference for how a command needs to be formatted or used.

- `man [options] command` : display the reference manual for a command.

The `man` command is mostly used without any options and with just the command that we want to find information on. For example to get information on the `ls` command we simply type in;

```
man ls
```

The first page of the manual for the `ls` command will then be displayed;

```
LS(1)                                User Commands                                LS(1)

NAME
    ls - list directory contents

SYNOPSIS
    ls [OPTION]... [FILE]...

DESCRIPTION
    List information about the FILES (the current directory by default).
    Sort entries alphabetically if none of -cftuvSUX nor --sort is
    specified.

    Mandatory arguments to long options are mandatory for short options
    too.

    -a, --all
        do not ignore entries starting with .
```

We can advance down the page (and back up) one line at a time with the arrow keys or move a little quicker one page at a time with the page-up / page-down keys.

At any stage we can press 'q' to quit the `man` program and return to the command line.

## The `man` command

The `man` command displays the documentation (or **manual**) available on the computer for a specified command. Think of it as a form of online documentation which is available on Unix / Linux operating systems. The `man` command can display documentation on different topics apart

from commands. These are called sections and include System Calls, Library functions, Devices, File formats, Games and Amusements, Conventions and Miscellany, System Administration and Privileged Commands and TCL commands. For the purposes of an introduction however, a newer user will identify the `man` command as being the reference for executable programs (which is section 1).

An individual command's documentation is typically referred to as a "man page". The page will normally be made up of a consistent range of sections such as;

- **NAME** : The name of the command itself and a short description of what the command does.
- **SYNOPSIS** : This is an outline describing how the command is supposed to be used.
- **DESCRIPTION** : A more detailed description of the command including the options available.
- **AUTHORS** : The people who wrote or assisted in the writing of the command.
- **BUGS** : Any known problems with the programs. Most often these aren't errors, but limitations.
- **ENVIRONMENT** : Any variables that might be required or limitations in the version of the shell that may be needed.
- **EXAMPLES or NOTES** : Examples of usage and general notes.
- **REPORTING BUGS** : If we find problems with the command, this tells us where we can report them.
- **COPYRIGHT** : This is the person or organization that holds the copyright to the command.
- **SEE ALSO** : Which suggests other commands that are related to this command.

Because Linux is an operating system that is a collaborative endeavour, the final product is a result of many different people contributing in different ways. The man pages are no exception and as a result we will find that some are more fully developed than others. While there is a degree of consistency, there will always be variations in the depth or completeness of the description. It is not intended that a man page be the complete repository of all knowledge on a command, but it is certainly a good first place to start.

For help when executing the `man` command, press the 'h' key. This will provide some assistance in navigating the command and will most likely be a reproduction of the commands that the text display program that `man` uses. The following is the help page displayed while using the default Raspbian OS and shows that it uses `less` to display the pages;

## SUMMARY OF LESS COMMANDS

Commands marked with \* may be preceded by a number, N.

Notes in parentheses indicate the behavior if N is given.

A key preceded by a caret indicates the Ctrl key; thus ^K is ctrl-K.

```
h H          Display this help.
q :q Q :Q ZZ  Exit.
```

---

## MOVING

```
e ^E j ^N CR * Forward one line (or N lines).
y ^Y k ^K ^P * Backward one line (or N lines).
f ^F ^V SPACE * Forward one window (or N lines).
b ^B ESC-v    * Backward one window (or N lines).
z             * Forward one window (and set window to N).
w             * Backward one window (and set window to N).
ESC-SPACE    * Forward one window, but don't stop at end-of-file.
d ^D         * Forward one half-window (and set half-window to N).
u ^U         * Backward one half-window (and set half-window to N).
ESC-) RightArrow * Left one half screen width (or N positions).
HELP -- Press RETURN for more, or q when done
```

**Test yourself**

1. Does the man command display the definitive reference material for a command?
2. Do you have to be connected to the Internet to display man pages?

## modprobe

The `modprobe` command allows us to add (or remove) modules to the [Linux Kernel](#)<sup>24</sup>. The Linux kernel is the code that forms the core of the Linux operating system, so it's kind of important. When changing hardware, the `modprobe` command allows us to import or remove the equivalent of windows device drivers to / from the kernel to enable / disable additional functionality.

- `modprobe [options] [modulename]` : Load or remove a Linux kernel module

For example to add the module `w1-therm` to support measurement of temperature via the 1-Wire bus we would execute the following command;

```
sudo modprobe w1-therm
```



It is necessary to do this as the superuser via [sudo](#).

## The `modprobe` command

The Linux kernel is designed with a monolithic structure, but with the ability to be able to change kernel modules while running. (Windows 7 and OS X use hybrid kernels which offer the advantage of being smaller in size, but they require more management of the drivers by the user and manufacturer).

To work around the disadvantage of having a large footprint, Linux kernel developers have incorporated the facility to add or remove kernel modules on the fly. This can be taken to the extreme where the the entire kernel module can be replaced without needing to reboot.

Kernel modules are typically located in the `lib/modules` directory and can be listed using the following command;

```
ls /lib/modules/$(uname -r)
```

An output might look something like the following;

---

<sup>24</sup><http://www.howtogeek.com/howto/31632/what-is-the-linux-kernel-and-what-does-it-do/>

```
pi@raspberrypi /lib/modules $ ls /lib/modules/$(uname -r)
kernel          modules.builtin  modules.dep.bin  modules.softdep
modules.alias    modules.builtin.bin  modules.devname  modules.symbols
modules.alias.bin  modules.dep      modules.order    modules.symbols.bin
```

We can also list all the loaded modules using the command `lsmod` as follows;

```
lsmod
```

A sample output might look similar to the following;

```
pi@raspberrypi /lib/modules $ lsmod
Module                Size  Used by
w1_therm              2559  0
wire                 25680  1 w1_therm
cn                   4636  1 wire
rfkill              16651  1 cfg80211
i2c_dev              6027  0
snd_bcm2835         18649  0
snd_pcm             73475  1 snd_bcm2835
snd_seq             53078  0
snd_seq_device        5628  1 snd_seq
snd_timer           17784  2 snd_pcm,snd_seq
snd                 51038  5 snd_bcm2835,snd_timer,snd_pcm,snd_seq,snd_seq\
_device
i2c_bcm2708          4990  0
```

We can also see the range of drivers that are available via the command;

```
ls /lib/modules/$(uname -r)/kernel/drivers/
```

Which would provide an output similar to the following;

```
pi@raspberrypi /lib/modules $ ls /lib/modules/$(uname -r)/kernel/drivers/
base      cdrom      extcon     input      mfd        nfc        rtc        staging    w1
bcma      char       gpio      leds       misc       of         scsi       uio        watchdog
block     connector  hid       md         mmc        power     spi        usb
bluetooth cpufreq   i2c       media      net        pps       ssb        video
```

## Options

The `modprobe` command has several options, but the vast majority of users will simply need to install a module which is done using the `sudo modprobe [modulename]` command (no options needed).

The only realistic command option that a novice user might use would be `-r` which would remove a module.

```
sudo modprobe -r w1-therm
```

## Arguments

The module name is the main argument used when executing the `modprobe` command. Multiple modules can be loaded by simply putting a space between the module names.

## Test yourself

1. What type of module is used in the Linux kernel vs the Windows kernel and what is the advantage of the Linux kernel approach
2. Show how the modules `mod1`, `mod2` and `mod3` can all be loaded using one use of the `modprobe` command.



## passwd

The `passwd` command is a vital system administrator tool to manage the passwords for user accounts. We can change our own account, or as the root user we can change other accounts.

- `passwd [options] username` : change the user password

To change the current user account all we need to do is execute the `passwd` command;

```
passwd
```

To allow our password to be changed, first we need to enter the password we're *going* to change;

```
Changing password for newpi.  
(current) UNIX password:
```

The we enter the new password;

```
Enter new UNIX password:
```

Then we enter the new password again to confirm that we typed it right the first time;

```
Retype new UNIX password:
```

Then we're told that the update has been successful;

```
passwd: password updated successfully
```

This is one of the simplest tasks using the `passwd` command, but there are others that engage a greater degree of complexity for user and password management.

## The `passwd` command

The `passwd` command allows us to change passwords for user accounts. As a normal user we can only change the password for our own account, while as the superuser ('root') we can change the password for any account. The `passwd` command can also change the properties of an accounts password via options such as;

- `-n` : set the minimum number of days between password changes
- `-x` : set the maximum number of days a password remains valid
- `-w` : Set the number of days of warning before a password change is required

To view the properties for our password we can use the `chage` command with the `-l` option to list the details of the user 'newpi's' password;

```
chage -l newpi
```

This will show us details similar to the following;

```
Last password change          : Feb 13, 2016
Password expires              : May 13, 2016
Password inactive             : never
Account expires               : never
Minimum number of days between password change : 0
Maximum number of days between password change : 90
```

We can set the minimum number of days between password changes (via `-n`), the maximum number of days a password remains valid (via `-x`) or the number of days of warning before a password change is required (via `-w`) as follows;

```
sudo passwd -n 0 newpi
sudo passwd -x 90 newpi
sudo passwd -w 7 newpi
```



We need to set these using `sudo` as it constitutes a system administration function that should be carried out by a user with `sudo` privileges.

The other administrative functions revolve around administration of other users passwords via the following options;

- `l` : locking a users password `sudo passwd -l newpi`
- `u` : unlocking a users password `sudo passwd -u newpi` (the password used previous to locking is available again)
- `e` : expire a users password `sudo passwd -l newpi` (forces them to change their password at next login)

## Test yourself

1. Can you change your own number of days of warning before a password change if you aren't on the `sudo-ers` list?
2. What is the maximum number of days between a password change?

## shutdown

The shutdown command allows us to manage the shut-down of the system from the command line in a variety of ways. This command allows a graceful exit from the system and has the option to alert other users so that they know what is going on. In a multi user system or as a method to avoid pressing the power button it is good practice to ensure that the system has every opportunity to exit in a managed way.

- shutdown [options] [time] [message] : shut down or restart a system.

shutdown must be executed as an administrative user and when carried out without options or arguments as follows;

```
sudo shutdown
```

... it will initiate a broadcast message to the users on the system that the computer will be shut down in one minute;

```
Shutdown scheduled for Sun 2016-02-14 18:00:30, use 'shutdown -c' to cancel
pi@raspberrypi:~ $
Broadcast message from root@raspberrypi (Sun 2016-02-14 17:59:30 NZDT):
```

```
The system is going down for power-off at Sun 2016-02-14 18:00:30 NZDT!
```

Then after a minute the following will appear as the system shuts down;

```
Broadcast message from root@raspberrypi (Sun 2016-02-14 18:00:34 NZDT):
```

```
The system is going down for power-off NOW!
```

In this scenario the system has an opportunity to exit gracefully and wherever possible every attempt is made to preserve any data or configuration (not that this can be taken for granted).

## The shutdown command

The shutdown command is designed to shut the system down in a managed and secure way. Logged-in users are alerted that the system is going down which provides an opportunity to save open work and to prepare for the outage.

We can vary the amount of time to delay before shutting down all the way down to no delay at all and we can vary the state that exists after shut down to reboot, halt or power down.

These different actions and more are available via the options as follows;

- -h : shut down the system and then halt
- -P : shut down the system and then power down
- -r : reboot after shut down
- -c : cancel a pending shut down.

These options can be accompanied by a '[time]' argument to specify when to perform the action; For example the following command will shut down the system and power it down in 5 minutes;

```
sudo shutdown -P +5
```

In addition we can include a '[message]' argument that will allow a custom message to be broadcast to alert users of the impending shut down. For example, to shut down and halt the system and to print out a custom message;

```
sudo shutdown -h "The system will be halting in one minute. Save your work now!\nw!"
```

In addition, we can carry out the shut down immediately by using either now or +0 as a time. For example to reboot immediately;

```
sudo shutdown -r now
```

## Test yourself

1. How long do we have to cancel a shutdown given the default time and what option do we use?

## su

The command `su` is used to switch from one account to another without needing to change out of the session that the original user is in. While it can be used to change to any user, it is most often used to change to the 'root' user to carry out administrative commands. The default action of `su` if invoked without a username is to change to the 'root' user. Getting a good understanding of `su` is an excellent step to make our lives easier administering the system.

- `su [options] [username]` : switch to another user.

To substitute the current user ('pi') for the user 'newpi' we would do the following

```
su newpi
```

The system will ask for the 'newpi' users password and once entered we will have switched to the user 'newpi'

Password:

```
newpi@raspberrypi:/home/pi$
```

An interesting aspect of this method of switching is that the working directory that we find ourselves in is the 'pi' users home directory (/home/pi). This is symptomatic of switching the user, but not to the environment of the new user. We can also switch to the new users environment by using '-' (that's a space followed by a hyphen) after the `su` command.

So the following substitutes the user and switches to the environment of the new user;

```
su - newpi
```

## The `su` command

The `su` command can have a similar function to `sudo` in the sense that it can allow us to switch to using root permissions to run commands while logged in as a different user. However, the substantive difference in this circumstance is that `su` will switch to the new user while `sudo` will simply execute the single command. This can be remedied by using the `-c` option with `su` such that we tell `su` to execute the command that directly follows it on the same line.

The name of the command was initially derived from the words substitute user. Although switch user is an accepted description as well.

Once we have switched users we can revert by typing 'exit' and pressing enter or by pressing 'CTRL-d'

**Test yourself**

1. Is substitute **user** or switch **user** a more accurate description of the derivation of the `su` command?
2. If you were going to run a series of commands as root would it be better to do so using `sudo` or `su`?
3. How do we ensure that our environmental variables are set to the new user when switching user?

## sudo

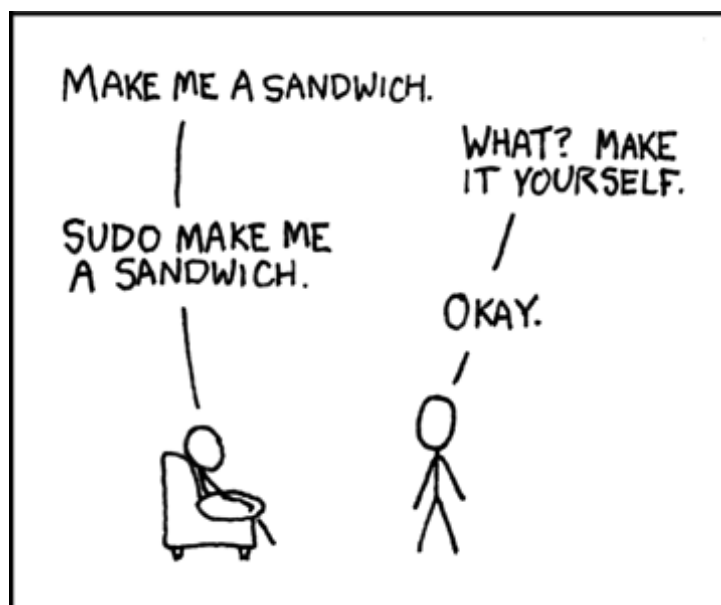
The `sudo` command allows a user to execute a command as the 'superuser' (or as another user). It is a vital tool for system administration and management.

- `sudo [options] [command]` : Execute a command as the superuser

For example, if we want to update and upgrade our software packages, we will need to do so as the super user. All we need to do is prefix the command `apt-get` with `sudo` as follows;

```
sudo apt-get update
sudo apt-get upgrade
```

One of the best illustrations of this is via the excellent cartoon work of the [xkcd comic strip](https://xkcd.com/149/)<sup>25</sup> ([Buy his stuff](http://store.xkcd.com/)<sup>26</sup>, it's awesome!).



Sudo courtesy xkcd

## The `sudo` command

The `sudo` command is shorthand for 'superuser do'.



The `sudo` command allows the user to run programs or give commands that should only be executed with a degree of caution as they could potentially affect the normal operation of the computer. However, a user can only use this command if they have the correct permissions to do so.

<sup>25</sup><https://xkcd.com/149/>

<sup>26</sup><http://store.xkcd.com/>

When we use `sudo` an authorised user is determined by the contents of the file `/etc/sudoers`. As an example of usage we should check out the file `/etc/sudoers`. If we use the `cat` command to list the file like so;

```
cat /etc/sudoers
```

We get the following response;

```
pi@raspberrypi ~ $ cat /etc/sudoers
cat: /etc/sudoers: Permission denied
```

That's correct, the 'pi' user does not have permissions to view the file. Let's confirm that with `ls`;

```
ls /etc/sudoers
```

Which will result in the following;

```
pi@raspberrypi ~ $ ls -l /etc/sudoers
-r--r----- 1 root root 696 May  7 10:39 /etc/sudoers
```

It would appear that only the root user can read the file!

So let's use `sudo` to `cat` the file as follows;

```
sudo cat /etc/sudoers
```

That will result in the following output;

```
pi@raspberrypi ~ $ sudo cat /etc/sudoers
#
# This file MUST be edited with the 'visudo' command as root.
#
# Please consider adding local content in /etc/sudoers.d/ instead of
# directly modifying this file.
#
# See the man page for details on how to write a sudoers file.
#
Defaults                env_reset
```



```

Defaults        mail_badpass
Defaults        secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"

# Host alias specification

# User alias specification

# Cmnd alias specification

# User privilege specification
root    ALL=(ALL:ALL) ALL

# Allow members of group sudo to execute any command
%sudo    ALL=(ALL:ALL) ALL

# See sudoers(5) for more information on "#include" directives:

#includedir /etc/sudoers.d
pi ALL=(ALL) NOPASSWD: ALL

```



DO NOT edit this file with a text editor! Always use the `visudo` command like the instructions say! (Interestingly, Raspbian has the [nano editor](#) (not vi) configured as the default [editor](#).)

There's a lot of information in the file, but there, right at the bottom is the line that determines the privileges for the 'pi' user;

```
pi ALL=(ALL) NOPASSWD: ALL
```

We can break down what each section means;

**pi**

```
pi ALL=(ALL) NOPASSWD: ALL
```

The pi portion is the user that this particular rule will apply to.

**ALL**

```
pi ALL=(ALL) NOPASSWD: ALL
```

The first ALL portion tells us that the rule applies to all hosts.

**ALL**

```
pi ALL=(ALL) NOPASSWD: ALL
```

The second ALL tells us that the user 'pi' can run commands as all users and all groups.

**NOPASSWD**

```
pi ALL=(ALL) NOPASSWD: ALL
```

The NOPASSWD tells us that the user 'pi' won't be asked for their password when executing a command with sudo.

**ALL**

```
pi ALL=(ALL) NOPASSWD: ALL
```

The last ALL tells us that the rules on the line apply to all commands.

Under normal situations the use of sudo would require a user to be authorised and then enter their password. By default the Raspbian operating system has the 'pi' user configured in the /etc/sudoers file to avoid entering the password every time.

If your curious about what privileges (if any) a user has, we can execute sudo with the -l option to list them;

```
sudo -l
```

This will result in output that looks similar to the following;

```
pi@raspberrypi ~ $ sudo -l
Matching Defaults entries for pi on this host:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin
```

User pi may run the following commands on this host:

```
(ALL : ALL) ALL
(ALL) NOPASSWD: ALL
```

**The 'sudoers' file**

As mentioned above, the file that determines permissions for users is /etc/sudoers. **DO NOT EDIT THIS BY HAND.** Use the visudo command to edit. Of course you will be required to run the command using sudo;

```
sudo visudo
```



I'm going to reinforce the point a bit more that starting to configure the `sudoers` file is a task that should be taken very seriously and with full knowledge of the security implications.

If you open up a `sudoers` file to edit it and you see something like the following;

```
bob ALL=(ALL) ALL
john ALL=(ALL) ALL
eve ALL=(ALL) ALL
someguy ALL=(ALL) ALL
```

It would indicate that there are a fair number of people with full administration rights to this server and perhaps this should be reviewed?

### **sudo VS su**

There is a degree of confusion about the roles of the `sudo` command vs the `su` command. While both can be used to gain root privileges, the `su` command actually switches the user to another user, while `sudo` only runs the specified command with different privileges. While there will be a degree of debate about their use, it is widely agreed that for simple on-off elevation, `sudo` is ideal.

### **Test yourself**

1. Write an entry for the `sudoers` file that provides `sudo` privileges to a user for only the `cat` command.
2. Under what circumstances can you edit the `sudoers` file with a standard text editor.

## useradd

The `useradd` command is used by a superuser (typically via `sudo`) to add a user system account. It's a fundamental command in the sense that Linux as a multi user system required a mechanism for adding users. Not that this command will get used every day, but it's important to know to enable us to administer users on the computer.

- `useradd [options] username` : add a user account

The following example is the simplest application of the command;

```
sudo useradd newpi
```

This creates the user, but in doing so it is set in a 'locked' state because it doesn't have a password associated with it. This can be set using the `passwd` command.

## The useradd command

The `useradd` command is not utilised on a regular basis, in fact, the man page for the command recommends using `adduser` which is a friendlier front end to this low level tool. It adds a user to the list of entities able to access the computer system although used without any options, the command leaves the new account locked until a password is set with `passwd`. It can only be used by the root user, so it is often prefixed by the `sudo` command.

## Options

There are several useful options that can be used with `useradd`, but the commonly used ones we will examine are;

- `-m` : Add the user's home directory if it doesn't exist
- `-s` : Specify the user's login shell

To add the home directory for the 'newpi' user and set their login shell to bash we would use the following command;

```
sudo useradd -m -s /bin/bash newpi
```

## Test yourself

1. What is the more user friendly alternative to `useradd`?

## usermod

The `usermod` command is used by a superuser (typically via [sudo](#)) to change a user's system account settings. It's not a command that will get used every day, but it's important to know that it's there to enable us to administer user details on the computer.

- `usermod [options] username` : Modify a users account configuration

The following example is one used in the early stages of setting up the Raspberry Pi where we want to make our user 'pi' a member of the group 'www-data' which will allow (in conjunction with other steps) the user to edit the files in a web server;

```
sudo usermod -a -G www-data pi
```

The `-a` option adds the user to an additional 'supplementary' group (`-a` must only be used in a situation where it is followed by the use of a `-G` option). The `-G` option lists the supplementary group to which a user will be added. In this case the group is `www-data`. Finally the user which is getting added to the supplementary group is `pi`.

## The `usermod` command

The `usermod` command is not utilised on a regular basis, but when required to administer a users administrative details it is useful to know what it is able to do and how to configure the command. It modifies a users settings (hence `usermod` = **user modify**) and can be used for such tasks as changing a users home directory, password expiry date, their default shell and the groups that the user belongs to. It can only be used by the root user, so it is often prefixed by the [sudo](#) command.

When `usermod` changes a users settings it does so utilising the following files;

- `/etc/group` : Group account information.
- `/etc/gshadow` : Secure group account information.
- `/etc/login.defs` : Shadow password suite configuration.
- `/etc/passwd` : User account information.
- `/etc/shadow` : Secure user account information.

## Options

There are about 15 options that can be used with `usermod`, but the commonly used ones we will examine are;

- `-d` : Change the user's home directory
- `-g` : Change the user's primary group

- `-a -G group` : Add the user to a supplemental group
- `-s` : Change the users default shell
- `-L` : Lock the users account by disabling the password
- `-U` : Unlock the users account by re-enabling the previous password

To change the home directory for the 'pi' user to the directory /home/newpi we would use the following command;

```
sudo usermod -d /home/newpi pi
```

To make 'pigroup' the primary group for the user 'pi';

```
sudo usermod -g pigroup pi
```



In order for this to be successful the group must exist. Also, any file from the user's home directory which was owned by the previous primary group of the user will be owned by this new group.

Different users will be comfortable in different shells. To change the users default shell we can use the `-s` option. The following example changes the default shell for the 'pi' user to the 'Korn' shell;

```
sudo usermod -s /bin/ksh pi
```

We can lock a users account using the `-L` option which modifies the /etc/shadow file by putting an exclamation mark in front of the encrypted password. The following example does this for the 'pi' user;

```
sudo usermod -L pi
```

Then we will want to unlock the locked user account with the `-U` option as follows;

```
sudo usermod -U pi
```

**Test yourself**

1. What might we need to do before changing the primary group of a user to a new group?
2. How might we manually unlock a users account without using the `usermod` command?

## who

The `who` command can tell us which users are currently logged into a system as well as when the system was last rebooted and a range of other system information details. This is a useful system administration tool and one that is especially relevant in a larger multi-user environment.

- `who [options]` : report the users logged onto the system.

In its simplest form the `who` command can tell us who is logged onto the system as follows;

```
who
```

While logged in as the user 'pi' the output will be;

```
pi      pts/0      2016-02-14 05:39 (10.1.1.33)
pi      pts/1      2016-02-14 06:09 (10.1.1.33)
newpi   pts/2      2016-02-14 06:10 (10.1.1.222)
```

Looking at the first line, this shows us the name of the user logged in (pi) the terminal that has been used to log onto the system (pts/0, which means **p**suedo **t**erminal **s**lave number 0), the date/time of the connection (2016-02-14 05:39) and the IP address that the connection was made from (10.1.1.33). We can see that as well as being connected twice from 10.1.1.33 as the user 'pi', there is another user 'newpi' connected from 10.1.1.222.

## The `who` command

The `who` command is essentially a system administration tool for determining a range of information about our system. While the name is obviously a reflection of the need to determine **who** is logged onto a system, there are a range of other details that can be determined using the command. The additional information is accessed via the options as follows;

- `-b` : time of last system **boot**
- `-d` : all the **dead** processes
- `--login` : the system **login** processes
- `-p` : the active **process** generated by **init**
- `-r` : the **runlevel** of the system
- `-t` : the **time** the system clock last changed
- `-T` : the users message status (a +, - or ?)
- `-u` : **users** logged in.
- `-a` : all of the above
- `-H` : print column **headers**

There is a remendous amount of detail in the options above and a great deal of it will not be relevant to a new user, but it's useful to present it here so that if we are running the `-a` option we know what we are getting ourselves in for.



```
who -aH
```

NAME	LINE	TIME	IDLE	PID	COMMENT	EXIT
	system boot	2016-02-13 22:27				
	run-level 5	2016-02-13 22:27				
LOGIN	tty1	2016-02-13 22:27		1119	id=tty1	
LOGIN	ttyAMA0	2016-02-13 22:27		1122	id=AMA0	
pi	+ pts/0	2016-02-14 05:39	00:22	1398	(10.1.1.33)	
pi	+ pts/1	2016-02-14 06:09	.	1479	(10.1.1.33)	
newpi	+ pts/2	2016-02-14 06:10	00:23	1505	(10.1.1.222)	

In the output we can see the headers;

NAME	LINE	TIME	IDLE	PID	COMMENT	EXIT
------	------	------	------	-----	---------	------

When the system was last booted (although anyone with a Raspberry Pi will know that I fudged the time here);

```
system boot 2016-02-13 22:27
```

The runlevel of the system;

```
run-level 5 2016-02-13 22:27
```



A run level is an initialisation state for the system that defines what system services are operating. They are designated with a number and can vary as follows;

- 0 Halt the system.
- 1 Single-user mode (for special administration).
- 2 Local Multiuser with Networking but without network service (like NFS)
- 3 Full Multiuser with Networking
- 4 Not Used
- 5 Full Multiuser with Networking and X Windows(GUI)
- 6 Reboot.

The + symbols after the 'pi' and 'newpi' users indicate that their message status;

pi	+ pts/0	2016-02-14 05:39 00:22	1398 (10.1.1.33)
pi	+ pts/1	2016-02-14 06:09 .	1479 (10.1.1.33)
newpi	+ pts/2	2016-02-14 06:10 00:23	1505 (10.1.1.222)

The different message statuses are;

- + allowing 'write' messages
- - disallowing 'write' messages
- ? cannot find terminal device

The message service is one where a user can send a message directly into another users terminal. Although we don't cover it in this book more information can be found with the command `man mesg`.

## Test yourself

1. Why did I mention that I fudged the boot time for the example description above?
2. Type the command `who am i` and determine the difference between that and the command `whoami`.

## whoami

The `whoami` command can tell us who is the current user. Depending on what shell we are using we may not be presented with the username at the command prompt. Believe it or not it is a relatively common occurrence to lose track of which user a is the operative user while working (especially in more complex environments). If this is the case we can find out who we are with a simple command.

- `whoami [options]` : report the current user.

The `whoami` command is a very simple one and is almost exclusively run without options as follows;

```
whoami
```

While logged in as the user 'pi' the output will be;

```
pi
```

## The `whoami` command

Working in complex multi-user systems carries the potential for confusion when working at the command line and / or with multiple terminal windows open of losing track of which user is the current effective user logged in. The ramifications of executing a command as the wrong user can be catastrophic if working as the 'root' user, so it behoves us to make sure that we know 'who' we are at the command line.

In many shells the current user is printed at the command prompt, so it's fairly obvious. However, some shells such as 'csh' do not by default present the current user name, so we need to be able to determine this with a command. The `whoami` command is (fairly obviously) a compression of the phrase "who am i?" and its output is fairly straight forward.

In addition to the simple example above, as a test we can run the command with `sudo` to simulate operating as the 'root' user as follows;

```
sudo whoami
```

The result should be the unsurprising;

```
root
```

... irrespective of which user we ran the command as.

The `whoami` command is useful in relation to the current shell, but to determine more about other users logged into the system we would use the `who` command.

**Test yourself**

1. What is the difference between a shell, a console and a terminal (trick question that will require research)?

# Command Cheat Sheet

## File Administration

- `cd` [options] *directory* : Used to **change** the current **directory**
- `chgrp` [options] *group files* : **Change group** ownership of one or more files & directories
- `chmod` [options] *mode files* : **Change** access **permissions** of one or more files & directories
- `chown` [options] *newowner files* : **Change** the **ownership** of one or more files & directories
- `cp` [options] *source destination* : **Copy** files and directories
- `find` [start-point] [search-criteria] [search-term] : **find** files
- `gzip` [options] *filename* : **compress** or **expand** files
- `ln` [options] *originalfile linkfile* : Create **links** between files or directories
- `ln` [options] *originalfile* : Create a **link** to the original file in the current directory
- `ls` [options] [arguments] : **List** the files in a particular directory
- `mkdir` [options] *directory* : **Make** a **directory**
- `mv` [options] *source destination* : **Move** and/or **rename** files and directories
- `pwd` [option] : prints the **current working directory**
- `rm` [options] *file* : **Delete** files or directories
- `rmdir` [options] *directory* : **Remove** empty **directories**
- `tar` [options] *archivename* [file(s)] : **archive** or **extract** files

## Accessing File Contents

- `cat` [options] [filenames] [-] [filenames] : **Display**, combine or create new **files**.
- `cut` [options] *file* : **Cut out** sections of each line of a file or files
- `diff` [options] *from-file to-file* : **Display** the **differences** between two files
- `grep` [options] *pattern* [*file*] : Search text and match patterns
- `head` [options] *filename(s)* : List the first part of a specified file
- `less` [options] *filename* : **Display** text **files**
- `more` [options] *filename* : **Display** a text **file**
- `tail` [options] *filename(s)* : List the last part of a specified file

## File Systems

- `fdisk` [options] [*device*] : **manipulate** **partition** tables.
- `mkfs` [options] *device* : **format** a **partition** with a file system
- `mount` [options] *type device directory* : **Mount** storage onto the file system
- `umount` [options] *device* and/or *directory* : **Unmount** storage from the file system

## System Information

- `date` [options] *+format* : display or set the date / time
- `df` [options] *file* : display the amount of **free space** on filesystems.
- `du` [options] *file* : display the amount **space used** in files and directories
- `free` [options] : displays information about **free and used memory** on the system

## Processes

- `crontab` [-u user] [-l | -r | -e] : **Schedule** a **task** to run at a particular time or interval
- `kill` [options] *PID(s)* : sends a signal to **terminate** a process or processes **by PID**
- `killall` [options] *command name* : sends a signal to **terminate** a processes **by name**
- `ps` [options] : displays a snapshot of the **current processes**
- `top` [options] : display **system** and process **information** in real time

## Network

- `curl` [options] [URL] : **download or upload files** to remote servers.
- `host` [options] *server* : checks a computers IP address from its **host-name** or the reverse
- `ifconfig` [arguments] *interface* [options] : **Configure** a **network** interface
- `ifconfig` [arguments] [interface] : **View network** interface details
- `ip` [options] *object sub-command* [parameters] : **network configuration** tool
- `netstat` [options] : network **connection monitoring** tool
- `ping` [options] *remote server* : **checks the network** connection to a remote server.
- `scp` [options] *user1@sourcehost:directory/filename user2@destinationhost:directory/filename* : securely copy files between computers
- `sftp` [options] [*user@*]*host-name[:directory]* : interactively copy files between computers securely
- `ssh` [options] [login name] *server* : **securely access** a remote computer.
- `traceroute` [options] *host-name* : trace the route packets take to network host
- `wget` [options] [URL] : **download or upload files** from the web non-interactively.

## Miscellaneous

- `apt-get install` *\*package-name\** : **Install** a package
- `apt-get remove` *\*package-name\** : **Un-install** a package
- `apt-get update` : **Update** the **database** of available packages
- `apt-get upgrade` : **Upgrade** the **packages** on the system
- `clear` : **clears** the terminal **screen**.
- `echo` [options] [string(s)] : **display text** on the screen.
- `groupadd` [options] *group* : **add a group** account
- `man` [options] *command* : **display the reference manual** for a command.

- `modprobe` [options] *modulename* : **Load** or remove a Linux kernel **module**
- `passwd` [options] *username* : **change** the user **password**
- `shutdown` [options] [time] [message] : **shut down or restart** a system.
- `su` [options] [*username*] : **switch** to another **user**.
- `sudo` [options] *command* : Execute a command as the **superuser**
- `usermod` [options] *username* : **Modify** a **users** account configuration
- `useradd` [options] *username* : **Add** a **user** account
- `who` [options] : report the **users** logged **onto the system**.
- `whoami` [options] : report the **current user**.

# Directory Structure Cheat Sheet

- / : The 'root' directory which contains all other files and directories
- /bin : Common commands / programs, shared by all users
- /boot : Contains the files needed to successfully start the computer during the boot process
- /dev : Holds [device files](#) that represent physical and 'logical' devices
- /etc : Contains configuration files that control the operation of programs
- /etc/cron.d : One of the directories that allow programs to be run on a [regular schedule](#)
- /etc/rc?.d : Directories containing files that control the mode of operation of a computer
- /home : A directory that holds subdirectories for each user to store user specific files
- /lib : Contains shared library files and kernel modules
- /lost+found : Will hold recoverable data in the event of an an improper shut-down
- /media : Used to temporarily mount removable devices
- /mnt : A mount point for filesystems or temporary mount point for system administrators
- /opt : Contains third party or additional software that is not part of the default installation
- /proc : Holds files that contain information about running processes and system resources
- /root : The home directory of the System Administrator, or the 'root' user
- /sbin : Contains binary executables / commands used by the system administrator
- /srv : Provides a consistent location for storing data for specific services
- /tmp : A temporary location for storing files or data
- /usr : Is the directory where user programs and data are stored and shared
- /usr/bin : Contains binary executable files for users
- /usr/lib : Holds shared library files to support executables in /usr/bin and /usr/sbin
- /usr/local : Contains users programs that are installed locally from source code
- /usr/sbin : The directory for non-essential system administration binary executables
- /var : Holds variable data files which are expected to grow under normal circumstances
- /var/lib : Contains dynamic state information that programs modify while they run
- /var/log : Stores log files from a range of programs and services
- /var/spool : Contains files that are held (spooled) for later processing
- /var/tmp : A temporary store for data that needs to be held between reboots (unlike /tmp)



# Regular Expression Cheat Sheet

The following are some of the most commonly used metacharacters and a very short description of their effect;

- [ ] Match anything inside the square brackets for ONE character
- ^ (circumflex or caret) Matches only at the *beginning* of the target string (when not used inside square brackets (where it has a different meaning))
- \$ Matches only at the *end* of the target string
- (period or full-stop) Matches any single character
- ? Matches when the preceding character occurs 0 or 1 times only
- \* Matches when the preceding character occurs 0 or more times
- + Matches when the preceding character occurs 1 or more times
- ( ) Can be used to group parts of our search expression together
- | (vertical bar or pipe) Allows us to find the left hand *or* right values

# 'find' Cheat Sheet

The process of searching with 'find' is about combining three things;

1. The start point for the search
2. The type of criteria or property that we are going to be evaluating on our search
3. The specific value of the criteria we want to search against.

Numeric arguments below can be specified as;

- $+n$  : for greater than  $n$ ,
- $-n$  : for less than  $n$ ,
- $n$  : for exactly  $n$ .

We can also employ [wildcards](#) to assist in matching names.

- `-name pattern` : the file name which matches the pattern *pattern*
- `-iname pattern` : Like `-name`, but the match is case insensitive
- `-mmin  $n$`  : the file's data was last modified  $n$  minutes ago
- `-mtime  $n$`  : the file's data was last modified  $n * 24$  hours ago
- `-newer file` : the file was modified more recently than *file*
- `-amin  $n$`  : the file was last accessed  $n$  minutes ago.
- `-atime  $n$`  : the file was last accessed  $n * 24$  hours ago
- `-user uname` : the file is owned by user *uname*
- `-group gname` : the file belongs to group *gname*
- `-executable` : matches files which are executable and directories which are searchable
- `-type  $c$`  : the file is of type  $c$ :
  - `b` block (buffered) special
  - `c` character (unbuffered) special
  - `d` directory
  - `p` named pipe (FIFO)
  - `f` regular file
  - `l` symbolic link
  - `s` socket
- `-size  $n$ [cwbkMG]` : File uses  $n$  units of space (rounding up). The following suffixes can be used:
  - `b` for 512-byte blocks (this is the default if no suffix is used)
  - `c` for bytes
  - `w` for two-byte words
  - `k` for Kilobytes (units of 1024 bytes)
  - `M` for Megabytes (units of 1048576 bytes)

- G for Gigabytes (units of 1073741824 bytes)
- -perm *mode*: The files have a specific set of permissions with *mode* different depending on how we want to access them
  - mode File's permission bits are **exactly** *mode*
  - -mode **All** of the permission bits *mode* are set for the file.
  - /mode **Any** of the permission bits *mode* are set for the file.