The Linux Command Line

Chapter 6

The operating system on your Raspberry Pi is a version of Linux. In all probability, it looks a bit like Windows, or – more likely – like the Mac OS. It has a graphical user interface (GUI) that allows you to interact with your folders and files by double-clicking, right- or left-clicking, or dragging and dropping. If you want to open a program, you look for it on a menu called something like "Applications".

That's all very well, but there is another way to interact with Linux: using the **command line interface** (**CLI**). With the CLI there are no images; nothing to click on. To get your computer to do something, you must type a properly constructed text command. Often, you'll only know that your command has been successfully executed because your computer won't respond. If it does respond, something has gone wrong and it's giving you an error message.

That sounds like hard work. Do we really want to bother with it? Yes, we do! For a start, by default, some versions of Linux boot directly into the CLI. To open the GUI, with all its lovely windows and menus, you then need to type the command "**startx**" and press Return. But that's by no means the only reason to get to grips with the CLI.

With the command line you can do things that you can't do using the GUI and you can do things easily that are difficult in the GUI. Compared with the graphical interface, the command line hardly uses any processing or graphics power, so it's great if you have heavy work for your computer to do. And, if you're connecting remotely to a Raspberry Pi that doesn't have its own monitor, then you'll have to use the CLI.

If you're new to Linux, you may not be familiar with the command line in general and with Linux commands in particular (they're sometimes similar to their Windows equivalents, but often they're completely different). This chapter is a quick introduction to the magic of the command line. It contains everything you need to get started.

Notes:

Lesson 6.1: Commands are just programs

Notes:

When you type into the terminal you are running programs. Most of the commands listed here run programs that give you the ability to command the system to do something. When you add programs to your Raspberry Pi, you will be able to run more commands.

If you enter a command and it doesn't work, it may well mean that the program isn't installed yet. For instance, if you try to create a new text file called "My Linux Commands" using the Nano text editor, you would type:

```
sudo nano "My Linux Commands"
```

But it might be that, instead of opening the new text file, your system returns the error message:

```
bash: nano: command not found
```

This just means that you have not installed Nano yet. However, most of the commands described on this page are the kind of "housekeeping" commands that come as default, so you shouldn't have this problem. Each program has an original author who is acknowledged at the bottom of the main page.

Don't worry too much about the actual commands used there – I will explain everything as we go along.

The Bash shell

To use the CLI, you need to know commands. The default command set is contained in the "**shell**" you are using. There are lots of shells out there, but the one on this Raspberry Pi is called "**Bash**". Bash is the default command line shell on most versions of Linux, as well as Mac OS, so it's well worth learning.

This chapter covers many of the most common and useful commands. If you can't find the command you need, try looking online. There are lots of good guides on the internet for using the Linux command line.

If you know the command you want to use, but don't know exactly how to use it – for instance, you don't know exactly the right "**syntax**" – you can use the "**man**" (manual) command in Linux. At the command line you could type:

```
$ man <command>
```

man

Displays information about the target, sourced from online reference manuals.

Tip...

The "syntax" of a command means the way it should be written to make it work. That means the order of the words, as well as any important punctuation that is also necessary.

Or, you could use the "**info**" command:

```
$ info <command>
```

info

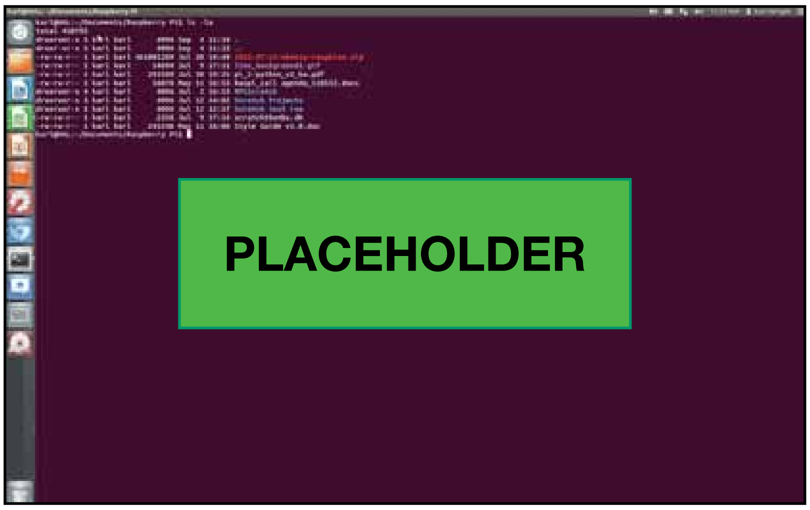
Displays online documentation about the target.

In the examples above, substitute <command> with the command you want to find out more about. A word of warning: the guidance given by "**man**" is sometimes a bit formal and very, very detailed. You could almost say, "If you think you know a command then go to the command's man page in order to find out that you don't really."

To find out more about Bash, take a look at its Wikipedia entry:
http://en.wikipedia.org/wiki/Bash_%28Unix_shell%29

Typing "man apt-get" into the command line returned all this information on the "apt-get" command and how to use it.

"Man" is short for manual.



Notes:

Lesson 6.2: Command syntax and file structure

Notes:

Commands take the form:

```
<Command> | <Switches> | <Parameters> | <Target>
```

In guides, such as this one, the brackets “<” and “>” are often used to indicate the place a command would take in a string of text being typed into the CLI. The horizontal line “|” is used to denote “or”. Confused? Okay, let’s look at the example above. We would read that as saying that commands can be used by themselves OR they can be used with:

- **Switches:** single letters, preceded by a hyphen, that adjust what the command does
- **Parameters:** things that the command needs to know in order to work
- **A target:** the thing (such as a file) that the command will be applied to

Let’s look at an example. We’ll start with the “**ls**” command, which you can use to see a list of a folder’s contents.

$$\underbrace{\texttt{ls}}_{\text{Command}}\ \underbrace{\texttt{-l}}_{\text{Switch}}\ \underbrace{\texttt{/home/brian}}_{\text{Target}}$$

This command tells the command line to list, in long format, the contents of the directory “/home/brian”. The *command* is “ls”; the *switch* “-l” tells Linux that you want the list in long format; and the *target* of this command is the directory “/home/brian”. In this example, there are no *parameters*, so that part of the command is just skipped.

> **ls**
> ***Lists the contents of directories.***

For a slightly more detailed example, let’s look at the “**mount**” command, which tells Linux to incorporate a new file system (such as a CD or DVD) into its own file system structure so that you can browse it.

$$\underbrace{\texttt{mount}}_{\text{Command}}\ \underbrace{\texttt{-F smbfs}}_{\text{Switch}}\ \underbrace{\texttt{//workgroup;fred:foo@192.168.1.99/homes}}_{\text{Parameters}}\ \underbrace{\texttt{/mnt/net}}_{\text{Target}}$$

This command tells the operating system to use the username “fred” and password “foo” (parameters) to make the shared drive called “homes” on the Windows server at 192.168.1.99 (parameter) appear in the directory tree at the point “/mnt/net” (target) using the Server Message Block Filing System (the -F switch).

> **mount**
> ***Makes a file structure available at a new location.***

Navigating the file system using "cd"

Notes:

The file system in Linux is hierarchical with nested directories (often called "**folders**") in a "**tree**". The top of the directory structure is denoted by the symbol "/", and directories underneath "/" are referred to using "**paths**", just like URLs in a web browser.

To go to a particular place in the directory structure, you use the command "**cd**", which stands for "change directory", followed by its location in the tree. For instance, the following command will take you to Brian's "Documents" folder in his home directory.

```
cd /home/brian/Documents
```

If you want to move up the directory tree, use the command:

```
cd ..
```

For example, if you're working in the folder "/home/brian/Documents/Project", the command "cd .." would take you back to "/home/brian/Documents". Make sure you leave a space between the letters "cd" and the two dots, otherwise the command won't work.

To find out more about the "cd" command see its Wikipedia entry:
http://en.wikipedia.org/wiki/Cd_%28command%29

cd

Changes the current working directory.

Tip...

The names of files and directories (folders) are case-sensitive in Linux, so the file "my_stuff" is different from the file "My_stuff".

Listing the files and folders in a particular place

Notes:

The “ls” command lists all contents of directory you’re working in. There are a range of switches you can use with “ls” to make it display exactly the files you’re interested in and all the useful information about those files. We’ve listed a few of those switches here:

- “**ls**”, the command with no switches, lists all the file in the current directory.
- “**ls -l**” lists the files and displays the long version of the information about each file or directory. Output may be colour-coded depending on the terminal preferences that you have set.
- “**ls -R**” asks for a “recursive” list – that is, a list including the contents of sub-directories as well as this directory.
- “**ls -A**” forces the system to show “hidden” files. Hidden files have names that start with a dot, which won’t usually be visible when doing a normal directory list.

Let’s look at two examples. In the first, we use the plain “ls” command with no switches.

```
$ ls

An_Gott_und_meine_Mutter.mid Domestic Programming Test.mid
An_Gott_und_meine_Mutter.mscz Engineering Quantum Physics Tutoring
appliances FluidR3 _ GM.ins School Windaz
```

In the second, we use “ls -l”. This displays the files in long format, telling you, among other things, the size, owner and security setting on each file.

```
$ ls -l

total 336
-rw-rw-r--. 1 brian brian 2429 Apr 2 20:27 An_Gott_und_meine_Mutter.mid
-rw-rw-r--. 1 brian brian 4085 Apr 2 19:52 An_Gott_und_meine_Mutter.mscz
drwxrwxr-x. 4 brian brian 4096 Apr 2 20:38 appliances
-rw-rw-r--. 1 brian brian 10919 Apr 2 19:52 brotplot.odt
```

Don’t worry, we’ll go into the details of this information later.

To find out more about the “ls” command, take a look at its Wikipedia Entry:
http://en.wikipedia.org/wiki/Ls

Change ownership of directories and files

Notes:

In the long List above, the files are owned by Brian and are also in the Brian group. What if we wanted to change the ownership of some of those files? Then we would use the "**chown**" command.

We'll use the file "*brotplot.odt*" as an example. Let's say we want to make Fred the file's owner. We would use the command:

```
$ chown fred brotplot.odt
```

If we run the command "ls -l" again, we will now see the result:

```
-rw-rw-r--. 1 fred brian 10919 Apr 2 19:52 brotplot.odt
```

The file is still in group Brian but is now owned by Fred. But what if we want to make the user "Foo" the owner of the file and also move the file to the "foo" group. We'd use the command:

```
$ chown foo:foo brotplot.odt
```

If we use "ls -l" again to see the long-format name of our file, we will see:

```
-rw-rw-r--. 1 foo foo 10919 Apr 2 19:52 brotplot.odt
```

To find out more about "chown" command see its Wikipedia entry:
http://en.wikipedia.org/wiki/Chown

chown
Changes the ownership of one of more files.

While we're at it, take a look at "**chgrp**" to see how it compares:
http://en.wikipedia.org/wiki/Chgrp

chgrp
Changes the group of one of more files.

Change access to directories and files

Notes:

When you use the "-l" switch to see the long list of information a directory's contents, the results you see are all preceded by a string of letters and dashes. For instance:

```
-rw-rw-r--. 1 fred students 10919 Apr 2 19:52 brotplot.odt
```

The first character in the sequence, in this case a dash, tells us what the object is. The most commonly-used characters are:

d	Indicates a directory
l	Indicates a link
-	Indicates a file

Our example above starts with a dash, so we know it's a file. In this case, it's actually a word processor file.

The nine characters that follow, describe access rights for the owner (user), group and the world (everyone else who may be able to access the file), in that order, each with three characters. The characters used here are:

r	Read access
w	Write access
x	Permission to execute the program
-	No access of this type

So, if all three groups had all three types of access, we would see "rwxrwxrwx". However, whenever someone doesn't have an access type, its letter is replaced with a dash.

If we again look at our example above, we can see that:

- Brian has read and write access to the file.
- Members of the group called "students" also have read and write access to the file.
- Everyone else (the "world") has only read access to the file.

The "x" flag is not listed at all. Remember, x stands for the ability to execute a file. If the x flag is not set then the file will not run as a program.

But what if we want to change the file's permissions? There are various ways to use the "**chmod**" command to do this, but probably the easiest is to use these groups:

u	for user/owner
g	for group
o	for other (everyone else)
a	for all three

Notes:

Now, let’s try out the “chmod” command:

```
chmod o+w brotplot.odt
```

This means add write access for the “other” group to “*brotplot.odt*”. Add is indicated by the “plus” sign.

```
chmod a-r brotplot.odt
```

This means remove read access for the all three (user, group and other) from “*brotplot.odt*”. Remove is indicated by the “minus” sign.

```
chmod u+x brotplot.odt
```

This means add execute access for the user/owner to “*brotplot.odt*”.

To find out more about the “chmod” command see its Wikipedia entry:
http://en.wikipedia.org/wiki/Chmod

chmod
Changes the access mode (permissions) of one or more files.

Lesson 6.3: The superuser

Notes:

The "**sudo**" command introduces the "superuser" or "root user". The term "root" is the name for the main administrator in a "Unix-like" system, such as Linux. There are many commands that only the root user can run.

Depending on the version of Linux you are using, you will either have to log on as the root user or prefix your command with "sudo". The default Debian distribution of Linux, for instance, has no root password set. So you will have to use the "sudo" command.

By using "sudo", you're saying, "Do the following command as the root user." When you do this, you will be asked for your password and, if you have the system permissions of a root user (commonly called "being in the admin group"), then the command will be run. If you don't have root permissions, you will get an error message.

Let's look at the example of adding a new user to your system, something that only the root user can do. On Debian, the command would look like this:

```
sudo adduser brian
```

This will start a script allowing you to set up a new user called "brian". If you ever try to do something in the command line and get the error message, "Only root can ..." then try the sudo command – it may fix your problem.

To find out more about "sudo" command, see its Wikipedia entry:
http://en.wikipedia.org/wiki/Sudo

sudo
If you have the appropriate permissions, execute the following command as the superuser.

Becoming the admin user

There is another way of doing things as root user and that is by using the "**su**" command, which stands for "substitute user" (or superuser). Invoking the "su" command means "become the root user".

As before, you'll be asked to enter your password to make sure that you have superuser rights. Once you have successfully authenticated with your password, all commands run as the superuser.

In some systems, however, you won't be able to use the "su" command because, by default, the root user isn't enabled. In this case, you will have to enable the root user, using this command:

```
passwd root
```

You will be asked to enter and then confirm a new root password. Assuming you can manage to enter the same password twice, you now have a root user and you can issue the command:

```
su -
```

When you enter passwords in Unix-like systems, the prompt remains blank: no blobs or asterisks stand in for the characters in your password. Don't be put off; the password is being entered nevertheless.

Finally, a word of warning: if you habitually do everything as the root user, eventually you will do something both educational and disastrous. For example, you could invoke:

```
rm -r -f /*
```

This will delete all the files in the whole system.

Or you might absent-mindedly type:

```
$ rm -r -f ./*
```

That would just delete the files in the current directory. And you do not usually get warnings as root user (other than this one). The first thing you'll know about your error is when all your files have suddenly vanished.

Only use root privileges when you really need them.

To find out more about the "su" command. see its Wikipedia entry:
http://en.wikipedia.org/wiki/Su_%28Unix%29

su
Become a superuser.

passwd
Create or change a password associated with the identified user.

Notes:

Tip...

Don't ever use these commands when operating as a superuser!

Lesson 6.4: Creating and destroying files and directories

Notes:

Finally, let's look at creating and destroying file and directories, as well as taking a closer look at the mount command.

Create a new empty file

To create a new empty file, go to where you want the file to be and type:

```
$ touch <filename>
```

The "**touch**" command actually updates a file's "last accessed" time to the current time and date, but if such a file doesn't already exist then it will create a new file of that name, with a file size of 0.

To find out more about the "touch" command, see its Wikipedia entry:
http://en.wikipedia.org/wiki/Touch_(Unix)

touch
Update the named file's access time and modification time (and dates) to the current time and date. If a file doesn't exist then a new file will be created with a file size of 0.

Create a new empty directory

To create a new directory, go to where you want the directory to be and type:

```
$ mkdir <directory name>
```

To find out more about "**mkdir**" command, see its Wikipedia entry:
http://en.wikipedia.org/wiki/Mkdir

mkdir
Create one or more directories in the current location.

Remove a file

To delete a file, go that file's location and type:

```
$ rm <filename>
```

To find out more about the "**rm**" command, see its Wikipedia entry:
http://en.wikipedia.org/wiki/Rm_%28Unix%29

rm
Remove (delete) one or more files in the current location.

Remove a directory

Notes:

To delete a directory, go its location and type:

```
$ rm -r -f <directory name>
```

This uses the "**rm**" command we used above, but the extra switches tell it to remove the directory, all its contents and also any sub-directories and their contents.

The "-r" means "recursive", which – in the case of a directory – removes the entire directory and all its contents, including sub-directories. Do be very sure that you mean this before using it – there's no "undo" option!

The "-f" switch indicates that the action is "forced" – that is, the program will remove write-protected files without prompting. This is also dangerous, so be very careful when using this as well.

A safer, but less powerful, option is to use the "**rmdir**" command:

```
$ rmdir <directory name>
```

This will also delete directories, but only if they're empty.

rmdir

Remove (delete) one or more directories in the current location, provided they are empty.

Connect to a device or filing system

The "mount" command allows you to connect a Unix system to external devices. There is no "C" drive, as in Windows. What happens in Linux is that a device is "mounted" somewhere in the filing system. When you navigate to that place, the items offered by the device will appear at that point.

This is a complex command. The switches, parameters and target of the mount command will vary according to the protocol of the system being mounted. Some things will "auto-mount". This is why, when you plug an SD card into a modern Linux system, the filing system will automatically pick it up.

Manual mounting requires a "mount point". That means a directory that will be filled with the mounted device when it is mounted. Often, this is in the directory "/mnt/" somewhere. Generally, before mounting new media, you must first ensure that there is a mount point. If there isn't, then you must create a directory at the point needed: for instance...

```
$ mkdir /mnt/netfolder
```

Notes:

You must also make sure that you use the necessary switches, parameters and directories. For instance, from the previous exercise:

```
mount -F smbfs //workgroup;fred:foo@192.168.1.99/homes /mnt/net
```

This, as we learned earlier, tells the system to mount the shared drive called “homes” on the Windows server at 192.168.1.99 , in the directory tree at the point “/mnt/net”, using the Server Message Block Filing System (the -F switch).

To find out more about the “mount” command, see its Wikipedia entry:
http://en.wikipedia.org/wiki/Mount_%28Unix%29

Further learning

The commands presented here are just a small selection of all the commands available on your system. You can find a comprehensive list of Linux commands on the O’Reilly website:
http://www.oreillynet.com/linux/cmd

Lesson 6.5: Remote access to the Raspberry Pi

Notes:

OpenSSH is an application that allows you to securely access Linux systems remotely over the network. You can use OpenSSH simply for secure file sharing. But it also allows you to log on to a system and control it over the network, even using the GUI, just as if you were sat in front of it.

The default installation of Linux on your Raspberry Pi should have "**SSH daemon**" running. This means that your Raspberry Pi is listening on port 22 for a remote computer asking to make a connection to it. In your case, this will probably mean your normal desktop or laptop computer.

Running a remote CLI

To connect to your Raspberry Pi, you will need an SSH client program. Linux and the Mac OS already have these installed. For Windows, you can download PuTTY (***http://www.chiark.greenend.org.uk/~sgtatham/putty***). Refer to the manual of your chosen SSH client program for more information on how to install and use it.

In order to make a successful connection, you must have port 22 open on both your remote host and the Raspberry PI. You will also need to set up a suitable user on the RPi (as we did earlier in this chapter).

If you are using a Linux or Mac client, simply enter this at the command line:

```
ssh <IP address of RPi> -l <username on RPi>
```

<IP address of RPi> should be replaced with the IP address of your Raspberry Pi, and <username on RPi> should be replaced with the username of your Raspberry Pi, as you set up previously. The character between them is a lower-case "L".

You should see output similar to this:

```
[brian@fc16toshiba ~]$ ssh -X  192.168.1.104 -l brian
The authenticity of host '192.168.1.104 (192.168.1.104)' can't be established.
RSA key fingerprint is 26:a4:a1:ab:c2:ff:50:99:d7:e1:49:6e:f2:90:fb:90.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.104' (RSA) to the list of known hosts.
brian@192.168.1.104's password:
Linux raspberrypi 3.1.9+ #9 Mon Apr 9 20:50:36 BST 2012 armv6l
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.

brian@raspberrypi:~$
```

At this point you are up and running. Everything you type is actually happening on your Raspberry Pi.

Running a remote GUI

Notes:

To remotely connect to your Raspberry Pi and use its GUI you will need to install an "**X server**" on your machine. Linux has an X server built in. For Windows, try Xming (***http://www.straightrunning.com/XmingNotes***) or Cygwin (***http://www.cygwin.com***) – you can download both for free. For the Mac OS, go to "Shell > New Remote Connection" and choose "Secure Shell (ssh)".

The following command uses the X switch to tell SSH to send the X commands to the X server on your host. Enter the following:

```
ssh -X <IP address of RPi> -l <username on RPi>
```

What is an X server?

A Linux computer (or other Unix-like system) running a GUI is almost certainly running "X" and also an X server. The X sends commands to the X Server about what kind of things to put on the screen, and the X server does it.

This means that the GUI and the X server are separate. This also means that you can run a program on the Raspberry Pi and have the graphical output appear on the screen of the X server somewhere else on the network. This removes a large amount of processor demand from the Raspberry Pi. And it also means that you can just plug your Raspberry Pi into your network, rather than giving it its own monitor, mouse and keyboard.

Open a remote browser

Notes:

Let's see what happens when I try to run a web browser remotely. The web browser on the Raspberry Pi is called **Midori**, so this is what we need to enter from the prompt after we connect remotely:

```
brian@raspberrypi:~$ midori &
[1] 5773
```

We need to include the "&" symbol after the program name because this tells the server to launch it as a separate process, which means we get our command line back.

We get the response "[1] 5773", which is the process number of the Midori program now running.

Sometimes you will get GTK errors, which report library shortcomings on host (the RPi) or server (your X server) but the process is pretty robust.

Midori running in Remote Gnome 3 UI using the Fedora 16 X Server.