**How to get to grips with your Raspberry Pi's command line interface**

**TUTORIAL** The text commands to get things done on your Pi

**By**[**Ben Everard**](http://www.techradar.com/author?searchTerm=Ben%20Everard)**from Linux Format Issue 171  June 30th**

<http://www.techradar.com/news/computing/pc/how-to-get-to-grips-with-your-raspberry-pi-s-command-line-interface-1161712>



The command line may look daunting to the uninitiated, but once you master the basics it's an easy beast to tame

**Related stories**

* [**How to install Raspbian on Raspberry Pi from a USB stick**](http://www.techradar.com/news/computing/pc/how-to-install-raspbian-on-raspberry-pi-from-a-usb-stick-1171084)
* [**Raspberry Pi tutorial: how to do more**](http://www.techradar.com/news/computing/pc/raspberry-pi-tutorial-how-to-do-more-1095946)
* [**How to stream files from Raspberry Pi to a smartphone**](http://www.techradar.com/news/computing/pc/how-to-stream-files-to-a-smartphone-from-raspberry-pi--1147006)

As you have no doubt discovered, Raspbian has a graphical interface similar to that of Windows or Mac OS X. You can do most of your day-to-day tasks in this interface. There's a file manager, web browser, text editor and many other useful applications.

However, sometimes you need an interface that's a bit more powerful, and this is where the Command Line Interface (CLI) comes in. It's also known as the Terminal or Shell.

This is an entirely text-based interface, where you type in commands and get a response. We won't lie to you: it will seem a bit confusing at first. Don't worry though, once you've had a bit of practice, it will start to make sense, and spending a little time learning it now will pay dividends in the future.

The first thing you need to do is open up a terminal. Click on LXTerminal on the Raspbian desktop. You should now see a line the looks like:

**pi@raspberrypi $**

This is the command prompt. Whenever you see this, you know the system is ready to receive input. Type **pwd**, and press return. You should see:

**/home/pi**

If you've changed your username, then you'll see a different line. The rather cryptically named **pwd** command stands for Print Working Directory, and the system simply outputs the directory you're currently in. When you start a terminal, it will go to your home directory (see the Raspbian Storage section for a little more information about what /home/ means).

Now we know where we are, the next logical thing to do is move about through the directories. This is done using the **cd** (change directory) command. Try entering:

**cd ..
pwd**

You should find that the system returns **/home**. This is because we've cd'd to '**..**', and two dots always points to the parent directory. To move back to your home directory, you can enter **cd pi**.

There is also another way you can do it. The (pronounced tilda) character always points to your home directory, so wherever you are in the filesystem, you can enter **cd**and you'll move home.

Now type **ls** and hit Enter. This will list all the files in the current directory. One of the big advantages of commands is that we can tell them exactly how we want them to behave. This is done using flags, which come after the command and start with a '**-**'. For example, if we want to list all the files in the current directory (including hidden ones, which start with a '**.**' on Unix-based systems), we use the flag **-a**. So, in your terminal, type **ls -a**.

This time, you should see more files appear. Another flag for **ls** is **-l**. This gives us more information about each file. Try it out by typing **ls -l** . You can even combine flags, such as in **ls -al**.

**Knowing what commands to use**



At this point, you're probably wondering how on earth you are supposed to know what commands and what flags you can use for a task. Well, there's good news and bad news. The good news is that it's usually not too hard to find out the flags for a command. Most support the -h or --help flags, which should give some information about what flags a command can take and how to use it. For example, if you run ls --help, you'll see a long list of flags and what they all do, including:

**-a, --all** do not ignore entries starting with **.**
**…**
**-l** use a long listing format

The second way of finding information on a command is using **man**. This is short for manual. It takes a single argument, that is, a word after the command that isn't preceded by a hyphen. It then displays information on the command given as an argument.

To see the man page for **ls**, type **man ls**. You can navigate through the page using the up and down arrows, or the page up and page down keys to scroll faster. To search for a word or phrase inside the man page, type **/**, then the phrase. For example, **/-l** will find all occurrences of **-l**.

You can use the **N** key and **Shift+N** to scroll forwards and backwards through the occurrences of **-l**. As we introduce more commands, it's good to take a look at the help and the man page to familiarise yourself with what they do.

Of course, you can always Google a command if you find the text-based help a little off-putting, but staying in the terminal will help you become more familiar with the command-line interface.

**How will I know?**



Remember we said there's good news and bad news? Well, the bad news is that it can be quite tricky to find commands if you don't know what they're called.

One helpful feature is the man keyword search. This is done with the **-k** flag. To search for all the programs related to 'browser' on your system, run **man -k browser**. You'll notice that this lists graphical programs as well as command line ones. This is because there's no real difference between the two. You can launch windows from the terminal, and sometimes even control them.

If you've got Iceweasel (a rebranded version of Firefox) installed on your Pi (it's not on there by default), you can open TuxRadar.com in a new tab in a currently running Iceweasel window with the command **iceweasel --new-tab www.tuxradar.com**.

We're now going to quickly introduce a few useful commands. **rm** deletes (ReMoves) a file. **mkdir** makes a new directory. **cp** copies a file from one place to another. This one takes two arguments, the original file and the new file. **cat** outputs the contents of one or more text files. This takes as many arguments as you want, each one a text file, and simply spits their contents out onto the terminal. **less** is a more friendly way of viewing text files. It lets you scroll up and down using the arrow keys. To exit the program back to the command line, press Q.

We'll use all of these commands in examples below, so we won't dwell on them too long. **find** is a useful command for finding files on your computer. You use it in the format find location flags. For example, to find every file on your computer that's changed in the last day, run **find / -mtime 1**

There are more details of what this means, and the different flags that can be used are below.

**Power up**

You can even watch movies in the command line. To stream this classic, just enter telnet towel.blinkenlights.nl and put some popcorn on

So far, you're probably thinking "I could have done all this in the graphical interface without having to remember obscure commands and flags." You'd be right, but that's because we haven't introduced the more powerful features yet.

The first of them are wildcards. These are characters that you can put in that match different characters. This sounds a bit confusing, so we're going to introduce it with some examples. First, we'll create a new directory, move into it and create a few empty files (we use the command touch, which creates an empty file with the name of each argument). Hint - you can use tab completion (see boxout) to avoid having to retype long names, such as in the second line.

**mkdir wildcards
cd wildcards
touch one two three four
touch one.txt two.txt three.txt four.txt**

Then run **ls** to see which files are in the new directory. You should see eight.

The first wildcard we'll use is **\***. This matches any string of zero or more characters. In its most basic usage, it'll match every file in the directory. Try running:

**ls \***

This isn't particularly useful, but we can put it in the middle of other characters. What do you think \*.txt will match? Have a think, then run **ls \*.txt**to see if you are right. How about **one\***? Again, run **ls one\*** to see if you were correct.

The wildcards can be used with any command line programs. They're particularly useful for sorting files. To copy all the .txt files into a new directory, you could run:

**mkdir text-files
cp \*.txt text-files**

We can then check they made it there correctly with:

**ls text-files/**

**Wildcard: ?**

The second wildcard we'll look at is **?**. This matches any single character. What do you think: **ls ???** will match? Have a guess, then run it to see if you're right.

We can also create our own wildcards to match just the characters we want.**[abc]** will match just a lower-case A, B and C. What do you think **ls [ot]\*** will match? Now try **ls [!ot]\*** What difference did the exclamation mark make? It should have returned everything that didn't start with a lower-case letter O or T.

The commands we've looked at so far have all sent their output to be displayed in the terminal. Most of the time this is what we want, but sometimes it's useful to do other things with it.

In Linux, you can do two other things: send it to a file, or send it to another program. To send it to a file, use the character followed by the filename. Run:

**ls > files
cat files**

and you should see that it creates a new file called files, which contains the output of **ls**.

The second option, sending it to another program, is another of the really powerful features of the Linux command line, since it allows you to chain a series of commands together to make one super command.

There are a lot of commands that work with text that are designed to be used in this way. They're beyond the scope of this tutorial, but as you continue to use the command line, you'll come across them and start to see how they can be linked together.

We'll take a look at a simple example. If you run **find /** (don't do it just yet!) it will list every file on the system. This will produce a reel of filenames that will quickly go off the screen. However, rather than direct it to the screen, we can send it to another command that makes the output easier to read. We can use the **less** command that we looked at earlier for this. Run:

**find / | less**

**Take it further**

We've only been able to touch on the basics of using the command line, but you should have enough now to get started, and hopefully you're starting to see just how powerful the command line interface is once you get to know it.

If you want to know more (and you should!) there are loads of resources around both in print and online. [**www.linuxcommand.org**](http://www.linuxcommand.org/) is a great place to start. Their book (The Linux Command Line) is avaialble from bookshops, or for free [**online**](http://linuxcommand.org/learning_the_shell.php).

**sudo**

When using the Raspberry Pi for normal use, you can work with files in your home directory (eg, /home/pi). You will also be able to view most other files on the system, but you won't be able to change them. You also won't be able to install software.

This is because Linux has a permissions system that prevents ordinary users from changing system-wide settings. This is great for preventing you from accidentally breaking your settings. However, there are obviously times when you need to do this.

You can use **/sudo** to run a command as the super user (sometimes called root), which can do pretty much anything on the system. To use it, prefix the command with **sudo**. For example:

**sudo apt-get install synaptic**

will install the package synaptic and make it available to all users.

**Linux Format cut-out-and-keep command line reference**

**Navigation and files**

**cd** Change directory. eg, **cd movies** moves to the movies folder. **cd**moves to your home directory, **cd /** moves to the root directory, and **cd ..** moves back one directory.

**ls** List files. By itself, it lists the files in the current directory. **ls** movies lists the files in the directory movies. **ls -a** lists all files (including hidden ones), and **ls -l** lists more information about each file.

**cp** Copy files. **cp orig-file new-file** copies **orig-file** to **new-file**.

**wget** Downloads a file from the internet. To download the Google home page to the current directory, use **wget www.google.com**.

**df -h** Displays the amount of space left on the device.

**pwd** Displays the current directory.

**Finding files**

**find <location> <tests>** useful flags include: **-mtime <number>** find files modified in the last **<number>** days. **<number>** could be, for example, **2**(exactly two days ago), **-2** (less than two days ago) or **+2** (more than two days ago). **-name <filename>** find files called **<filename>**.

**-iname <filename>** matches files called **<filename>** but not case sensitive.

**-writable** finds files that are writable. There are many more options. See the **man** page for a detailed list. For example, **find / -mtime -2 -writable**finds all files on the filesystem that were changed less than two days ago and are writable by the current user.

**Remote working**

**ssh** Log in to a remote computer using Secure SHell (SSH protocol). **ssh pi@192.168.1.2** will log in as user pi on the computer at the IP address 192.168.1.2. Note, this will only work if the remote computer has an SSH server running.

**scp** Secure CoPy. **scp file pi@192.168.1.2 :/home/pi** will copy file to the directory **home/pi** on the machine with 192.168.1.2. **scp pi@192.168.1.2:/home/pi/file**. will copy **/home/pi/file** from the machine 192.168.1.2 to the current directory. Note, this will only work if the remote machine has an SCP server running.

**Wildcards**

\* Matches any string of characters, or no characters.

**?** Matches any single character.

**[abc]** Matches a, b or c.

**[!abc]** Matches any character except a, b or c.

**[A-Z]** Matches any character in the range A–Z (ie, any upper-case letter).

**[A-z]** Matches any character in the rance A–z (ie, any upper- or lower-case letter).

**[one, two]** Matches the words one and two.

**Information about the computer**

**top** Displays the programs that are currently using the most CPU time and memory.

**uname** Displays information about the kernel. **uname -m** will output the architecture it's running on.

**lscpu** Lists information about the CPU.

**dmesg** Displays the kernel messages (can be useful for finding problems with hardware).

**Text files**

**head** Displays the first ten lines of a text file. Change ten to any other number with the **-n** flag. eg, **dmesg | head -n 15** displays the first 15 lines of the kernel message log.

**tail** Displays the last ten lines of a text file. Can use the **-n** flag like head. Can also keep track of a file as it changes with the **-f** (follow) flag. eg, **tail -n15 -f /var/log/syslog** will display the final fifteen lines of the system log file, and continue to do so as it changes.

**less** Allows you to scroll through a text file.

**cat** Dumps the contents of a text file to the terminal.

**nano** A user-friendly command line text editor (**Ctrl+X** exits and gives you the option to save changes). Special keys **Ctrl+C** Kills whatever program is running in the terminal. **Ctrl+D** Sends the end-of-file character to whatever program is running in the terminal. **Ctrl+Shift+C** Copies selected text to the clipboard. **Ctrl+Shift+V** Pastes text from the clipboard.

**Installing software**

**tar zxvf file.tar.gz**
**tar xjf file.tar.bz**

**./configure** When you unzip a program's source code, it will usually create a new directory with the program in it. **cd** into that directory and run**./configure**. This will check that your system has everything it needs to compile the software.

**make** This will compile the software.

**make install** (needs **sudo**) This will move the newly compiled software into the appropriate place in your system so you can run it like a normal command.

**apt-get** This can be used to install and remove software. For example,**sudo apt-get install iceweasel** will install the package iceweasel (a rebranded version of Firefox). **sudo apt-get purge iceweasel** will remove the package.

**apt-get update** will grab an up-to-date list of packages from the repository (a good idea before doing anything).

**apt-get upgrade** will upgrade all packages that have a newer version in the repository.

**apt-cache search <keyword>** will search the repository for all packages relating to keyword.