

# Command line: Learn the ropes

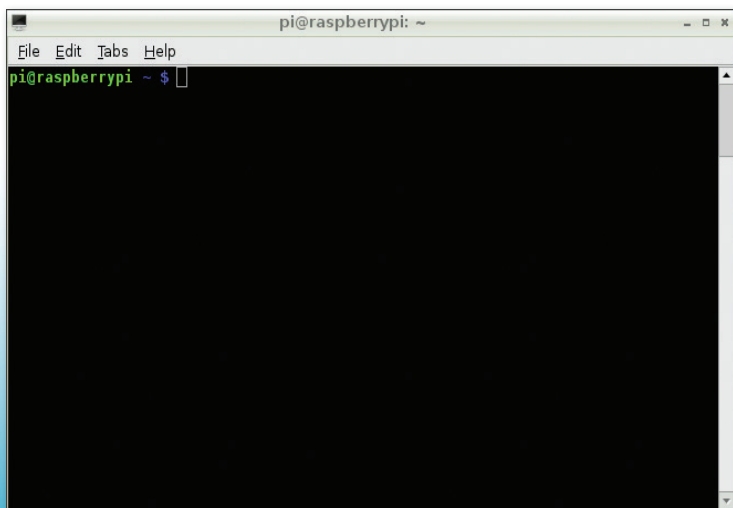
Get to grips with your Raspberry Pi's command line interface and unleash its full power without using the mouse.

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 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.



■ LXTerminal running on the Raspbian desktop.

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 that looks like:

```
pi@raspberrypi ~ $
```

This is the command prompt. Whenever you see this, you know the system is ready to receive input. Now type **pwd**, and press **Enter**. 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.

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**ed 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 now by typing **ls -l**. You can even combine flags, such as in **ls -al**.

## Interactive programs

---

Most of the commands we're dealing with here are non-interactive. That means you set them running and then wait for them to finish. However, not all command line programs work like this. For example, when you first booted Raspbian, it started a config tool that ran in the terminal. There are a few other programs that work in a similar way. Traditionally, the most common has been text editors that allow you to work on files if you don't have a graphical connection.

There are a few quite complicated ones that are great if you spend a lot of time working from the command line, but they can be hard to learn. There's also an easy-to-use terminal-based text editor called **nano**. Enter **nano** followed by a filename at the command prompt to start it. You can then navigate around the text file and make any changes you need. Press **Ctrl+X** to save your work and exit back to the prompt.

# 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.

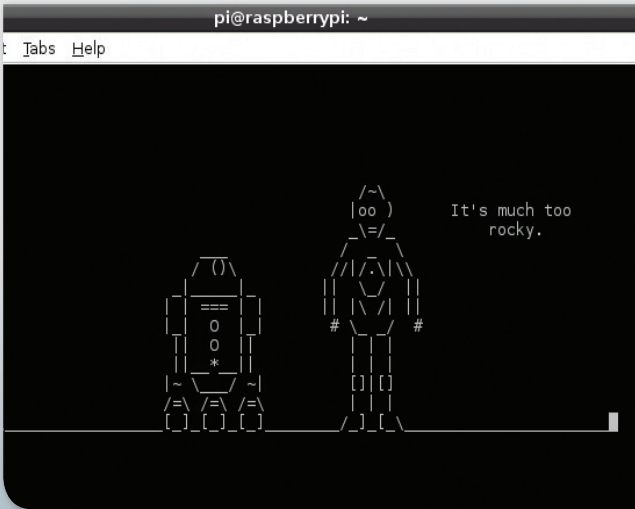
Well, there's good news and bad news. The good news is that it's usually not too difficult to find out the flags for a command. Most commands support the **-h** or **--help** flags, which should give some information about what flags a command can take and how to use it. For example, if you run **ls --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 familiarize 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.



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

## Tab completion

When you're dealing with long filenames, it can be very annoying to have to type them out every time you want to run a command on them. To make life a bit easier, the terminal uses tab completion. This means that if you start typing a filename and press the **Tab** key, the system will try to fill in the rest of the name. If there's only one file that fits what you've typed so far, it will fill in the rest of the name for you (try typing **cd /h** then pressing **Tab**). If there are more than one, it will fill in as far as the two are the same. If you press **Tab** again, it will show the options (try typing **cd /m**, and then pressing **Tab** twice).

# 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 on to 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 these commands in examples below, so we won't dwell on them for too long.

```
File Edit Tabs Help
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 speci-
    fied.

    Mandatory arguments to long options are mandatory for short options
    too.

    -a, --all
        do not ignore entries starting with .

    -A, --almost-all
        do not list implied . and ..

    --author
    Manual page ls(1) line 1 (press h for help or q to quit)
```

■ Probably the most important command in any Unix-like system is **man**, since it is the key to understanding every other command. Take time to become familiar with the structure and language used and it will make life easier in the future.

## find

**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 over the page.

# Power up

So far you're probably thinking to yourself, "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 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/
```

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.

## sudo

When using the Raspberry Pi for normal use, you can work with files in your home directory (for example, **/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 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.

# Pipes and redirection

The commands that we've been looking 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 with a command: 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
```

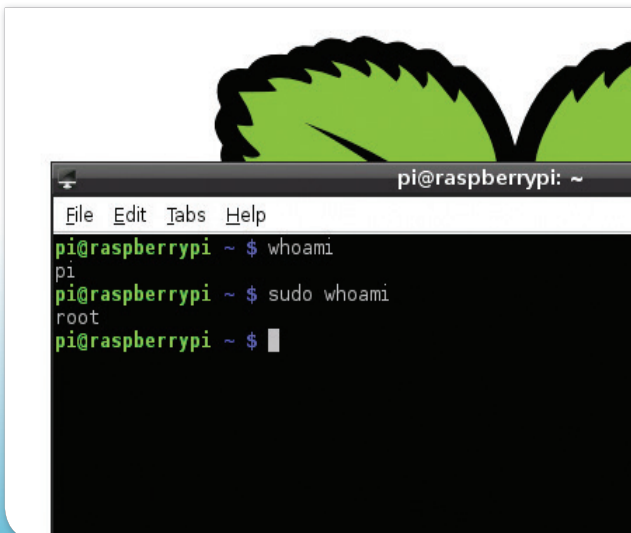
and you should see that it creates a new file called **files**, which contains the output of **ls**.

The second option, sending it to another program, is another of the really powerful features of the Linux command line, because 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 (or 'pipe') it to another command that makes the output easier to read. We can use the **less** command that we looked at on page 13 for this. Run:

```
find / | less
```



■ Use the **sudo** command to switch between the normal user 'pi', and the superuser 'root'.

## Take it further

We've only been able to touch on the basics of using the command line, but you should have enough knowledge now to get started, and hopefully you're beginning 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 in print and online. [LinuxCommand.org](http://LinuxCommand.org) is a great place to start. Its book (*The Linux Command Line*) is available from bookshops, or for free online [www.linuxcommand.org/lc3\\_learning\\_the\\_shell.php](http://www.linuxcommand.org/lc3_learning_the_shell.php).

# Packages: How do they work?

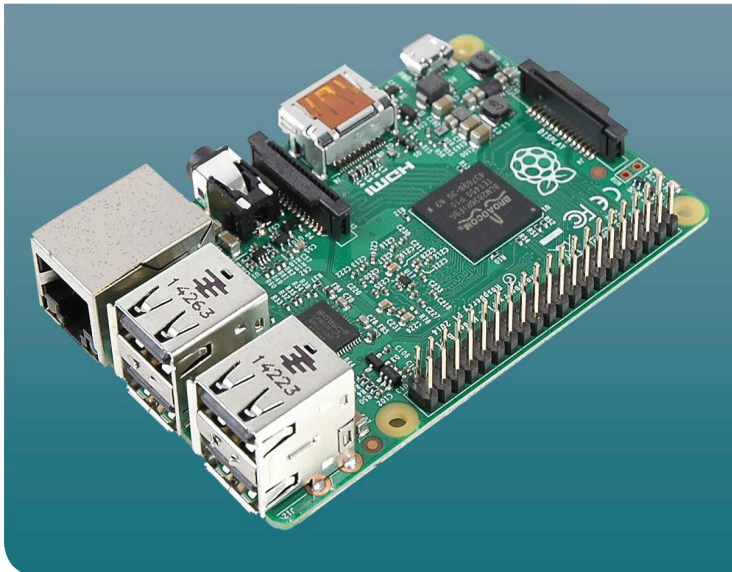
Discover how Raspbian's package manager, **apt-get**, gets software from online repositories and manages it on your system.

If you're used to Windows, you may be used to each bit of software having its own installer, which gets the appropriate files and puts them in the appropriate places.

Linux doesn't work in the same way (at least, it doesn't usually). Instead, there is a part of the operating system called the package manager. This is responsible for getting and managing any software you need. It links to a repository of software so it can download all the files for you. Since Linux is built on open source software, almost everything you will need is in the repositories and free. You don't need to worry about finding the install files, or anything like that – the package manager does it all for you.

There are a few different package managers available for Linux, but the one used by Raspbian is **apt-get**. Arch uses a different one, so if you want to try this distribution on your Raspberry Pi, you'll need to familiarize yourself with the **pacman** software, which we won't cover here.

Before we get started, we should mention that since this grabs software from the online repositories, you will need to connect your Pi to the internet before following this section.



**apt-get** is a command line program, so to start with you'll need to open a terminal (see the command line interface section on page 10 for more information on this). Since package management affects the whole system, all the commands need to be run with **sudo**. The first thing you need to do is make sure you have the latest list of software available to you. Run:

```
sudo apt-get update
```

Since all the software is handled through the package manager, it can update all the software for you so you don't need to bother doing it for each program separately. To get the latest versions of all the software on your system, run:

```
sudo apt-get upgrade
```

This may take a little while, because open source software tends to be updated quite regularly. In Linux terms, you don't install particular applications, but packages. These are bundles of files. Usually, each one represents an application, but not always. For example, a package could be documentation, or a plug-in, or some data for a game. In actual fact, a package is just a collection of files that can contain anything at all.

In order to install software with **apt-get**, you need to know the package name. Usually this is pretty obvious, but it needs to be exactly right for the system to get the right package. If you're unsure, you can use **apt-cache** to search through the list of available packages. Try running:

```
apt-cache search iceweasel
```

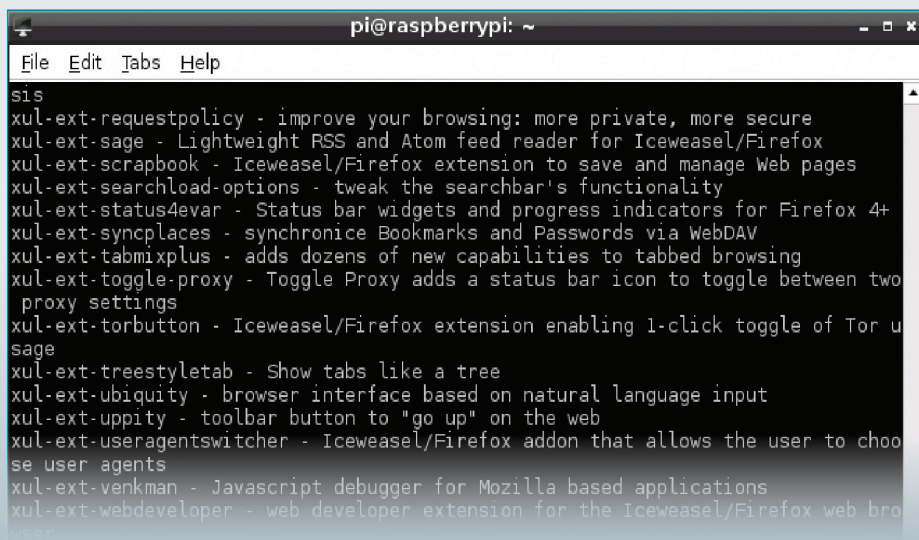
This will spit out a long list of packages that all have something to do with the web browser (*iceweasel* is a rebranded version of *Firefox*).

To install *iceweasel*, run:

```
sudo apt-get install iceweasel
```

You will notice that **apt-get** then prompts you to install a number of other packages. These are dependencies. That means that *iceweasel* needs the files in these packages in order to run properly. Usually you don't need to worry about these – just press **Y** and the package manager will do everything for you.

However, if your SD card is running low on space, you may sometimes come across a program that has so many dependencies that they'll overfill the device. In these cases, you'll either need to free up some space or find another application that has fewer dependencies.

A screenshot of a terminal window titled 'pi@raspberrypi: ~'. The window has a menu bar with 'File', 'Edit', 'Tabs', and 'Help'. The terminal output shows the command 'apt-cache search iceweasel' and its results: 'xul-ext-requestpolicy - improve your browsing: more private, more secure', 'xul-ext-sage - Lightweight RSS and Atom feed reader for Iceweasel/Firefox', 'xul-ext-scrapbook - Iceweasel/Firefox extension to save and manage Web pages', 'xul-ext-searchload-options - tweak the searchbar's functionality', 'xul-ext-status4ever - Status bar widgets and progress indicators for Firefox 4+', 'xul-ext-syncplaces - synchronise Bookmarks and Passwords via WebDAV', 'xul-ext-tabmixplus - adds dozens of new capabilities to tabbed browsing', 'xul-ext-toggle-proxy - Toggle Proxy adds a status bar icon to toggle between two proxy settings', 'xul-ext-torbutton - Iceweasel/Firefox extension enabling 1-click toggle of Tor usage', 'xul-ext-treestyletab - Show tabs like a tree', 'xul-ext-ubiquity - browser interface based on natural language input', 'xul-ext-uppity - toolbar button to "go up" on the web', 'xul-ext-useragentswitcher - Iceweasel/Firefox add-on that allows the user to choose user agents', 'xul-ext-venkman - Javascript debugger for Mozilla based applications', and 'xul-ext-webdeveloper - web developer extension for the Iceweasel/Firefox web browser'.

■ **apt-cache** in a terminal will give you a list of available packages.



If you then want to remove *iceweasel*, you can do it with:

```
sudo apt-get purge iceweasel
```

The package manager will try to remove any dependencies that aren't used by other packages.

You'll often see packages with **-dev** at the end of package names. These are only needed if you're compiling software. Usually you can just ignore these.

**apt-get** is a great tool, but it isn't as user-friendly as it could be. There are a couple of graphical tools that make package management a bit easier. The first we'll look at is *Synaptic*. This isn't installed by default on Raspbian, so you'll have to get it using **apt-get** with:

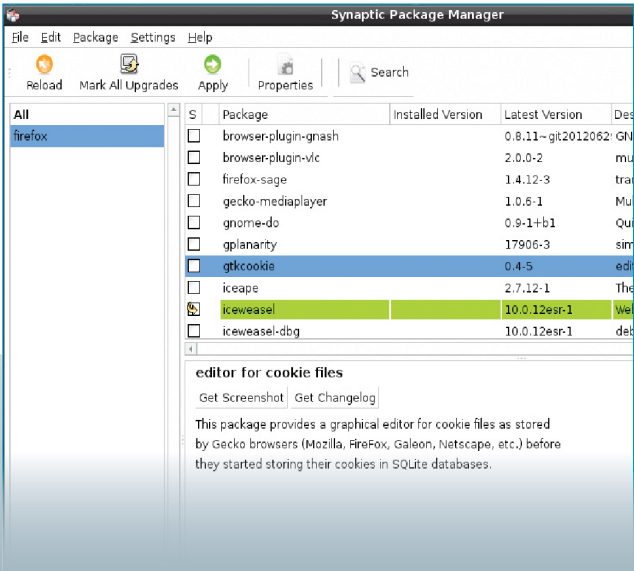
```
sudo apt-get synaptic
```

This works with packages in the same way as **apt-get**, but with a graphical interface. Once the package is installed, you can start it with:

```
--where is synaptic
```

See below for more information.

The second graphical tool is the Raspberry Pi App store. Unlike **apt-get** and *Synaptic*, this deals with commercial software as well as free software, so you have to pay to install some things. It comes by default on Raspbian, and you can get it by clicking on the icon on desktop. See 'Raspberry Pi Store' on the next page for more information.



### Synaptic

*Synaptic* lets you do everything you can with the command line **apt-get**, but the graphical interface is easier to use. We find it especially useful when searching, because the window is easier to look through than text on the terminal.

■ *Synaptic* provides a user-friendly front-end to the apt package management system.

# Compiling software

Sometimes, you'll find you need software that isn't in the repository, and so you can't get it using **apt-get**. In this case, you'll need to compile it.

Different projects package their source code in different ways, but usually, the following will work. Get the source code from the project's website, and unzip it. Usually, the filename will end in .tar.gz or .tgz. If this is the case, you can unzip it with:

```
tar xzvf <filename>
```

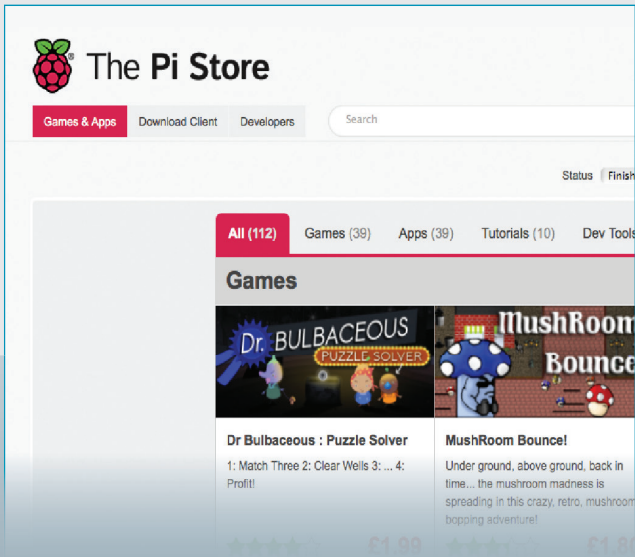
If the filename ends in .tar.bz2, you need to replace **xzvf** with **xjfb**. This should now create a new directory which you need to **cd** into. Hopefully, there'll be a file called **INSTALL**, which you can read with **less INSTALL**. This should tell you

how to continue with the installation. Usually (but not always), it will say:

```
./configure
make
sudo make install
```

The first line will check you have all the necessary dependencies on your system. If it fails, you need to make sure you have the relevant -dev packages installed.

If that all works, you should have the software installed and ready to run.



**Raspberry Pi store**

The Raspberry Pi store allows users to rate the software, so you can see how useful other people have found it. It also includes some non-free software. However, it doesn't have anywhere near the range that is available through **apt-get** or *Synaptic*.

■ The Pi Store is an app store for your Raspberry Pi.