

*The*  
**MagPi**  
ESSENTIALS

[ 2<sup>ND</sup> ]  
[ EDITION ]

# CONQUER THE COMMAND LINE

>\_

*The Raspberry Pi*  
TERMINAL GUIDE

Written by **Richard Smedley**

**UPDATED FOR 2019!**  
+ 4 NEW CHAPTERS



# *The* **MagPi**

## ESSENTIALS

LEARN | CODE | MAKE

### AVAILABLE NOW:

- > CONQUER THE COMMAND LINE
- > EXPERIMENT WITH SENSE HAT
- > MAKE GAMES WITH PYTHON
- > CODE MUSIC WITH SONIC PI
- > LEARN TO CODE WITH SCRATCH
- > HACK & MAKE IN MINECRAFT
- > ELECTRONICS WITH GPIO ZERO
- > LEARN TO CODE WITH C
- > THE CAMERA MODULE GUIDE
- > AIY PROJECTS

---

*The*  
**MagPi**  
ESSENTIALS

From the makers of the  
official Raspberry Pi magazine

---

---

# OUT NOW IN PRINT

# ONLY £3.99

---

[store.rpipress.cc](http://store.rpipress.cc)



GET THEM  
DIGITALLY:



Available on the  
**App Store**



GET IT ON  
**Google Play**

# WELCOME TO CONQUER THE COMMAND LINE

**S**ometimes only words will do. Graphic user interfaces (GUIs) were a great advance, creating an easy route into computer use for many non-technical users. For complex tasks, though, the interface can become a limitation: blocking off choices, and leaving a circuitous route even for only moderately complicated jobs.

(Re-)Enter the command line: the blinking cursor that many thought had faded away in the 1990s. For getting instructions from user to computer – in a clear, quick, and unambiguous form – the command line is often the best way. It never disappeared on UNIX systems, and now, thanks to Raspbian on the Raspberry Pi, a new generation is discovering the power of the command line to simplify complex tasks, or instantly carry out simple ones.

If you're not comfortable when faced with the \$ prompt, then don't panic! In this fully updated book, we'll quickly make you feel at home, and able to find your way around the terminal on the Pi, or any other GNU/Linux computer: getting things done, and unlocking the power of the command line.

**FIND US ONLINE** [raspberrypi.org/magpi](http://raspberrypi.org/magpi)

**GET IN TOUCH** [magpi@raspberrypi.org](mailto:magpi@raspberrypi.org)

**The  
MagPi**

## PUBLISHING

Publishing Director: **Russell Barnes**  
Director of Communications: **Liz Upton**  
CEO: **Eben Upton**



## DESIGN

Critical Media: [criticalmedia.co.uk](http://criticalmedia.co.uk)  
Head of Design: **Lee Allen**  
Designer: **Mike Kay**

## EDITORIAL

Editor: **Phil King**  
Writer: **Richard Smedley**  
Contributors: **Lucy Hattersley,**  
**Simon Long**



In print, this product is made using paper sourced from sustainable forests and the printer operates an environmental management system which has been assessed as conforming to ISO 14001.

This book is published by Raspberry Pi (Trading) Ltd, Maurice Wilkes Building, St. John's Innovation Park, Cowley Road, Cambridge, CB4 0DS. The publisher, editor and contributors accept no responsibility in respect of any omissions or errors relating to goods, products or services referred to or advertised in this product. Except where otherwise noted, content in this book is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY-NC-SA 3.0). ISBN: 978-1-912047-66-6

# The MagPi

## ESSENTIALS

### CONTENTS

#### 06 [ CHAPTER ONE ]

##### **DON'T PANIC**

Take a look around and discover things

#### 11 [ CHAPTER TWO ]

##### **READ/WRITE TEXT**

Get working on files

#### 16 [ CHAPTER THREE ]

##### **PERMISSION TO INSTALL**

Raspbian's system for installing and updating

#### 21 [ CHAPTER FOUR ]

##### **MANIPULATING TEXT**

Connect together multiple simple commands

#### 26 [ CHAPTER FIVE ]

##### **CUSTOMISE THE COMMAND LINE**

Make Raspbian more personal

#### 31 [ CHAPTER SIX ]

##### **CONNECTING DISKS**

Tackle the management of removable storage

#### 36 [ CHAPTER SEVEN ]

##### **PREDICTABLE NETWORKING**

Give the Pi a permanent network address of its own

#### 41 [ CHAPTER EIGHT ]

##### **STOPPING A PROCESS**

No need to turn it off and on again: just kill the process!

#### 46 [ CHAPTER NINE ]

##### **REMOTE PI**

Access the Pi with Secure Shell

#### 51 [ CHAPTER TEN ]

##### **DOWNLOADING & INSTALLING**

Add software and write to SD cards

#### 56 [ CHAPTER ELEVEN ]

##### **START AND STOP AT YOUR COMMAND**

Manage startup and shutdown

#### 64 [ CHAPTER TWELVE ]

##### **SAVE IT NOW!**

Protect your data with backups

#### 74 [ CHAPTER THIRTEEN ]

##### **EASY COMPILATION**

Build software from source code

#### 82 [ CHAPTER FOURTEEN ]

##### **COMMANDING THE INTERNET**

Get online from the command line

#### [ RICHARD SMEDLEY ]



Since soldering together his first computer – a ZX81 kit – and gaining an amateur radio licence as GW6PCB, Richard has fallen in and out of love with technology. Swapping the ZX81 for a guitar, and dropping ham radio for organic horticulture, he eventually returned to the command line, beginning with a computer to run his own business, and progressing to running all the computers of an international sustainability institution. Now he writes about Free Software and teaches edible landscaping.

# [ CHAPTER ONE ] DON'T PANIC

In the first chapter, we take a look around and discover that things aren't as strange as they might appear...

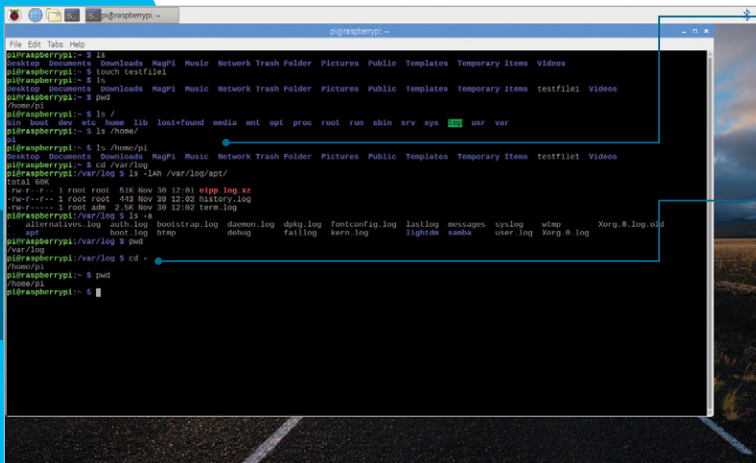
[ READ THE MANUAL ]

Help is included, with `man(ual)` pages, but they can be a little overwhelming. Use them to check out some extra options beyond the switches like `-a` we use here. To read the `ls` man page, type `man ls`.

It's not a throwback to the past, but a quick and powerful way of getting your Raspberry Pi to do what you want, without all that RSI-inducing menu chasing and icon clicking. The command-line interface was a great step up from manually toggling in your instructions in octal (base-8), using switches on the front of the machine! Graphical user interfaces (GUIs) brought friendly visual metaphor to the computer, losing some power and expressiveness. With the Raspberry Pi, you can get the best of both worlds by knowing both: after reading through this guide, you'll soon be as comfortable at the command prompt as you are at your desktop.

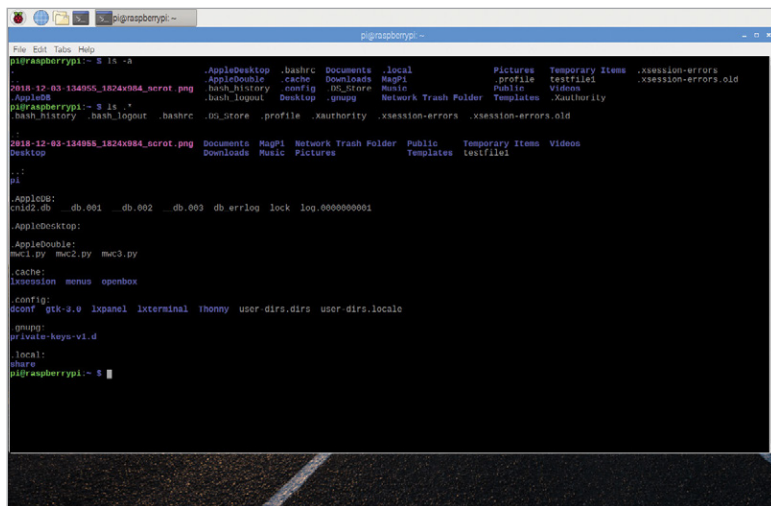
Unlike some earlier versions of Raspbian, Stretch boots you straight to a GUI, although you can change this behaviour in the settings. The command-line environment is still there: hold down the **ALT+CTRL** keys and press **F1** (the first function key on the keyboard), and you'll arrive at a 'virtual console'. Press **ALT+F2** through to **F6** and you'll find five further consoles waiting for you to log in.

You can drop into these any time you like, but for now press **ALT+F7** and you'll be back in mouse and menu land. The command line is also available through a program called a terminal emulator (often referred to as a term or xterm). You'll also find people referring to the shell, or Bash. Don't worry about that for now; just click on the icon at the top of the screen that looks like a black television screen,



The command line is only a click away: it is called Terminal and you can find it under Accessories in the menu

Commands are terse, but, once learned, they're a quick way of navigating and reading your files and folders



**Fig 1** Switches modify behaviour in commands; **ls -la** shows (dot) files in your listing that are usually hidden from view

**[ PRESS RETURN ]**

To save repeating it in the text, we'll confirm here that each time you type in a command, you need to hit the **RETURN** or **ENTER** key at the end, to tell the Pi you've issued Bash with a command.

or go to Accessories>Terminal in the menu: the terminal now awaits your commands.

## Look around

If you're used to looking at files and folders in a file manager, try to clear your mind of the icons and concentrate on the names. Type **ls** and press **RETURN** (see 'Press Return' box). On a fresh Raspbian Stretch with Recommended Software install, you'll just see a few directories, including **MagPi**. Type **ls MagPi** (see 'Lazy Completion' box) and you'll see a listing of what's in it.

Commands like **ls** are not cryptic (at least not intentionally) but they are terse, dating back to a time when the connection to the computer was over a 110 baud serial line, from an ASR 33 teletype terminal. If you think it's strange to be defined by 50-year-old technology, just remember that your QWERTY keyboard layout was reputedly designed both to stop mechanical typewriter keys jamming, and to enable salespeople to quickly type 'typewriter' using the top row!

## File path

You can list files and folders anywhere in your system (or other connected systems) by providing the path as an argument to your





command. The path is the folder hierarchy: on a Windows computer, in a graphical file browser, it starts with ‘My Computer’; on your Pi it starts at `/`, pronounced ‘root’ when used on its own as the root of your file system. Try entering `ls /` – again we get terseness, and names like ‘bin’, which is short for binary, and is the directory where many programs are kept (enter `ls /bin` to see the details). `ls /dev` shows hardware devices in the Pi. Try `ls /home` – see that ‘pi’? That’s you: you are logged in as user pi. If you’ve changed your login name, or if you have created extra users, they’ll all be listed there too: every user gets their own home directory; yours is the `/home/pi` folder in which we found ourselves in earlier. Before, with **MagPi**, we used the relative path (the absolute path would be `/home/pi/MagPi`) because we’re already home. If you need to check your location, type `pwd` (present working directory).

“ Commands are not cryptic (at least not intentionally), but they are terse ”

### There’s no place like ~

For any logged-in user, their home directory is abbreviated as `~` (the tilde character). Type `ls ~` and you’ll see. There’s apparently not much in your home directory yet, but Raspbian keeps a lot hidden from the casual glance: files and folders beginning with a dot, known as ‘dot files’, contain configuration information for your system and its programs. You don’t need to see these files normally, but when you do, just ask `ls` to show you all files with a command switch. You can do this with either the full switch `--all`, or the abbreviation `-a` like so: `ls -a ~`. Traversing the pathways of the directory hierarchy can be easier from the command line than clicking up and down a directory tree, particularly with all the shortcuts given. Your `ls -a` showed you `.` and `..` as the first two directories; these shortcuts represent the current and the parent directory respectively. Try listing the parent directory – from `/home/pi`, entering `ls ../..` will show you two layers up. If you want to list the hidden files without the `.` and `..` appearing (after all, they’re present in every directory, so you don’t need to be told), then the switch to use is `-A`.

### [ LAZY COMPLETION ]

You don’t need to type all of `ls MagPi` (for example) – after `ls M`, hit the **TAB** key and it will auto-complete. If you’ve more than one file beginning with `M`, they’ll all be listed and you can type more letters and hit **TAB** again.

Before we move on to other commands, let's look briefly at chaining switches together: **ls -lh ~**

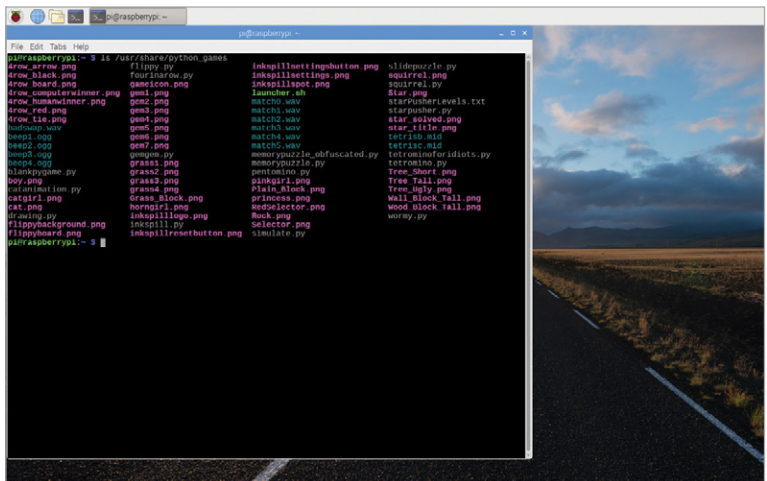
**-l** gives you more information about the files and folders, and **-h** changes the units from bytes to kB, MB, or GB as appropriate. We'll look at some of the extras the **-l** listing shows you in more detail later, particularly in chapters two and three.

## Time for change

That's enough looking: let's start moving. **cd** is short for change directory, and moves you to anywhere you want in the file system: try **cd /var/log** and have a look (**ls**, remember). Files here are logs, or messages on the state of your system that are saved for analysis later. It's not something you'll often need to think about: Raspbian is a version of an operating system that also runs across data centres and supercomputers, where problem monitoring is very important. It is, however, useful to know, particularly if you have a problem and someone on a forum advises you to check your logs.

**cd ~** will take you where you expect it. Try it, then **pwd** to check. Now try **cd -** (that's a hyphen), the **'-'** is a shortcut for 'wherever I was before I came here'. Now we've looked around, we can move on to beginning to do things to our files.

Right Who needs icons when you can fit a listing of 78 files into a small window? Coloured fonts indicate file types

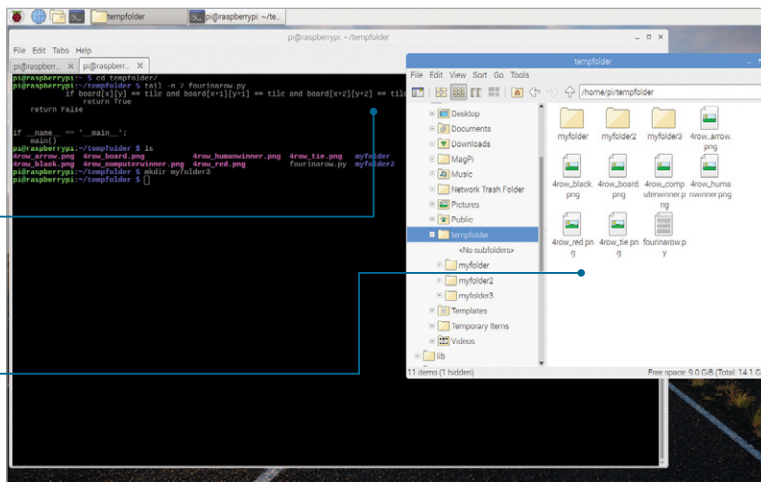


# [ CHAPTER TWO ] READ/WRITE TEXT

In this chapter, we get working on files

The command line offers tools to get text from different parts of a file, like skipping to the conclusion

Create and name files and directories with keystrokes, rather than mouse-clicks and keystrokes



### [ MORE INFO ]

Many utilities have info pages, giving far more information than their man page. If you're feeling brave, try [info nano](#) for a comprehensive guide to nano.

Now that we can navigate folders and list files, it's time to learn how to create, view, and alter both files and folders. Once more, it's not that the task is difficult, rather that the forms of the commands (particularly when editing) are unfamiliar, coming from an era before Common User Access (CUA) standards were created to ease switching between applications and operating systems.

Stick with it: with just the first two chapters of this book under your belt, you'll be able to do plenty of work at the command line, and start getting comfortable there.

## Creating a directory

We're going to dive straight into working with files and folders by creating a new directory. Assuming you already have a terminal open (see 'Instant applications' box), and you're in your home directory (`pwd` to check, `cd ~` to get back there if necessary), type `mkdir tempfolder` and have a look with `ls`.

`mkdir`, as you've probably guessed, creates a new directory or folder. Let's use it to experiment with altering one of the included Python games. Don't worry: we're not going to be programming Python, just making a small change by way of illustration. `cd tempfolder` (use tab completion: `cd t` then hit the **TAB** key). In the following example, we'll be copying some files to this directory.

First, make sure Python Games is installed – if not, click the top-left Raspberry Pi icon on the desktop, select Preferences, then Recommended Software, tick the box next to Python Games in the list, and then click Apply to install it.

We'll copy over the files from the **python\_games** directory:

```
cp /usr/share/python_games/fourinarow.py .
cp /usr/share/python_games/4row_* .
```

## Wildcard

The **.** (dot) at the end of the commands refers to 'just here', which is where we want the files copied. Also, **4row\_\*** is read by the Pi as 'every file beginning **4row\_**' – the **\*** is known as a wildcard, and this one represents any number of characters (including none); there are other wildcards, including **?**, which means any single character.

Try **python fourinarow.py** and you'll see you can run the local copy of the game. To change the game, we need an editor – sidestepping the UNIX debate about which one is best, we'll use the Pi's built-in editor: nano. Unless you've previously used the Pico

“ We're going to dive straight into working with files and folders by creating a new directory

editor, which accompanied the Pine email client on many university terminals in the 1980s and 1990s, it will seem a little odd (**Fig 1**, overleaf). That's because its conventions predate the **CTRL+C** for copy type standards found in most modern programs. Bear with us.

## Editing and paging

**nano fourinarow.py** will open the game for editing; use the arrow keys to go down nine lines, and along to the **BOARDHEIGHT** value of **6**. Change it to **10** (both the **BACKSPACE** and **DELETE** keys will work in nano). The last two lines of the screen show some shortcuts, with

### [ INSTANT APPLICATIONS ]

Although you can open the terminal emulator from the menu – Accessories > Terminal – for this, and any other app, just hit **ALT+F2** and type its command name: **lxterminal**.

## [ SWITCHING HELP ]

You don't need to wade through the man page to see what switches are available: type `--help` after the command to be shown your options, e.g. `rm --help`

^ (the caret symbol) representing the **CTRL** key: **CTRL+O**, followed by **RETURN** will 'write out' (save) the file; then use **CTRL+X** to exit. Now, `python fourinarow.py` will open an oversized board, giving you more time to beat the computer, should you need it. However, there's now no room to drag the token over the top of the board: go back and change the `BOARDHEIGHT` value to `9`, with nano.

If you want to take a look through the `fourinarow.py` listing without entering the strange environment of nano, you can see the entire text of any file using `cat`: e.g., `cat fourinarow.py`. Unfortunately, a big file quickly scrolls off the screen; to look through a page at a time, you need a 'pager' program. `less fourinarow.py` will let you scroll up and down through the text with the **PAGE UP** and **PAGE DOWN** keys. Other keys will do the same job, but we'll leave you to discover these yourself. To exit `less`, hit **Q** (this also works from `man` and `info` pages, which use a pager to display text).

## Cats, heads & tails

If editor wars are a UNIX tradition we can safely ignore, there's no getting away from another tradition: bad puns. `less` is an improvement over `more`, a simple pager; the respective man pages will show you the differences. One advantage the relatively primitive `more` has is that at the end of a file it exits automatically, saving you reaching for the **Q** button. Admittedly, this is not a huge advantage, and you can always use `cat`.

Fortunately, `cat` is not a feline-based pun, but simply short for 'concatenate': use it with more than one file and it concatenates them together. Used with no argument – type `cat` – it echoes back what you type after each **ENTER**. Hit **CTRL+C** to get out of this when you've finished typing in silly words to try it. And remember that **CTRL+C** shortcut: it closes most command-line programs, in the same way that **ALT+F4** closes most windowed programs.

**Fig 1** The default editor, nano, has unusual command shortcuts, but they're worth learning, as you'll find nano installed on virtually all Linux boxes, such as your web host

```

pi@raspberrypi: ~/tempfolder
File Edit Jobs Help
pi@raspberr... x pi@raspberr... x
GNU nano 2.2.6 File: fourinarow.py

# Four-In-A-Row (a Connect Four clone)
# By Al Sweigart al@inventwithpython.com
# http://inventwithpython.com/pygame
# Released under a "Simplified BSD" license

import random, copy, sys, pygame
from pygame.locals import *

BOARDWIDTH = 7 # how many spaces wide the board is
BOARDHEIGHT = 9 # how many spaces tall the board is
assert BOARDWIDTH >= 4 and BOARDHEIGHT >= 4, 'Board must be at least 4x5'

DIFFICULTY = 2 # how many moves to look ahead. (>2 is usually too much)

SPACESIZE = 50 # size of the tokens and individual board spaces in pixels

FPS = 30 # frames per second to update the screen
WINDOWWIDTH = 640 # width of the program's window, in pixels
WINDOWHEIGHT = 480 # height in pixels
    
```

You can peek at the first or last few lines of a text file with **head** and **tail** commands. **head fourinarow.py** will show you the first ten lines of the file. **head -n 5 fourinarow.py** shows just five lines, as does **tail -n 5 fourinarow.py** with the last five lines. On the Pi, **head -5 fourinarow.py** will also work.

## Remove with care

**nano afile.txt** will create a new file if **afile.txt** does not already exist: try it, and see if it works when you exit the file before writing and saving anything. We've done a lot already (at least, nano makes it feel like a lot), but it's never too early to learn how to clean up after ourselves. We'll remove the files we've created with **rm**. The remove tool should always be used with care: it has some built-in safeguards, but even these are easy to override (**Fig 2**). In particular, *never* let anyone persuade you to type **rm -rf /** – this will delete the entire contents of your Pi, all the programs, everything, with little to no chance of recovery.

Have a look at what files we have: if you're still in the **tempfolder** you made, then **ls** will show you the Four-in-a-Row files you copied here. Remove the program, then the .png files with careful use of the **\*** wildcard.

```
rm fourinarow.py
rm 4row*.png
```

**cd ..** to get back to **/home/pi** and **rm -r tempfolder** will remove the now empty folder. The **-r** (recursive) option is necessary for directories, and will also remove the contents if any remain.

In the next chapter, we'll delve into file permissions and updating your Pi's software from the command line.

```

pi@raspberrypi ~/tempfolder
File Edit Tabs Help
pi@raspberr... x | pi@raspberr... x
pi@raspberrypi ~/tempfolder $ ls
4row_arrow.png      desktop_1_002.png
4row_black.png      desktop_1_003.png
4row_board.png       fourinarow.py
4row_computerwinner.png  myfolder
4row_humanwinner.png  myfolder2
4row_red.png         myfolder3
4row_tie.png         pi@raspberrypi: --tempfolder_004.png
desktop_1_001.png
pi@raspberrypi ~/tempfolder $ rm -r myfolder*
pi@raspberrypi ~/tempfolder $ ls
4row_arrow.png      4row_tie.png
4row_black.png      desktop_1_001.png
4row_board.png       desktop_1_002.png
4row_computerwinner.png  desktop_1_003.png
4row_humanwinner.png  fourinarow.py
4row_red.png         pi@raspberrypi: --tempfolder_004.png
pi@raspberrypi ~/tempfolder $

```

**Fig 2** **rm** is a powerful removal tool: use with great care!

# [ CHAPTER **THREE** ]

# PERMISSION TO INSTALL

We look at Raspbian's efficient system for installing and updating software, among other things



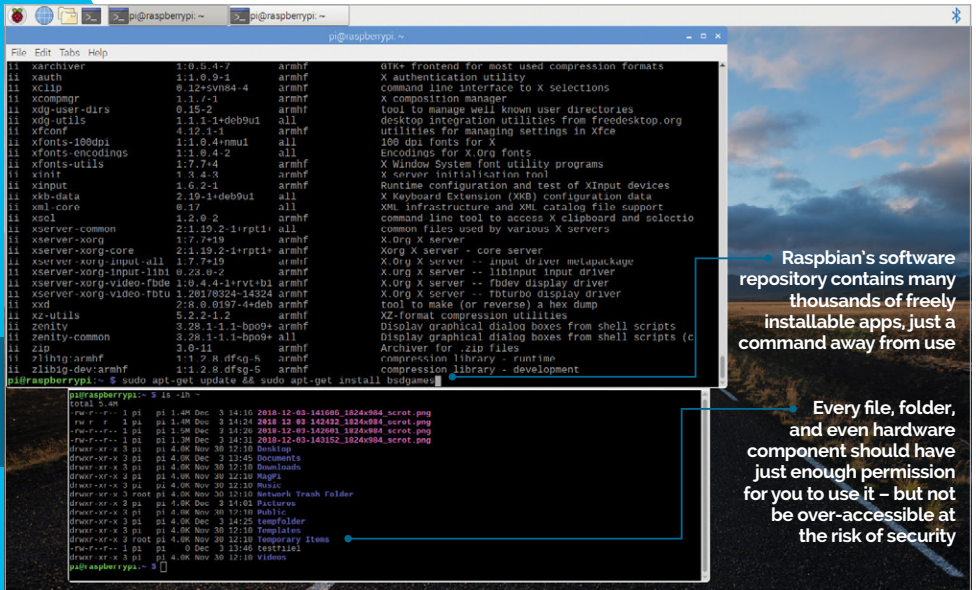
Installing software should be easy, but behind every piece of software is an evolving set of dependencies that also need installing and updating. Keeping them separate reduces unnecessary bloat and duplication, but adds the potential for bugs, missing files, and even totally unresolvable clashes.

Fortunately, Debian GNU/Linux cracked the problem back in the 1990s with the Debian Package Management system and the Advanced Package Tool (APT) – and Debian-based systems, like Ubuntu and the Pi's Raspbian, inherit all of the benefits. Here we'll show you the basics you need to know to install new software and keep your system up to date from the command line, and then look at the not entirely unrelated field of file ownership and permissions.

Using the **apt** command to update your system's list of installable software should be as simple as issuing the command like so: **apt-get update**. Try this logged in as user pi, though, and you'll just get error messages. The reason for this is that changing system software on a GNU/Linux (or any type of UNIX) system is a task restricted to those with administrative permissions: the godlike superuser, or admin, also known as root.

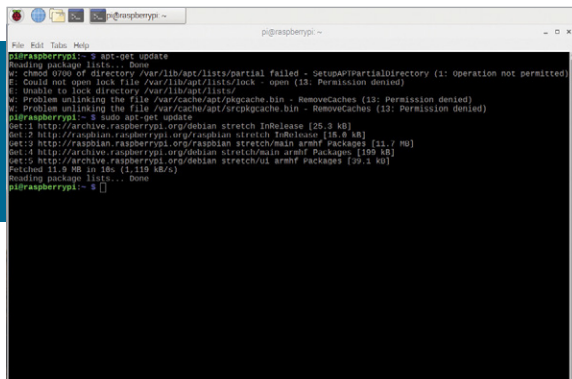
[ SHARED RESPONSIBILITY ]

If you share your Pi, read up on **sudo** and the **visudo** command to find how to give limited but useful admin privileges to the other users.



Raspbian's software repository contains many thousands of freely installable apps, just a command away from use

Every file, folder, and even hardware component should have just enough permission for you to use it – but not be over-accessible at the risk of security



**Fig 1** Raspbian updates its listing of thousands of available apps, providing you give it admin permissions

## Pseudo root, su do

We'll get onto permissions properly a bit later, but for now you'll be pleased to know that you can fake it, using the **sudo** command. **sudo** potentially offers a fine-grained choice of permissions for users and groups to access portions of the admin user's powers. However, on the Pi, Raspbian assumes, quite rightly, that the default user will be someone just wanting to get on with things, and **sudo** in front of a command will pretty much let you do anything. You have been warned!

The following two commands will update Raspbian's installed software (**Fig 1**):

```

sudo apt-get update
sudo apt-get upgrade
  
```

You can wait for one to finish, check everything is OK, then issue the other command, or you can save waiting and enter both together with:

```

sudo apt-get update && sudo apt-get upgrade
  
```

The **&&** is a Boolean (logical) AND, so if the first command doesn't run properly, the second one will not run at all. This is because for a logical AND to be true, both of its conditions must be true.

It's always worth running the **update** command before installing new software, too – minor updates are made even in stable distributions such as Raspbian, to address any issues. We've just

run an update, so no need to repeat that for now. Sticking with a command-line theme, we're going to install an old suite of terminal games:

```
sudo apt-get install bsdgames
```

## Searchable list

It is possible to find particular apps with apt-cache search: **apt-cache search games**. You can also examine individual packages with apt-cache show: **apt-cache show bsdgames**.

APT is actually a front end to the lower-level **dpkg**, which you can call to see what you have installed on the system: **dpkg -l**. Even on a fresh system, that's a large listing; we'll show you how to get useful information from such listings another time.

Downloaded packages hang around in **/var/cache/apt** and if you find yourself short on disk space, issuing **sudo apt-get clean** will clear out the archive, without affecting the installed software.

Now, remember the extra details that **ls -lh** showed us in chapter 1? Try **ls -lh /etc/apt**.

That **-rw-rw-r--** at the beginning of the listing for **sources.list** comprises file attributes, telling you who may use the file. Other entries in the listing have a **d** at the beginning, indicating they are directories. You'll also see hardware devices have a **c** here, for character device — **ls -l** on **/dev/input**, for example. On Linux, everything is a file, even your mouse! A dash (-) at the start tells us this is just a regular file; it's the remaining nine characters that cover permissions.

Every file has an owner and a group membership. Files in your home directory belong to you. If you're logged in as user **pi** and **ls ~ -l**, you'll see **pi pi** in each listing, telling you the owner and the group. Note that we put the switch at the end this time: that's a bad habit under certain circumstances, but we're just showing you what's possible. Owner and group aren't always the same, as **ls -l /dev** will show you.

## File attributes

The file attributes, after the file type, are three groups of three characters (rwx) telling you which users may read, write or execute the file or directory for, respectively, the user who owns the file, the group

### [ FREE TO USE ]

Software in the Raspbian repository is not just free to use, but freely modifiable and redistributable. Free software, like Raspbian's Debian base, is built on sharing: for education and for building community.

[ PROBLEMS? ]

Fine-grained permissions make for greater security, but can trip you up. Typing **sudo** in front of a command that doesn't work is both a diagnosis and a quick workaround of a permissions problem.

owner, and everyone else ('others'). Execute permissions are needed to run a file if it's a program – such as **launcher.sh** which runs the Python games in your **usr/share/python\_games** folder, and thus it has the **x** – and for directories, so that you may **cd** into them.

**cd** into **usr/share/python\_games** and then enter the command **sudo chmod a-x launcher.sh** – the **a** stands for all (user, group and others), use **u**, **g**, or **o** to just change one. Try opening Python Games from the main menu and it won't work. We could restore normal running with **sudo chmod a+x launcher.sh**, but instead we'll use: **sudo chmod 755 launcher.sh**.

### Octal version

Those numbers are an octal representation of user, group, and others' permissions: in each case, read is represented by 4, write by 2, and execute by 1, all added together. So here we have 7s for read+write+execute for user, and 5 for read+execute for group and all other users. **ls -l** and you'll see we're back to **-rwxr-xr-x**.

You can use **chown** to change who owns a file and **chgrp** to change which group it belongs to. Make a new text file and **sudo chown root myfile.txt** – now try editing it and you'll find that while you can read the file, you can no longer write to it. You can also make a file that you can write to and run, but not read!

In the next chapter, we'll be doing useful things with the output of our commands; before moving on, though, why not try your hand at **robots** from the **bsdgames** package we installed?

The **id** command shows what group access you have, for permission to use and alter files and devices

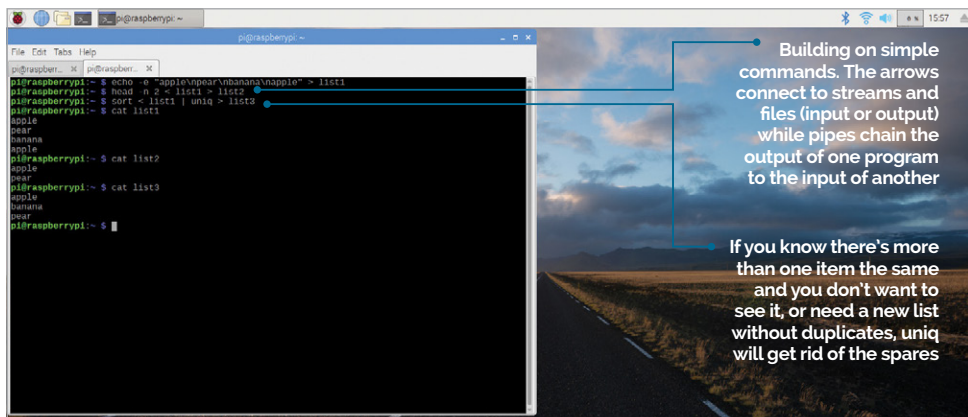
```

pi@raspberrypi ~
File Edit Tabs Help
pi@raspberrypi ~
pi@raspberrypi ~
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuid:x:100:101::/var/lib/libuid:/bin/sh
pi:x:1000:1000,,,:/home/pi:/bin/bash
sshd:x:101:65534:/var/run/ssh:/usr/sbin/nologin
ntp:x:102:104::/home/ntp:/bin/false
stard:x:103:65534:/var/lib/stard:/bin/false
messagebus:x:104:106::/var/run/dbus:/bin/false
usbmux:x:105:46:usbmux daemon,,:/home/usbmux:/bin/false
lightdm:x:106:109:Light Display Manager:/var/lib/lightdm:/bin/false
pi@raspberrypi ~ $
pi@raspberrypi ~ $ whoami
pi
pi@raspberrypi ~ $
pi@raspberrypi ~ $ id
uid=1000(pi) gid=1000(pi) groups=1000(pi),4(adm),20(dialout),24(cdrom),27(sudo),32(audio),44(video),46(usbdev),60(games),100(users),105(netdev),999(input)

```

# [ CHAPTER FOUR ] MANIPULATING TEXT

Discover pipes and learn how to connect multiple simple commands together for more powerful text processing



## [ ABSOLUTE PATH ]

We're using `~/mylisting4.txt` with `~` short for `/home/pi`. If you `cd` to `~` then you can simply use the file name without the `~/`

**T**he UNIX family of operating systems, which includes other flavours of GNU/Linux and also Apple's macOS, deals with data from commands as streams of text. This means that commands can be chained together in countless useful ways. For now, though, we'll focus on giving you a firm foundation to building your own custom commands.

## Getting our feet wet

When a command is called at the terminal, it is given three streams, known as standard input (stdin), standard output (stdout), and standard error (stderr). These streams are plain text, and treated by the Pi as special files. As we noted in chapter 3, 'everything is a file': this is what gives the Pi and other UNIX family systems the ability to put together simple commands and programs to build complex but reliable systems.

Normally, stdin is what you enter into the terminal, while stdout (command output) and stderr (any error messages) appear together. The reason the last two have a separate existence is that you may want to redirect one of them – error messages, for example – somewhere away from the regular output your commands produce. We'll look at separate error messages later, but first we need to know how to redirect and connect our output to other commands or files.

Connecting commands together are pipes, the `|` symbol found above the backslash on both GB and US keyboards (although the two

keyboards for English speakers place the `\` respectively to the left of Z, and at the far right of the home row). When you type a command such as `ls -l`, the output is sent by Raspbian to the `stdout` stream, which by default is shown in your terminal. Adding a pipe connects that output to the input (`stdin` stream) of the next command you type. So...

```
ls -l /usr/bin | wc -l
```

...will pass the long listing of the `/usr/bin` directory to the wordcount (`wc`) program which, called with the `-l` (line) option, will tell you how many lines of output `ls` has. In other words, it's a way of counting how many files and folders are in a particular directory.

## Search with grep

One of the most useful commands to pass output to is `grep`, which searches for words (or 'regular expressions', which are powerful search patterns understood by a number of commands and languages), like so:

```
grep if /usr/share/python_games/catanimation.py
```

This displays every line in the `catanimation.py` file containing the character sequence 'if' (**Fig 1**, overleaf) – in other words not just the word 'if', but words like 'elif' (Python's `else if`), and words like 'gift' if they were present. You can use regular expressions to just find lines with 'if', or lines beginning with 'if', for example.

Piping search results and listings to `grep` is the way we find a needle in one of Pi's haystacks. Remember `dpkg` from the last chapter, to see what was installed? Try...

```
dpkg -l | grep -i game
```

...to remind yourself which games you've installed (or are already installed). The `-i` switch makes the search case insensitive, as the program may be a 'Game' or 'game' in the description column. A simple `dpkg -l | more` lets you see output a page at a time.

`sort` will, as the name suggests, sort a listing into order, with various tweaks available such as `-f` to bring upper and lower case together.

## [ REGEXP ]

Regular expressions (regexp) are special characters used in text searches, such as `[a-z]` to match any letter (but not numbers), and `^` to match to the beginning of a line.

## [ FILING HOMEWORK ]

There are many more commands beyond `grep`, `sort` and `uniq` that can be chained together. Take a look at `cut` if you're feeling adventurous.

**Fig 1** No matter how long the file, `grep` will dig out the lines you need. It's also handy for finding the results you want from a multi-page output

```

pi@raspberrypi ~ $ grep if ~/python_games/catanimation.py
if direction == 'right':
    if catx == 280:
elif direction == 'down':
    if caty == 220:
elif direction == 'left':
    if catx == 10:
elif direction == 'up':
    if caty == 10:
if event.type == QUIT:
pi@raspberrypi ~ $
  
```

One way to collect unsorted data is to combine lists. `sort` will put the combined listing back in alphabetical order:

```
ls ~ /usr/share/python_games | sort -f
```

Suppose you copied one of the games to your home directory to modify: you know it's there, but you don't want to see the same name twice in the listings. `uniq` will omit the duplicated lines or, with the `-d` switch, show only those duplicates.

```
ls ~ /usr/share/python_games | sort -f | uniq
```

## File it away

Pipes are not the only form of redirection. `>` (the 'greater than' symbol) sends the output of a program into a text file, either creating that text file in the process, or writing over the contents of an existing one.

```
ls /usr/bin > ~/mylisting4.txt
```

Now look in `mylisting4.txt` and you'll see the output of `ls /usr/bin`. Note that each item is on a separate line (**Fig 2**). Your terminal displays multiple listings per line for space efficiency; however, for easy compatibility between commands, one listing per line is used. Most commands operate on lines of text; e.g., `grep` showed you in which lines it found 'if'. Note that some commands need a dash as a placeholder for the stdin stream being piped to them:

```
echo "zzzz is not a real program here" | cat mylisting4.txt -
```



## Appending

If you want to add something to the end of a file without overwriting the contents, you need `>>`.

```
echo "& one more for luck!" >> ~/mylisting4.txt
```

`echo` simply displays whatever is in the quote marks to stdout; the `-e` switch lets you add in special characters, like `\n` for newline (see below). You can look at the last few lines of a file with `tail ~/mylisting4.txt`. `<` will link a program's input stream to the contents of a file or stream. Make an unsorted list to work on, and sort it:

```
echo -e "aardvark\nplatypus\njellyfish\naardvark" >
list1
sort < list1
```

You can also combine `<` and `>`:

```
head -n 2 < list1 > list2
```

...will read from `list1`, passing it to `head` to take the first two lines, then putting these in a file called `list2`. Add in a pipe:

```
sort < list1 | uniq > list3
```

Lastly, let's separate that stderr stream: it has file descriptor 2 (don't worry too much about this), and `2>` sends the error messages to any file you choose:

```
cat list1 list2 list3
list4 2>errors.txt
```

The screen will display the 'list' files you do have, and the 'No such file or directory' message(s) will end up in `errors.txt` – `2>>` will append the messages to the file without overwriting previous contents.

```

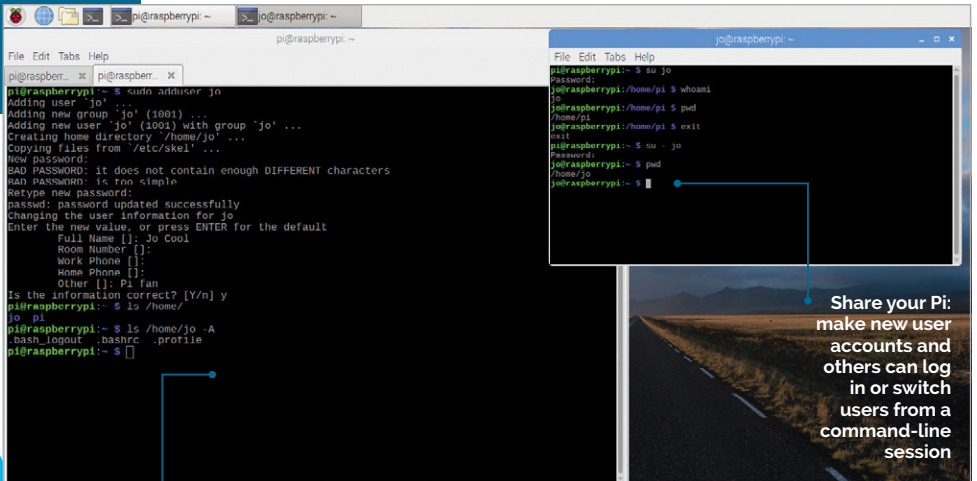
pi@raspberrypi ~
File Edit Tabs Help
pi@raspberrypi ~ pi@raspberrypi ~
pi@raspberrypi ~ $ ls /usr/bin/ > mylisting1.txt
pi@raspberrypi ~ $ more mylisting1.txt
2to3
2to3-2.7
2to3-3.2
flitoppa
adp
bconnect
addpart
addr2line
alsa_in
alsaLoop
alsalixer

```

**Fig 2** With redirection, you can get all of the output from a command saved straight into a text file. Save your error messages to ask about them on the forums!

# [ CHAPTER FIVE ] CUSTOMISE THE COMMAND LINE

We make Raspbian a little more personal as we get it to behave and look just the way we want it to



The command-line environment is personal to each user. You can change your identity with or without a change of environment, depending upon what you need to do in another role

**T**ake a look at that blinking cursor on your terminal, and at what's behind it: **pi@raspberrypi ~ \$**

The \$ is known as the 'dollar prompt', awaiting your command; before it you see the ~ (tilde), shorthand for 'home' – which is **/home/pi** in this case. Before that is [user name]@[computer name], in the form **pi@raspberrypi**. Not only is this informative (at least if you've forgotten who and where you are), but it's also something you can change and personalise.

## New user

Let's start with that user name: pi. If more than one person in your family uses the Pi, you may want to keep the pi user for shared projects, but set up individual login accounts for family members, including yourself. Creating a new user in Raspbian is easy: **sudo adduser jo** will create a new user account named **jo**. You will be prompted for a password (pick a good one) and lots of irrelevant info (dating back to shared university computers of the 1970s) that you can safely ignore by just pressing **ENTER** at each prompt. Now we have a user account for jo, have a look at **/home/jo**. Does it look empty? Use **ls -A**. Jo has never logged into the computer, so you will see the absence of most of the contents of **/home/pi** for now, but there is a **.bashrc** and a couple of other config files.

```

pi@raspberrypi:~$ env
IFS=() mp3=00;36:* mp3=00;36:* ogg=00;36:* ra=00;36:* wav=00;36:* axa=00;36:* oga
00;36:* ssp=00;36:* xsp=00;36:*
SSH_AUTH_SOCK=/tmp/ssh-4fUr/etGADf0/agent.2574
PATH=/home/pi/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/
usr/local/games:/usr/games
DESKTOP_SESSION=LXDE-pi
MAIL=/var/mail/pi
PWD=/home/pi
LANG=en_GB.UTF-8
LXSESSION_PID=2674
SHLVL=2
HOME=/home/pi
XDG_CONFIG_HOME=/home/pi/.config
LOGNAME=pi
XDG_DATA_DIRS=/usr/local/share/:/usr/share/ra:pi-overrides:/usr/share/:/usr/
share/gdm/:/var/lib/menu-xdg/
DBUS_SESSION_BUS_ADDRESS=unix:abstract=/tmp/dbus-eEqnZWhSY1,guid=8e5bb6c8f994d69
3282960f855781fe5
WINDOWPATH=7
DISPLAY=:0.0
XDG_CURRENT_DESKTOP=LXDE
XAUTHORITY=/home/pi/.Xauthority
_=usr/bin/env
pi@raspberrypi ~$ env

```

Above Bash stores information, from your previous 'present working directory' to who you are, in environment variables like **OLDPWD** and **USER**. See individual variables with e.g. **echo \$USER**, or view them all with **env**

[ HOME RUN ]  
If you're logged in as user pi, then **~** is a shortcut to **/home/pi** – but **ls ~jo** can be used as a shortcut to list **/home/jo**, substituting any other user name as desired, with tab completion working after **~j** is typed.

Not every user has a home directory and logs in: enter **cat /etc/passwd** and you'll see a lot of users listed that aren't people. This is because files and programs running on a UNIX-

type system have to belong to a user (and a group – take a look at **/etc/group**), as we saw back in chapter 1 when we did **ls -l**. The user passwords are fortunately not listed in the **/etc/passwd** file in plain text, so if you want to change a password you'll need to use the **passwd** command: **sudo passwd jo** will change the password for user jo. If you're logged in as user pi, then simply calling **passwd** will prompt you to change pi's password.

Transformations in the virtual world are always easier than those in nature, and this is the case with switching from being 'pi' to 'jo': we use the change (or substitute) user command, **su**, like so: **su jo**. After typing this, you should see the prompt change to **jo@raspberrypi**; you can also confirm who you are logged in as with **whoami**.

### Changing identity

**su - jo** (note the dash) is usually preferred, as you'll gain all of jo's specific environment settings, including placing you in **/home/jo**. Note that on many other Linux systems, **su** on its own will enable you to become the root or superuser, with absolute powers (permissions to run, edit, or delete anything). Raspbian (and some other popular GNU/Linux systems like Ubuntu) prefer **sudo** to run individual programs with root permissions. Root's godlike powers may be temporarily attained with **sudo -s** – try it (as user pi) and note how the prompt changes (enter **exit** to exit) – but it's generally a bad idea to run with more permissions than you need! For any user, you can customise elements of their command-line user most simply by

editing `~/bashrc`. Take a look through that configuration file now (as user `jo`): `more ~/bashrc`. Note a number of variables in all capital letters, such as `HISTSIZE` and `PS1`. The last of these controls the prompt you see, currently `jo@raspberrypi ~ $`. To change it (for the duration of your current terminal session), try something like: `export PS1="tutorial@magpi > "`.

This is a temporary change: type `exit` and you've left the `su` value of `jo`, so you'll see `pi@raspberrypi ~ $` once more. If you `su` back to `jo`, the `magpi` prompt will still be gone. To make your change permanent, you need to put the `PS1` value you want into `~/bashrc`. A search around the web will bring up many fancy options for better customising the Bash prompt.

The `~/bashrc` file is read upon each login to a Bash session, or in other words, every time you log into a console or open a terminal. That's unless you change Raspbian's default shell away from Bash,

“ Transformations in the virtual world are always easier than those in nature ”

something you may have reason to do in the future – there are interesting alternatives available for extra features or for smaller memory footprint – but let's not worry about that for now. You can put all sorts of commands in there to personalise your environment: command aliases are great for regularly used combinations.

## Alias

As user `pi`, see what's there with: `grep alias ~/bashrc`. There are a few aliases already in there, particularly for the `ls` command. One entry is: `# alias ll='ls -l'`. This sounds quite useful, although the `#` indicates that it is 'commented out', which means that it will not be read by Bash. Open `.bashrc` in your text editor (double-click the file in File Manager after pressing `CTRL+H` to show hidden files) – the simple Text Editor will do for now as although we've touched on using `nano` for editing text from the command line, we aren't going to go into this

## [ BASIC ACCOUNT ]

`adduser` creates a new user, then takes care of all of the extra details like making a home directory. If all you want is a user created with no extra frills, then the command you want is `useradd`.

[ WHO AM I? ]

From a virtual console (CTRL+ALT+F1 to F6), su and that's who you're logged in as. From an xterm, you can change to someone else, but start another app from the menu and you'll be back to your original login.

in detail until the next chapter. Removing the # will mean that now when you type **ll**, you'll get the action of running **ls -l**. Handy, but we could make it better. Change it to: **alias ll='ls -lAhF'** and you'll get an output in kB or MB, rather than bytes, along with trailing slashes on directory names and the omission of the ever present . and .. (current and parent) directories. Changes take effect after you next start a Bash session, but then you can just run that alias as a command (Fig 1). To disable an alias for a session, use: **unalias ll**.

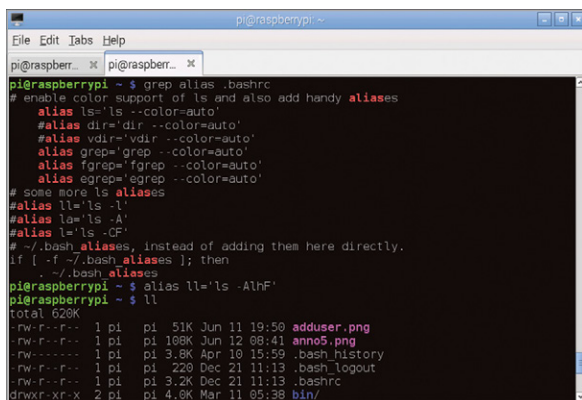
### Key point

We'll end with the very first thing many users need to change: the keyboard map. The system-wide setting is found in **/etc/default/keyboard**, but often you need to change it just for individual users. If £ signs and letters without accents are not sufficient for them, log in as the user who wants a different keyboard, or add **sudo** and the correct path to the commands below. For example, for a Greek keyboard:

```
touch ~/.xsessionrc
echo "setxkbmap el" > ~/.xsessionrc
```

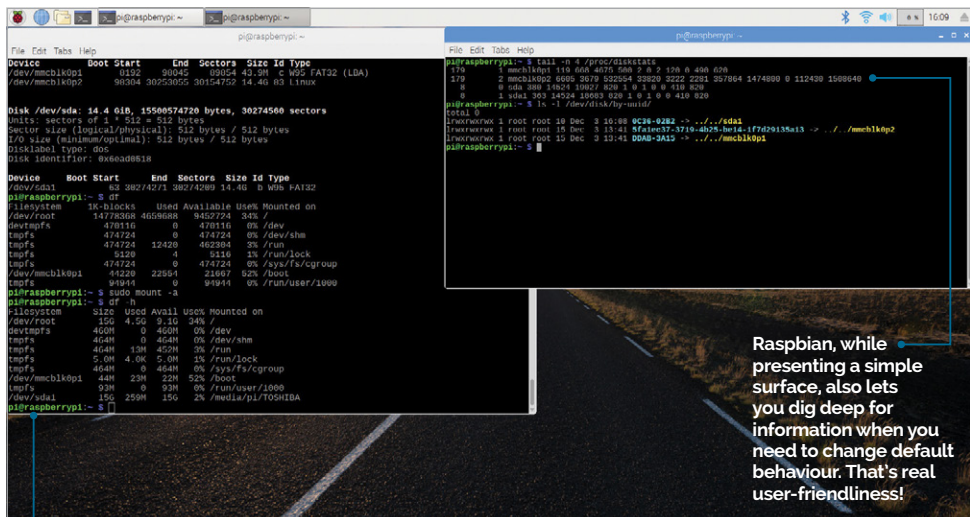
Replace **el** with **pt**, **us**, or whatever language you desire. Note that the config file we created – **.xsessionrc** – holds settings that are read when we log in to the GUI, so the keyboard setting will cover not just the terminal, but every app used in the session.

**Fig 1** Those terse, two- or three-letter commands are not set in stone: make your own shortcuts to keep, or just for use over a specific session



# [ CHAPTER **SIX** ] CONNECTING DISKS

For chapter six, we're tackling the management of removable storage



Even simple utilities have multiple uses: **df**, by showing space available, reminds the user which disks are mounted and can be accessed by the Pi

## A

lthough Raspbian will, when booted as far as the GUI, automatically mount any disk-type device (USB flash key, camera, etc.) plugged into the USB port and offer to open it for you (Fig 1, overleaf), you may wish to get more direct control of the process. Or, as is more often the case, you may want to mount a disk when the Raspberry Pi is running a project that doesn't involve booting as far as the GUI, as it's not necessary for most sensor projects.

## Connected or mounted?

Plugging a drive or flash memory device into your Pi (*connecting* it to your computer) is not the same as making it available for the Pi to interact with (*mounting* it) so that Raspbian knows what's on it and can read, write, and alter files there. It's an odd concept to accept: the computer knows there's a disk plugged in, but its contents remain invisible until the Pi is told to mount it. It's a bit like seeing a book on your shelf, but not being allowed to open or read it.

Disks and disk-like devices are mounted by Raspbian on a virtual file system, and you'll rarely need to worry about what goes on beneath that layer of abstraction, but to see some of it, type **mount**. The information displayed is of the form *device on mount point, file-system type, options*. You'll see lots of device 'none' for parts of the virtual

### [ IN DEPTH ]

If you want to delve deeper into what goes on inside Raspbian and other GNU/Linux systems, try Brian Ward's excellent book, *How LinuxWorks* ([magpi.cc/ZEhaBF](http://magpi.cc/ZEhaBF)).



system that you don't need to worry about; the devices that concern us start with `/dev/` and have names like `/dev/mmcblk0p1` for partitions of the Pi's SD card, and `/dev/sda1` for plugged-in USB drives.

Plug in a USB drive (a flash drive should work fine, but some hard drives may require a separate power supply). Like most computer desktops, Raspbian automatically mounts the disks, so (unless you boot to a virtual console) you'll need to unmount it. `mount` will show an entry beginning something like `/dev/sda1 on /media/pi/UNTITLED...` and you can unmount with `sudo umount /dev/sda1` (yes, that is unmount without an 'n'). An error will result if the device is in use, so change directory and/or close apps using files from the device. Now we can mount it just the way we want.

## Finding the disk

The `/dev/sda1` refers to the first, or only, partition (a section of the hard drive that is separated from other segments) on `/dev/sda`. The next device plugged in will be `/dev/sdb1`. You can see what's being assigned by running `tail -f /var/log/messages`, then plugging in the USB device. On other Linux systems, if `/var/log/messages` draws a blank,

### [ DISK & DISK SPACE ]

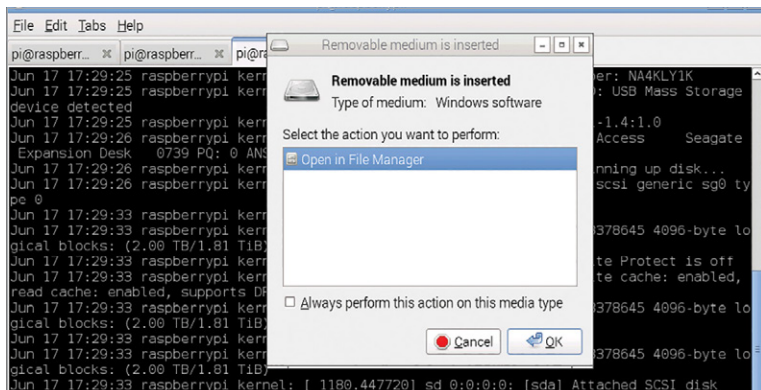
The `df` command shows you space on mounted drives: just type `df` and you'll also get a list of connected drives. It's more readable than `mount -l`, though lacking file type info. It's also quicker to type!

“ An error will result if the device is in use, so change directory and/or close apps ”

try `/var/log/syslog`. Stop the tail with `CTRL+C`. Another way of seeing connected devices that aren't necessarily mounted is `fdisk`, a low-level tool used to divide disks up into partitions, before creating file systems on those disks (see the 'Format' box on page 35). Called with the list option, `sudo fdisk -l`, it performs no partitioning, but simply lists partitions on those disks connected to your Pi. It also gives file-system information, which you need in order to mount the disk. Lastly, you need a mount point (somewhere to place the device on the file-system hierarchy) with appropriate permissions. Create one with:

```
sudo mkdir /media/usb
sudo chmod 775 /media/usb
```

**Fig 1** Raspbian wants to mount plugged-in disks, and take care of the details for you – note that the GUI tells you it's 'Windows software' – while the command line beneath has information for you to take control when you need the job done in a particular way, telling you it's an NTFS file system



Mount the disk with `sudo mount -t vfat /dev/sda1 /media/usb`, where vfat (or NTFS or ext2) is the file-system type.

## File-system table

Raspbian knows which disks to mount at boot time by reading the file-system table (`/etc/fstab`), and we could put our `/dev/sda1` in there, but if we start up with two drives plugged in, the wrong one may be selected. Fortunately, disks (or rather, disk partitions) have unique labels known as UUIDs randomly allocated when the partition is created. Find them with `sudo blkid`, which also helpfully tells you the label, if any, that often contains the make and model of external drives, or look in `/dev/disk/by-uuid`.

For an NTFS-formatted drive, we called `sudo nano /etc/fstab` and added the following line to the end of the file:

```
/dev/disk/by-uuid/E4EE32B4EE327EBC /media/usb
ntfs defaults 0 0
```

This gives the device name (yours will be different, of course), mount point, file-system type, options, and two numeric fields: the first of these should be zero (it relates to the unused dump backup program), while the second is the order of check and repair at boot: 1 for the root file system, 2 for other permanently mounted disks for data, and 0 (no check) for all others. `man mount` will tell you about possible options.

## Editing with nano

We touched briefly on nano in chapter 2. Looking in a little more depth, the first thing to be aware of is the dozen shortcuts listed across the bottom two lines of the terminal: each is the **CTRL** key (represented by the caret **^**) held at the same time as a single letter key. For example, **^R** for ReadFile (i.e. *open*), **^O** for WriteOut (in other words, *save*), and **^X** for Exit. Remember those last two for now, and you'll be able to manage nano. However, if you learn more of them, you will really race through your editing tasks.

While nano lacks the power features of Emacs and Vim, its two main command-line code editor rivals, it has useful features such as a powerful Justify (**^J**), which will reassemble a paragraph of line-break strewn text into an unbroken paragraph, or apply line breaks at a fixed character length. This is a legacy of its development for email composition. **^K** cuts the line of text the cursor is on, but it isn't just a delete function: each cut is added to a clipboard. **^U** will paste the entire clipboard at the cursor position: it's great for gathering together useful snippets from a longer text.

Hit **^O** to save `fstab`, and the shortcut listing changes, with many now beginning **M** instead of **^** – this is short for Meta, which means the **ALT** key on your keyboard (once upon a time, some computers had several modifier keys, such as Super and Hyper). One 'hidden' shortcut after **^O** is that at this point, **^T** now opens a file manager to search for the file/directory where you want to save.

After saving, exit nano; now `sudo mount -a` will mount the external drive at the desired mount point (Fig 2), regardless of what else is plugged in. If you have other new entries in `/etc/fstab`, then `sudo mount /media/usb1t` (or whatever entry you put in `fstab`) will mount just that chosen device if you don't want to mount any of the others.

Having got inside connected disks, the next chapter will see us accessing all of the Pi, but remotely, from anywhere on the planet with an internet connection.

```

pi@raspberrypi ~ $ sudo nano /etc/fstab
pi@raspberrypi ~ $ ls /mnt/
pi@raspberrypi ~ $ sudo mkdir /mnt/1t
pi@raspberrypi ~ $ df
Filesystem            1K-blocks    Used Available Use% Mounted on
rootfs                3023728    2706624    143792  95% /
/dev/root             3023728    2706624    143792  95% /
devtmpfs              219744         0    219744   0% /dev
tmpfs                 44784      256    44528   1% /run
tmpfs                 5120         0     5120   0% /run/lock
tmpfs                 89560         0    89560   0% /run/shm
/dev/mmcblk0p1        57288      9920     47368  18% /boot
pi@raspberrypi ~ $ sudo mount -a
pi@raspberrypi ~ $ df
Filesystem            1K-blocks    Used Available Use% Mounted on
rootfs                3023728    2706624    143792  95% /
/dev/root             3023728    2706624    143792  95% /
devtmpfs              219744         0    219744   0% /dev
tmpfs                 44784      256    44528   1% /run
tmpfs                 5120         0     5120   0% /run/lock
tmpfs                 89560         0    89560   0% /run/shm
/dev/mmcblk0p1        57288      9920     47368  18% /boot
/dev/sda1             1953497884 262723236 1698773848 14% /mnt/1t
pi@raspberrypi ~ $

```

**Fig 2** Once we've put our removable disk in the file-system table (`/etc/fstab`), `mount -a` will read the config from there to mount your disks, saving you from having to remember the details

## [ FORMAT ]

Copying a disk image negates the need to format the disk. Should you need to format a new partition, or convert a disk to ext4 format, read the manual: `man mkfs` and for individual file-system types such as `man mkfs.ext4`

# [ CHAPTER SEVEN ] PREDICTABLE NETWORKING

In this chapter, we give the Raspberry Pi a permanent network address of its own

**R**aspbian takes care of automatically connecting in most situations, but sometimes you need to override automatic configurations to ensure a consistent network setting for your Raspberry Pi project: Raspbian has the tools, and we'll show you the essentials you need to stay connected.

Plug an Ethernet cable from your ADSL router / modem to your Raspberry Pi (or connect via wireless LAN) and, automatically, Raspbian knows where it is on the network, and can talk to the outside world.

All of this is thanks to DHCP – Dynamic Host Configuration Protocol – which provides network configuration for every device connected into a network. Typically, this comes in the form of an IPv4 (Internet Protocol version 4) address, a pair of four numbers separated by a period. For example: 192.168.0.37

The first two sets of numbers, 192.168, mark the start of a private range. These are the numbers for all devices in your house, ranging from 192.168.0.0 to 192.168.255.255.

Check your Raspberry Pi's current connection with the **ifconfig** command. This should show, amongst others, a line like inet 192.168.0.37 (with your own numbers). This will be below the etho

The IP address is likely to be a private address in the range beginning 192.168.0.0. Here we can see it beneath etho (because our Raspberry Pi is connected via an Ethernet cable). Our address is 192.168.0.37

The ifconfig command tells you information about your Raspberry Pi's network address

```

pi@raspberrypi:~$ ifconfig
etho: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.0.37 netmask=255.255.255.0 broadcast=192.168.0.255
    inet6 fe80::1dcd:180d:2bae:4d62 prefixlen 64 scopeid 0x2<link>
    ether b8:27:eb:15:36:d1 txqueuelen 1000 (Ethernet)
    RX packets 47223 bytes 12171777 (11.0 MiB)
    RX errors 0 dropped 1571 overruns 0 frame 0
    TX packets 29968 bytes 1762889 (16.0 MiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 245 bytes 19060 (18.6 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 245 bytes 19060 (18.6 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

wlan0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    ether b8:27:eb:40:63:84 txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

pi@raspberrypi:~$ AC
pi@raspberrypi:~$
  
```

section if you are connected via Ethernet, or under the wlan0 section if you're connected wire wireless LAN.

A faster way to get your IP address is to enter `hostname -I` on the command line.

## How to set up your Raspberry Pi to have a static IP address

Usually when you connect a Raspberry Pi to a local area network (LAN), it is automatically assigned an IP address. Typically, this address will change each time you connect.

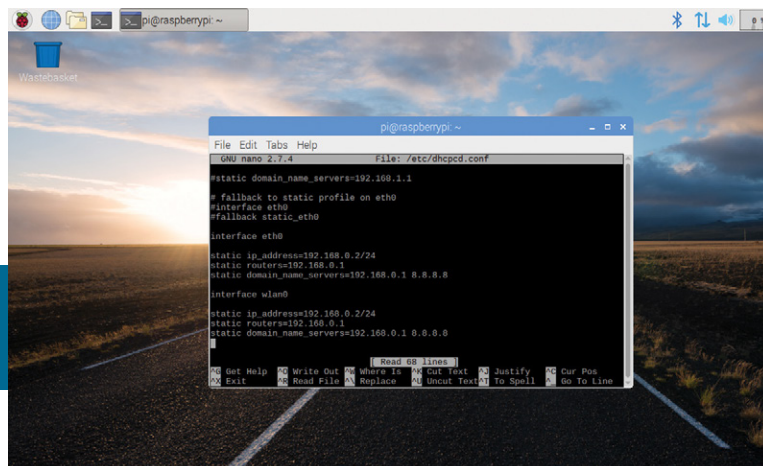
“ You might want your Pi to boot up with the same IP address each time ”

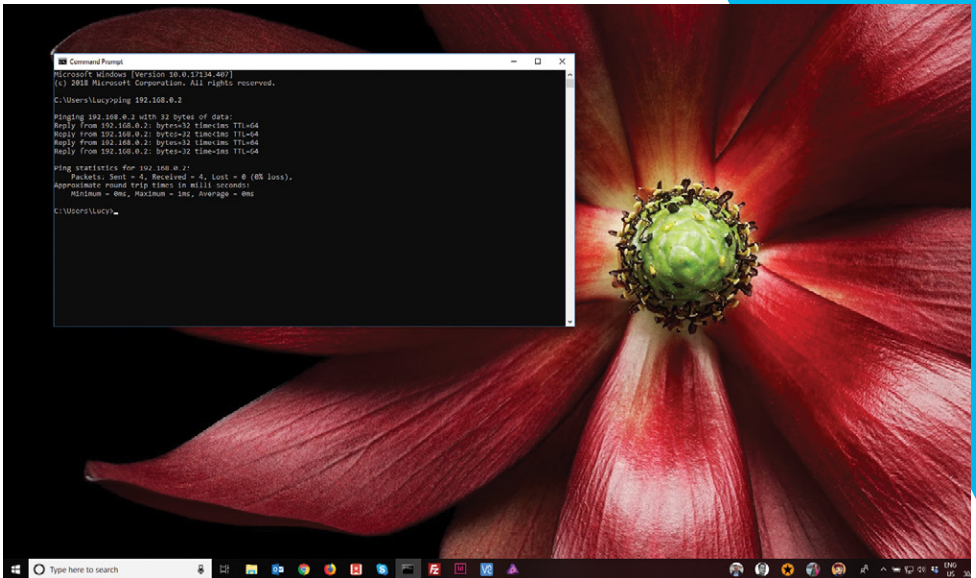
Sometimes, however, you might want your Pi to boot up with the same IP address each time. This can be useful if you are making a small self-contained network, or building a standalone project such as a robot. Here's how to do it.

Edit the file `/etc/dhcpd.conf` as follows (Fig 1):

Type `sudo nano /etc/dhcpd.conf` at the command prompt. Scroll to the bottom of the script, and add the following lines:

**Fig 1** Edit the `/etc/dhcpd.conf` file to determine which static IP address to use with a Raspberry Pi





**Fig 2** Use ping from another computer to detect if your Raspberry Pi is responding to network requests

```

interface eth0
static ip_address=192.168.0.2/24
static routers=192.168.0.1
static domain_name_servers=192.168.0.1 8.8.8.8

interface wlan0
static ip_address=192.168.0.2/24
static routers=192.168.0.1
static domain_name_servers=192.168.0.1 8.8.8.8

```

Save the file with **CTRL+O** and then exit nano with **CTRL+X**.

Your Raspberry Pi will now boot up with the IP address 192.168.0.2 every time; we didn't use 192.168.0.1 as this is reserved for the router. You can of course use any address you like, but in the configuration above, the range must be between 192.168.0.2 and 192.168.0.255.

Reboot with **sudo shutdown -r now**. Log in and type **hostname -I**. You should then see the IP address you set in the eth0 or wlan0 entry (192.168.0.2).

Normally you don't want your computer set to use a static IP address. You can change the network configuration back by editing `dhcpcd.conf` again using `sudo nano /etc/dhcpcd.conf`. Remove all the lines you added in the previous step, then save the file again.

## Ping!

Ping is the most basic tool in the network testing armoury, but one which is often called upon. It sends an ICMP (Internet Control Message Protocol) ECHO\_REQUEST to a device on the network. ICMP is built into every connected device and used for diagnostics and error messages: a ping will produce a reply from the pinged machine, which tells you it is on, and connected, and that the network is working between you and it. Information about packets lost, and time taken, also helps with fault diagnosis.

A successful `ping localhost` from the Raspberry Pi tells you not just that the local loopback interface is working, but that localhost resolves to 127.0.0.1, the local loopback address. Name resolution is the cause of many computing problems – see 'Domain Name Servers' box. Now ping the Pi from another machine on your local network: `ping 192.168.0.2` (Fig 2) – you'll need to use the static IPv4 address you set, rather than ours, of course. If you're doing this from a Windows machine, ping defaults to five attempts; from another UNIX machine (another Pi, a Mac, or Ubuntu or other GNU/Linux), it will carry on until you stop it with **CTRL+C**, unless you set a number of ECHO\_REQUEST sends with `-c` like so:

```
ping -c 5 raspberrypi.org
```

## IPv6

The four-digit IP address style we use (such as 192.168.0.2) is IPv4. A newer standard, IPv6, is becoming more common. These are longer 128-bit addresses represented in hexadecimal (for example, fd51:42f8:caae:d92e::1). Look at the example code in `dhcpcd.conf` for setting up a static address with IPv6.

## Free / public DNS

As well as dynamic DNS providers, some of those listed at **FreeDNS.com** offer public DNS servers. For a wider listing of alternatives to Google's DNS servers, have a search on Google itself.

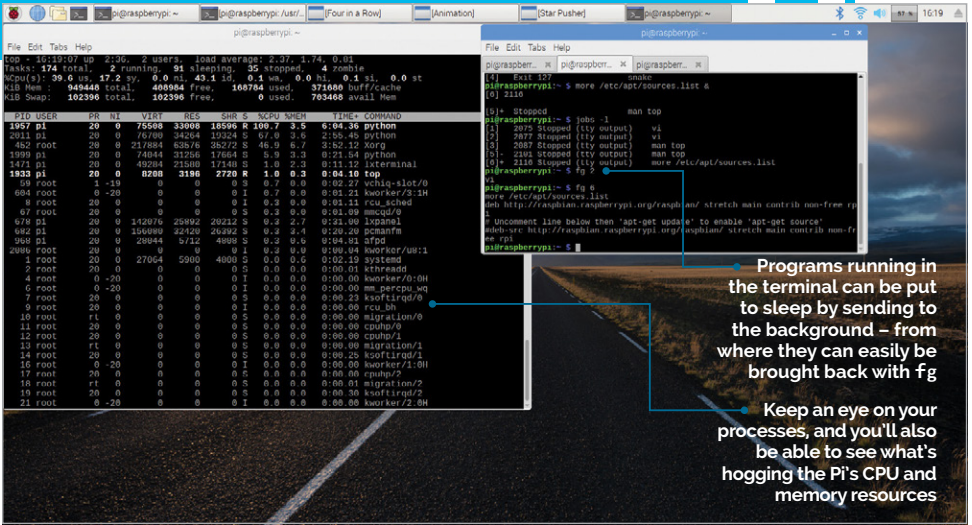
### [ DOMAIN NAME SERVERS ]

We added `192.168.0.1`  
`8.8.8.8` to  
our `domain_`  
`name_servers`  
line in `/etc/`  
`dhcpcd.conf`.  
This is for the  
Google public  
domain name  
server (DNS). The  
DNS maps public  
IP addresses like  
`raspberrypi.org`  
to IP addresses  
(in this case  
`93.93.128.230`).  
You'll need  
to add a DNS  
reference to  
access websites  
in a browser.



# [ CHAPTER EIGHT ] STOPPING A PROCESS

As close to perfect as Raspbian is, things can go wrong . In this chapter, we learn that there's no need to turn the Raspberry Pi off and on again: just kill the process!



Ever lost the 'off switch' for a program? Sometimes a piece of software you're running seems to have no inclination to stop: either you cannot find how to quit, or the app has a problem, and won't respond to your **Q**, **CTRL+C**, or whatever command should close it down.

There's no need to panic, and certainly no need to reboot: just identify the process and quietly kill it. We'll show you how, and look at what else can be done with knowledge of processes.

## Processes

Find the many processes running on your Pi with the **ps** command. As a minimum, on Raspbian, it's usually called with the **a** and **x** switches – which together give all processes, rather than just those belonging to a user, and attached to a tty – and with **u** to see processes by user; **w** adds wider output, and **ww** will wrap over the line end to display information without truncating.

Type **ps auxww** to see, then try with just **a** or other combinations. You will notice that these options work without the leading dash seen for other commands. Both the lack of dashes, and the particular letters, **a** and **x**, date back to the original UNIX **ps** of the early 1970s, maintained through various revisions by one of UNIX's two family

branches, BSD, and baked into the first GNU/Linux `ps`. UNIX's other branch, System V, had extended and changed `ps` with new options and new abbreviations for command switches, so for `ps ax` you may see elsewhere `ps -e` (or `-ef` or `-ely` to show in long format).

The `ps aux` listing has various headers, including the USER which owns the process, and the PID, or Process IDentification number. This starts with 1 for `init`, the parent process of everything that happens in userspace after the Linux kernel starts up when you switch the Pi on.

Knowing the PID makes it easy to kill a process, should that be the easiest way of shutting it down. For example, to kill a program with a PID of 3012, simply enter `kill 3012`, and to quickly find the process in the first place, use `grep` on the `ps` list. For example, locating `vi` processes:

```
ps aux | grep -i vi
```

The `-i` (ignore case) isn't usually necessary, but occasionally a program may break convention and contain upper-case letters in its file name. You can also use `killall` to kill by program name: `killall firefox`

## Piping commands

Naturally, you can pipe `ps`'s output to select the PID and feed directly to the `kill` command:

```
kill $(ps aux | grep '[f]irefox' | awk '{print $2}')
```

We don't have space for an in-depth look at `awk` (we're using it here to print the second field of `grep`'s output: the PID), but the `[f]` trick at the beginning of `Firefox` (or whatever named process you want to kill) prevents the `grep` process itself being listed in the results; in the `vi` example above, `grep` found the `grep` process itself as well as `vi` (and anything with the letter sequence `vi` in its name).

The output of `ps` also shows you useful information like percentage of memory and CPU time used, but it's more useful to see these changing in real time. For this, use `top`, which also shows total CPU and memory use in the header lines, the latter in the format that you can also find by issuing the command `free`. For an improved `top`:

### [ KEEP ON TOP ]

When using a virtual console, it can be worth keeping `htop` running so that if there are any problems, you can `CTRL+ALT-FN` there for a quick look for any problems – even if the GUI freezes.

[ QUICKER BOOT ]

The start up process of Raspbian Wheezy is controlled by the traditional SysV init. Raspbian Jessie, like other GNU/Linux distributions, has moved to the faster (but monolithic) SystemD – we touch on some of the differences in chapter 11.

**sudo apt-get install htop**

htop is scrollable, both horizontally and vertically, and allows you to issue commands (such as **k** for kill) to highlighted processes. When you’ve finished, both top and htop are exited with **Q**, although in htop you may care to practise by highlighting the htop process and killing it from there (see **Fig 1**). htop also shows load over the separate cores of the processor if you have a Pi 2 or 3.

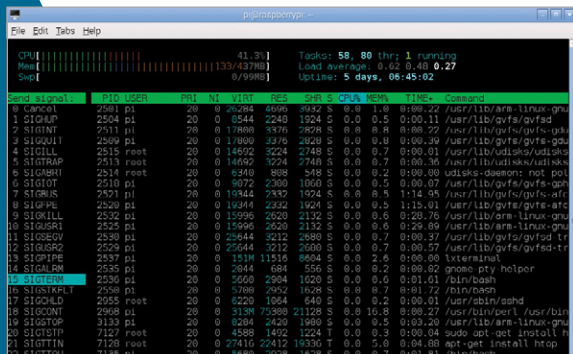
**Background job**

Placing an ampersand (&) after a command in the shell, places the program in the background – try with: **man top &** and you’ll get an output like: **[1] 12768**.

The first number is a job number, assigned by the shell, and the second the PID we’ve been working with above. man top is now running in the background, and you can use the job control number to work with the process in the shell. Start some other processes in the background if you wish (by appending **&**), then bring the first – man top – to the foreground with **fg 1**. Now you should see man running again.

You can place a running shell program in the background by ‘suspending’ it with **CTRL+Z**. fg will always bring back the most recently suspended or backgrounded job, unless a job number is specified. Note that these job numbers apply only within the shell where the process started. Type **jobs** to see background processes; **jobs -l** adds in process IDs (PID) to the listing.

**Fig 1** htop tells you what’s running, what resources it’s using, and lets you interact with the process, even killing htop from within htop



**Signals**

When we send a kill signal from htop, we are given a choice of signal to send. The most important are **SIGTERM**, **SIGINT**, and **SIGKILL**.

The first is also the signal **kill** will send from the command line if not called with a modifier: it tells a process to stop, and most programs will respond by catching

the signal, and first saving any data they need to save and releasing system resources before quitting.

**kill -2** sends SIGINT, which is equivalent to stopping a program from the terminal with **CTRL+C**: you could lose data. Most drastic is **kill -9** to send SIGKILL, telling the kernel to let the process go with no warning. Save this one for when nothing else works.

Mildest of all is the Hang Up (HUP) signal, called with **kill -1**, which many daemons are programmed to treat as a call to simply re-read their configuration files and carry on running. It's certainly the safest signal to send on a critical machine.

## Staying on

**nohup** will run a program which will continue after the terminal from which it is started has closed, ignoring the consequent SIGHUP (hangup) signal. As the process is detached from the terminal, error messages and output are sent to the file **nohup.out** in whichever directory you were in when you started the process. You can redirect it – as we did in chapter 4 – with **1>** for stdout and **2>** for stderr; **&** is a special case for redirecting both stdout and stderr:

```
nohup myprog &>backgroundoutput.txt &
```

One use of **nohup** is to be able to set something in motion from a SSH session, which will continue after an interruption. For example, restarting the network connection to which you are connected:

```
sudo nohup sh -c "ifconfig wlan0 down && ifconfig wlan0 up"
```

Note that the **nohup.out** log file created here will need sudo privileges to read – or reassign with:

```
sudo chown pi:pi nohup.out
```

```

File Edit Tabs Help
root 2474 0.0 0.7 80968 3290 S 3:21 0:00 /usr/lib/polkit/polkitd --read
root 2482 0.0 0.0 12968 1296 S 3:21 0:00 /usr/lib/transition/ty-LDSE.pl -e 0
root 2484 0.0 0.1 3376 712 Tt1 c 3:21 0:00 /usr/bin/dmcc-daemon -exit-with-se
root 2490 0.0 0.2 3312 1072 S 3:21 0:00 /usr/bin/dmcc-daemon -fork_gri
root 2492 0.0 1.7 35968 7660 Tt1 S 3:21 0:00 sshd: sshd [pid=116,nowid=0,co
root 2494 0.2 2.0 80800 32592 Tt1 S 3:21 21:31 lsopen1 -profile LDCE.pl
root 2498 0.0 2.0 81052 14400 Tt1 S 3:21 0:00 pcamr --swatop --profile libfai
root 2501 0.0 1.6 32284 4624 Tt1 S 3:21 0:00 /usr/lib/ana-libra_gnuconfr/libco
root 2504 0.0 0.5 8514 2248 S 3:21 0:00 /usr/lib/gfs/gfsd
root 2509 0.0 0.7 47600 2316 S 3:21 0:00 /usr/lib/gfs/gfs-ql-volume-sonit
root 2513 0.0 0.7 14662 3224 S 3:21 0:00 /usr/lib/udisks/udisks-daemon
root 2514 0.0 0.1 6348 898 S 3:21 0:00 udisks-daemon: not polling any dev
root 2518 0.0 0.0 49172 2580 S 3:21 0:00 /usr/lib/gfs/gfs-ql-volume-sonit
root 2520 0.0 0.5 49344 2332 S 3:21 1:14 /usr/lib/gfs/gfs-qlc-volume-sonit
root 2522 0.0 0.8 49396 2620 Tt1 S 3:21 0:00 /usr/lib/ana-libra_gnuconfr/libm
root 2526 0.0 0.7 35644 3212 Tt1 S 3:21 0:00 /usr/lib/gfs/gfsd -swatop
root 2534 0.0 2.0 154508 11372 Tt1 RL 3:21 0:17 /usr/bin/perl /usr/bin/shutter
root 2536 0.0 0.1 2644 604 Tt1 S 3:21 0:00 gnuconfr-helper
root 2538 0.0 0.0 5560 2294 Tt1 S 3:21 0:01 /bin/bash
root 2550 0.0 0.0 5700 2952 Tt1/2 S+ 3:21 0:01 /bin/bash
root 2650 0.0 0.0 6260 1064 S 3:21 0:00 /usr/bin/nc
root 2960 0.0 0.1 326956 11252 Tt1 S 3:21 2:01 /usr/bin/perl /usr/bin/shutter
root 3138 0.0 0.0 6024 2420 S 3:21 0:00 /usr/lib/ana-libra_gnuconfr/gconfr
root 4646 0.0 0.0 0 0 S 3:12 0:04 [worker/02:1]
root 6297 0.0 0.0 0 0 S 0:11 0:00 [worker/02:2]
root 7122 0.0 0.8 4588 1402 Tt1/2 T 15:53 0:00 sudo apt-get install htop
root 7128 1.5 5.0 27416 22412 Tt1/2 T 15:53 0:04 apt-get install htop
root 7135 0.6 0.0 5680 2268 Tt1/0 S+ 15:53 0:01 /bin/bash
root 7140 0.0 0.2 4448 1298 Tt1/0 S+ 15:54 0:00 Ya
root 7413 0.0 0.0 0 0 S 15:54 0:00 [worker/02:0]
root 7552 0.0 0.0 0 0 S+ 15:55 0:00 [worker/02:1]
root 7556 0.0 0.1 2480 752 Tt1/0 T 15:55 0:00 awk
root 7560 0.0 0.2 4408 1100 Tt1/0 T 15:55 0:00 awk

```

**Fig 2** Everything running has a process ID (PID) that can be used to control that program; find them all with **ps aux**

## [ KEEP ON RUNNING ]

**nohup** is useful for a program that will be running for some time in the background – perhaps a sensor project you are working on – until you feel happy enough to add it to Raspbian's startup processes.

# [ CHAPTER NINE ] REMOTE PI

Learn how to access the Raspberry Pi from remote PCs and devices with Secure Shell (SSH)

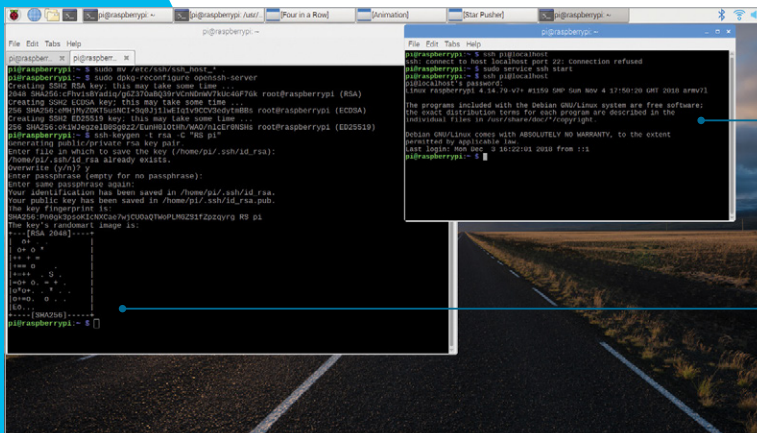
It's great that the Raspberry Pi is so portable, but sometimes you may want to use it without taking it with you. Here, the Pi's default Raspbian OS is a real strength, as UNIX-like operating systems have been used this way for over 40 years.

Over time, as the internet has given the opportunity for malicious users to connect to computers, old standards like Telnet and rlogin have been replaced by Secure Shell (SSH), based on public-key cryptography. Once set up, secure connections are simple, and open to scripted, automatic connection for your projects. Note: you're advised to change your Pi's login password – with **passwd** – before using SSH.

If the SSH server is not enabled by default on your version of Raspbian, run **sudo raspi-config** and go to the advanced settings to enable SSH. Check the IP address assigned to the Pi with **ifconfig** (note the 'inet addr' for the etho or wlan0 interface). Now you can try connecting from another computer on your network.

## Connecting with SSH

From a Mac or GNU/Linux computer, use **ssh** from a terminal to connect to the Pi. Assuming a default setup, and **ifconfig** revealing an IP address of 192.168.0.2, connect with **ssh pi@192.168.0.2** and enter your password. You can use the OpenSSH client on Windows 10 PCs; for earlier PCs, install an SSH client like PuTTY (**magpi.cc/uLyftk**), which also works with SCP, Telnet, and rlogin, and can connect to a serial port. Android users can try the ConnectBot client.



You can test on the Pi if SSH is running, and start the service from the command line – as you can any service (look in **/etc/init.d/** and **/etc/init/** if you're curious about other services)

The Raspbian install image shares its keys with everyone else who has a copy. Generate your own, and personal keys for the user account, for secure remote access

You should now be at the command-line interface of your Pi. If you got any sort of error, check from the Pi that SSH is really up and running by entering `ssh@localhost` on the Pi itself. If that works, SSH is up and running on the Pi, so take a closer look at network settings at both ends.

## Hello, World

Now we can access the Pi on the local network, it's time to share with the world. Step one, for security reasons, change the `PermitRootLogin yes` entry in `/etc/ssh/sshd_config` to read: `PermitRootLogin no` using `sudo nano`. After making any changes to the SSH server's configuration, you must restart the service for them to take effect, or at least reload the configuration file: `sudo service ssh reload`. Note there's also a file in `/etc/ssh/` called `ssh_config`, which is for the SSH client; the `d` in `sshd_config` is short for 'daemon', the UNIX term for a service which runs constantly in the background.

You can also change port 22 to any unlikely number, but be sure to check it still works. You'll need to begin `ssh -p 12123` (or whichever port you have chosen) to tell your client you're not using the default port 22.

To reach your Pi from anywhere on the internet, you need an IP address, which will connect you to your board even though it's behind an ADSL router. Of course, if your Pi is in a data centre, with its own public IP address, you don't need any workaround.

There are numerous services such as [DuckDNS.org](https://DuckDNS.org) providing free-of-charge dynamic DNS (DDNS), to point a constant IP address to the changing one allocated to you by your ISP. However, the largest of these, DynDNS, has ended its free service, which provides a useful reminder that you cannot assume that a free service will be around for ever.

There are several steps to configuring a DDNS setup, no matter which service and software client you choose. Some are detailed in the [raspberrypi.org](https://raspberrypi.org) forums, and there's a good guide to `ddclient` at [samhobbs.co.uk](https://samhobbs.co.uk).

Otherwise, if your broadband router can handle both port forwarding and dynamic DNS, you can set it up to point to port 22 (or a chosen alternate port) on the Pi. You may even find your ISP offers static IP addresses.

## Bye bye FTP

File Transfer Protocol (FTP) was not designed for security: data, and even passwords, are transmitted unencrypted. The Secure Copy

### [ INTERRUPTED SERVICE ]

While you can restart most services with `sudo service ssh restart`, replacing `restart` with `reload` permits configuration changes to be registered with less disruption, which is key for some projects.



Program (SCP), which runs over SSH, is best for transferring files. The syntax of the command mimics the command-line cp program:

```
scp pi@192.168.0.2:/home/pi/testfile1 .
```

Here we're transferring a file from the Pi, across a local network, to the current location (the dot shortcut). Note that you can use wildcards for groups of similarly named files, and can recursively copy directories and their contents with the **-r** switch after **scp**.

## A secure key

If you're trying this on something other than Raspbian, you may not have the SSH server installed. It's often found in a package called **openssh-server**. With Raspbian, you have a pair of keys (public and private) in **/etc/ssh/**. Unfortunately, they'll be the same as those held by everyone else with a copy of the Raspbian image that you downloaded. First, remove the existing keys:

```
sudo rm /etc/ssh/ssh_host_*
```

Alternatively, you can move them somewhere out of the way. Regenerate the system-wide keys with:

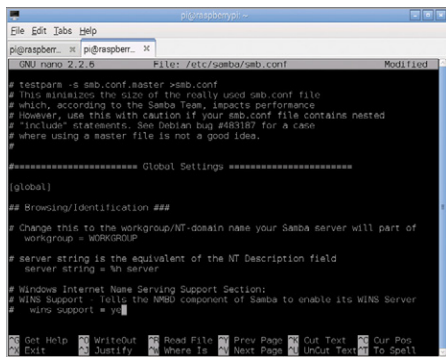
```
sudo dpkg-reconfigure openssh-server
```

For keys personal to you as a user, type **ssh-keygen -t rsa -C "comment"**, where **"comment"** is anything you want to identify the key with: name, email, or machine and project, for example. You'll be asked for a passphrase to secure the key – if you press **ENTER**, you'll get a key with no passphrase, which makes life easier when making scripted (automated) connections, but removes an extra layer of security. You can create keys from any computer with the SSH package, and move the public key to the Pi, but we'll work on the assumption that the Pi is the only handy UNIX-like computer, and we'll be generating the keys there.

If you accepted the defaults, your personal keys will now be in **~/.ssh** with the correct permissions. By default, **sshd** looks in **~/.ssh/authorized\_keys** for public keys, so we need to copy the

### [ SAMBA STEPS ]

Samba is *extremely* well documented, with separate man pages for everything from **smb.conf** to **smbpasswd**, and excellent online books at [samba.org](http://samba.org) – look for **smb.conf** examples.



**Fig 1** There's a lot of configuration in Samba, but simply adding your WORKGROUP name to the default settings should get you up and running

contents of **id\_rsa.pub** to there. The following command will work even if you already have an **authorized\_keys** with contents (make sure you use both **>>** symbols with no gap between them):

```
cd ~/.ssh && cat id_rsa.pub >> authorized_keys
```

Using SCP, copy the private key to **~/.ssh** on your laptop, or wherever you will access the Pi from, removing it from the Pi if it's to act as the

server. Once you confirm SSH works without passwords, you can edit **/etc/ssh/sshd\_config** to include **PasswordAuthentication no**. If you are sticking with passwords, replace 'raspberrypi' with something stronger.

## Shared drive

You may be using a service like Dropbox to share files between machines. There is no need to do this on a local network, as the Samba networking protocol on the Pi lets Windows PCs access it as a shared drive (**Fig 1**, page 48). Samba is already installed in recent versions of Raspbian, or you can install it using:

```
sudo apt-get install samba samba-common-bin
```

Edit **/etc/samba/smb.conf** with a WORKGROUP value (for Windows XP and earlier; try **workgroup = WORKGROUP**) and/or HOME (For Windows 7 and above). Ensure that Samba knows **pi** is a network user:

```
sudo smbpasswd -a pi
```

Then restart with:

```
sudo service samba restart
```

The Pi should now show up in Windows Explorer under Network. You can fine-tune **smb.conf** for what's shared (including printers), and permissions.

### [ LOST KEYS? ]

The private key half of your key pair should be kept secure – but safe, too. Keep a backup of the private key on a memory card in a safe place.

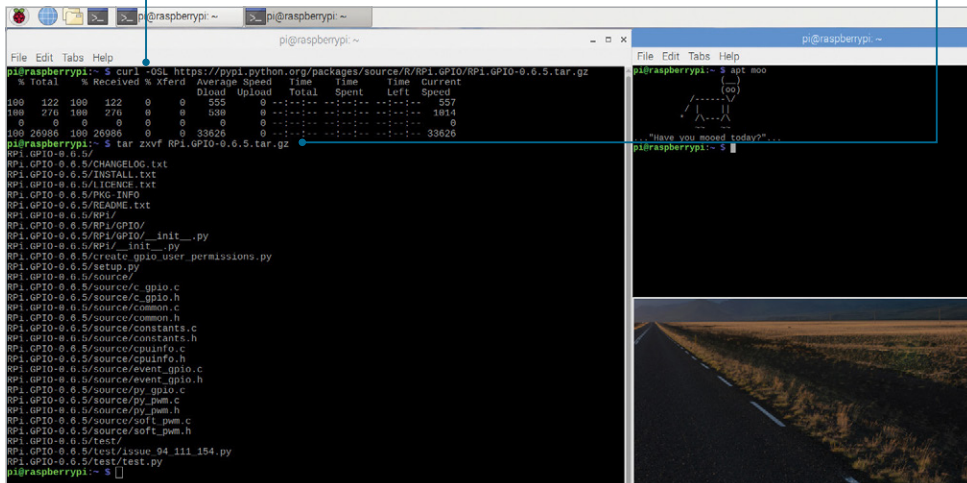
# [ CHAPTER **TEN** ]

# DOWNLOADING & INSTALLING

We look at downloading and unpacking software,  
and show you how to create new Raspbian SD cards

**curl** can be used in place of **wget** for simply downloading files, but its strengths lie elsewhere, in its extensive features – these take in everything from proxy support and user authentication, to FTP upload and cookies

The **tar** command packs or unpacks an archive of files and directories; it also handles uncompressing the download first



**R**unning an **apt** command (see chapter 3) allows access to a huge collection of software – several thousands of packages in the main Raspbian repository – but sometimes we need to add software from outside the main repository.

If we are lucky, we find that someone has packaged up the software in the **.deb** format used by Raspbian, or even created a whole repository to take care of the dependencies. We'll look briefly both at adding repositories, and dealing with other kinds of downloads, trying the venerable **vi** editor along the way.

Information about repositories is kept in the **/etc/apt/sources.list** file, which on a new install just contains the Raspbian repository. Rather than editing this file to add other repositories, you are advised to add them in a **.list** file to the **/etc/apt/sources.list.d** directory. To add a new repository, use **sudo nano** to create a **.list** file there and inside it, add a source in the following format:

```
deb http://apt.pi-top.com/raspbian/ jessie main
```

Here, **jessie** is a Debian (and Raspbian) release name: all Debian releases have been named after characters in the *Toy Story* series of films since 1996 (former Debian project leader Bruce Perens was involved in the early development of Debian while working at Pixar). Stretch followed Jessie in June 2017.

Most software is in the **main** repository, which can be freely copied or mirrored anywhere. Other components, like **non-free**, allow repositories to contain software you may not be free to pass on, keeping it separate from Raspbian's FOSS repository.

## wget & curl

Having added our repository source in a file in **sources.list.d**, we need to get the key for it and use **apt-key** to install it. Packages authenticated using keys added by **apt-key** will be considered trusted.

```
wget -O - -q http://apt.pi-top.com/apt.pi-top.com.gpg.key | sudo apt-key add -
```

Wget downloads from the URL given. The **-O** switch directs the download to stdout, from where it is piped to apt-key (the trailing dash there tells apt-key to read its input from the stdin stream, which is where it receives the output from wget). After any change to the **sources.list.d** directory, you should run:

```
sudo apt-get update
```

This updates Raspbian's knowledge of what's available to install from pi-top's packages – for a full list, enter:

```
grep ^Package /var/lib/apt/lists/apt.pi-top.com_raspbian_dists_jessie_main_binary-armhf_Packages
```

To install one, for example, **sudo apt-get install 3d-slash**.

Wget is a simple but robust download tool, with a powerful recursive feature that helps fetch entire websites, but it does have mild security risks, so be careful using it to fetch scripts. An alternative is curl, a file transfer tool that works with many protocols and can be used for simple

### [ VI IMPROVED ]

If you really want to get to grips with vim, you'll need to **sudo apt-get install vim** – the vim.tiny package already in Raspbian is very limited.

downloads. It dumps to stdout by default; to save as a file with the same name as the resource in the URL, use the **-O** switch. For instance:

```
curl -OSL https://pypi.python.org/packages/source/R/  
RPi.GPIO/RPi.GPIO-0.6.5.tar.gz
```

Here, the **-S** switch will show any errors, while the **-L** switch will enable curl to reattempt to fetch the requested file if the server reports that it has a different location.

## Unzip

The Python GPIO library downloaded above is compressed with gzip, which losslessly reduces the size of files, and can be decompressed with **gunzip**. The contents here are files rolled into a tar archive (instead of **.tar.gz**, you'll sometimes find similar archives ending **.tgz**), and the **tar** command can do the decompression and untarring in one:

```
tar zxvf RPi.GPIO-0.6.5.tar.gz
```

Note that the dash is not needed for single letter options in tar. The first switch, **z**, calls gzip to decompress the archive, then **x** extracts the contents. **v** is verbose, informing you of the process as it happens, and **f** tells tar to work with the named file(s), rather than stdin. Miss out the **z** and tar should automatically detect the necessary compression operation.

The result in this case is a folder containing, among other things, a setup script to run the installation (read the **INSTALL.txt** file first):

```
cd RPi.GPIO-0.6.5  
sudo python setup.py install
```

While gzip is more efficient than zip (and even more efficient options like bzip2 are available), sometimes you'll get a plain old zip file, in which case **unzip** is the command you want.

```
unzip 2018-11-13-raspbian-stretch-lite.zip
```

### [ EASTER EGG ]

Read `man apt`, and you may see: "This APT has Super Cow Powers." If it's there, try typing `apt-get moo` to see what happens.

## Disk image

Having downloaded and unzipped an image for Raspbian, you cannot copy it across to a second microSD card (connected to the Pi via a USB card reader) with regular `cp`, which would simply put a copy of it as a file on the card. We need something to replace the SD card's file system with the file system and contents that exist inside the Raspbian disk image, byte-for-byte, and for this we can use a handy little built-in utility called `dd`.

`dd` converts and copies files – any files, even special devices like `/dev/zero` or `/dev/random` (you can make a file full of zeroes or random noise) – precisely copying blocks of bytes. To copy our Raspbian image,

“ Be very careful that the destination in the command matches the correct disk ”

we will need to unmount the secondary microSD card we've plugged in via a USB card reader. Use `sudo fdisk -l` both before and after plugging in the card (you can also use `df` to see what's mounted) to see attached devices. If, say, a `/dev/sdb` appears, with the size equal to the SD card, then unmount with `umount /dev/sdb1`. Now copy the disk image with:

```
sudo dd if=~/.Downloads/2018-11-13-raspbian-stretch-
lite.img of=/dev/sdb bs=1M
```

Development of Raspbian's ancestor UNIX started in 1969, so we've covered a few utilities with a long heritage in this book, but that `if=` in place of the usual dashes for command-line options indicates a lineage stretching back to the early 1960s, and IBM's Data Definition (DD) statement from the OS/360 Job Control Language (JCL).

Be very careful that the destination matches the correct disk, or you will lose the contents of another storage device! The `bs=1M` is a block size default; `4M` would be another safe option. Now put the card in another Pi and go and have fun!

# [ CHAPTER ELEVEN ]

## START AND STOP AT YOUR COMMAND

We take a look at scripts to manage the way Raspbian starts and shuts down

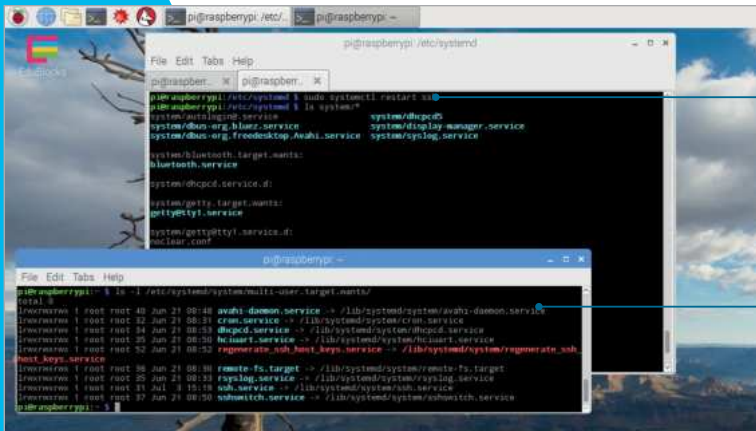


**O**ver the last few years, every major GNU/Linux distribution – Raspbian included – has changed the way that it starts up. This means that much of the older literature on bookshelves and websites, dealing with where to put files in **rc.local** to get them to automatically run at startup, is no longer correct – unless you’ve yet to upgrade from Raspbian Wheezy.

One thing remains the same: although the first process the kernel starts is not **/sbin/init**, but **/lib/systemd/systemd** (still with a PID of 1), it is still the parent process of everything that happens in user space once the Linux kernel has finished initialising devices and drivers, and mounted the file system.

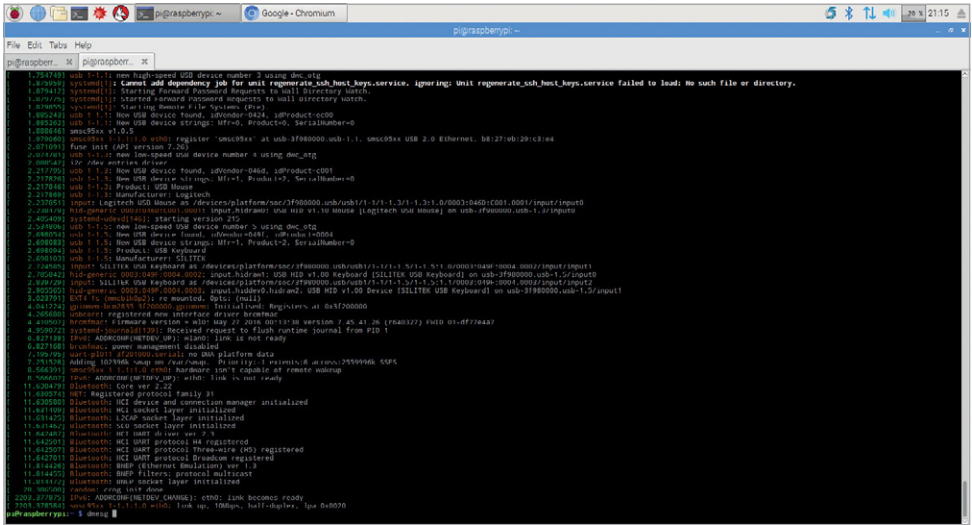
**init** gets everything else started, and usually ends with the prompt inviting you to log in to your Pi. Recent versions of Raspbian hide most of the messages that this startup process generates, but you can see them by typing **dmesg**. They’re also stored in **/var/log/kern.log**.

Startup – **init** – is the start of user space; this is the place where you can put your own programs to affect how the Raspberry Pi runs, without having to modify the code of your Linux kernel! Traditionally, GNU/Linux distributions implemented a version of the UNIX System V **init**, which had a well-defined startup process with run levels that would indicate whether the system was at startup, ‘single user mode’ (rescue mode – a handy way to get back in when you’ve lost your password, or a security headache), console mode, the GUI, or heading for shutdown.



With Raspbian Jessie's move to **systemd**, **systemctl** replaces **service** to restart, or query status, of server software

Adding a service to **systemd** creates a symlink to its real location as part of the process – don't do it manually



**dmesg gives you the startup messages from the kernel – nuggets of useful info buried in plenty of legacy boot information**

Along the way, files in **/etc** with names beginning **rc0** through **rc6** get called – running startup scripts in **/etc/init.d** – and **/etc/rcS.d** which contains files always called at startup, regardless of run level.

Those **/etc/init.d** scripts can be called directly to start or stop your databases, web servers, or anything else that needs intervention. Many support further commands such as **status**, to check a service is running properly, and **reload** – the latter useful if you want a service to take a look at fresh config settings without doing a full restart.

### sudo /etc/init.d/couchbase-server status

Although the regular and predictable scripted startup of Sys-V init makes it easy to place your own programs in the startup process – particularly useful on an unattended Pi – the performance of a purely sequential startup process is poor, even when booting from a solid state disk. Enter systemd...

## Systemd

Systemd (like Upstart – see ‘Refuseniks’ box on p62) can start services in parallel, and can defer service starts until they are needed. Rather than many scripts for individual components, a target is set,

and systemd resolves the dependencies until it reaches that target, avoiding any fixed startup sequence along the way.

Files are found under **/etc/systemd/system** and there's a lot to learn, but as Raspbian now starts up far more quickly, at least we've created extra time for all that learning. The one thing to remember for any user is that systemd and its service manager are controlled with the **systemctl** command.

#### **sudo systemctl restart ssh**

...will restart the SSH server – something you'll need to do if you change the port it listens on, for example. For compatibility, as well as **/etc/init.d** scripts to start and stop services, the system of service commands that worked on older versions of Raspbian, such as

#### **sudo service apache2 reload**

...still works (here we ask a running Apache2 Web server to reread its configuration files).

Under **/etc/systemd/system/multi-user.target.wants** you will find files like **cron.service** which, when examined closely with **ls -l**, you'll see are links to files of the same name in **/lib/systemd/system** (other GNU/Linux distributions may place the files under **/usr/lib**).

Don't worry if what's inside these files looks confusing; there's a logic to them with their conditional dependencies, but you can safely forget about them until you need to get some software working automatically on every system restart for your Pi project. Even then, we'll show you another way with crontab – otherwise you'll need to be aware of the following, as those links aren't created manually:

#### **sudo systemctl enable postgresql.service**

...will create the link, and means PostgreSQL will be enabled upon startup. To control the service before the next restart:

#### **sudo systemctl daemon-reload**

...will make systemd aware of the changes.

#### [ WHO / WHERE ]

Although run levels are no longer particularly meaningful under systemd, you can still check which run level you are in with **who -r**

[ SYSTEMD DOT ]

Systemd is a complex system, with many elements – from `systemd.unit` to `systemd.slice` – but they all have their own man pages, and it's worth reading through to help get the overall concept.

Backwards compatibility with `init.d` scripts – and even run levels – is maintained by `systemd`

## Linked In

Systemd makes links between files automatically, but there will be times you'll want a file to appear to be in a local directory when it is elsewhere, with a handy little command we have not so far had a chance to show you: `ln`.

`ln` makes a link which allows a file to effectively exist in two places at once. In the following example:

```
sudo ln -s /usr/share/doc/python-numpy/THANKS.txt
numpy-THANKS.txt
```

...a file will appear in your current working directory. But `ls -l` and you'll see that it's a special type of file, a link pointing to the actual file. Edit `numpy-THANKS.txt` and you'll find that `THANKS.txt` in the linked directory will be edited.

Soft, or 'symbolic' links, are created with the `-s` switch – you don't even need the file to which you're linking to actually exist, which makes it handy if you're linking across a network, or to a removable drive.

```
pi@raspberrypi: /etc/systemd
# Should-Stop:          $portmap
# X-Start-Before:       nis
# X-Stop-After:         nis
# Default-Start:       2 3 4 5
# Default-Stop:        0 1 6
# X-Interactive:        true
# Short-Description:    Example initscript
# Description:          This file should be used to construct scripts to be
                        placed in /etc/init.d.
### END INIT INFO

More information on the format is available from insserv(8). This
information is used to dynamically assign sequence numbers to the
boot scripts and to run the scripts in parallel during the boot.
See also /usr/share/doc/insserv/README.Debian.
pi@raspberrypi:/etc/systemd $ ls /etc/init.d
alsa-utils          dphys-swapfile      mountall.sh          rc                  ssh
avahi-daemon        fake-hwclock         mountdevsubfs.sh    rc.local            sudo
bluetooth           halt                 mountkernfs.sh       rcS                 triggerhappy
bootlogs            hostname.sh          mountnfs-bootclean.sh  README              udev
bootmisc.sh         hwclock.sh          mountnfs.sh          reboot              udev-finish
checkfs.sh          kbd                 networking           rncbind             umountfs
checkroot-bootclean.sh keyboard-setup       nfs-common           rsync               umountnfs.sh
checkroot.sh        killprocs           ntp                  rsyslog            umountroot
console-setup       kmod                plymouth             sendigs            urandom
cron                lightdm             plymouth-log        sendsigs            x11-common
dbus                mtd                 procps              single
dhcpcd              mountall-bootclean.sh raspfi-config        skeleton
```

It's called a symbolic link because it works by linking to the name of the target file, rather than to the file data itself. Create a hard link:

```
sudo ln /etc/bluetooth/main.conf mybluetooth.conf
```

...and you have two names (and locations) for the same file – sounds like the same thing? Not exactly: if you delete the original file in the first example, you can replace it with a new file of the same name, but different contents, and the symbolic link will point to the new file. Remove the original file in the hard link case, and the link still points to the data.

## Location, location

The startup scripts – whether `init.d` or `systemd` – are generally for daemons, processes you want running all of the time, like web servers and databases. There are plenty of programs which do housekeeping that need to be run periodically – hourly, daily, weekly. For this purpose, the cron software utility is ideal for scheduling the running of such programs and tasks. Cron searches its configuration directories and runs through the scripts it finds there – have a look at the different folders in `/etc` with names beginning with ‘cron’.

The easiest way to get to know where things like this are on your system is to search with `locate` – which is not installed by default on Raspbian. Enter `sudo apt-get install mlocate`, followed by `sudo updatedb`, then `locate cron` – which will find you every file or directory with cron as its name or as part of its name.

The `locate` tool maintains a database of every file on the system, which itself is updated daily by cron. If you've made a lot of changes, or want to find out where some software you've just installed has put its config file, get `locate` to update its database with `sudo updatedb`.

The built-in alternative is `find`, a powerful utility which enables you to search particular directories – or the whole file system – by name or name fragment, size of file, how long ago they were modified, or whether they're bigger than another file – enough to deserve a whole chapter of its own. Because it searches the file system, rather than a cached listing, it takes longer than a `locate`, but it is always up to date, and has search options not found in `locate` (see the ‘RegExp’ box). To replicate our `locate cron` command with the `find` tool:

### [ REGEXP ]

The `find` tool can search by regular expressions, as well as (part) name. They're a whole book topic in themselves, but well worth investigating once you've got command-line basics under your belt, as regexps can be used with many commands.

[ REFUSENIKS ]

Not everybody wanted to move to Systemd. Ubuntu tried an alternative called Upstart, before joining the masses in the Systemd move. Raspbian, like Ubuntu, is based upon Debian, whose systemd adoption resulted in some developers creating a new distro called Devuan to carry on releasing Debian with a Sys-V init.

```
find / -name '*cron*
```

If you were looking for cron or crontab, but not anacron, you could search for 'cron\*' instead. There will be more output than you want, so pipe it through a pager, or perhaps a grep. Back to using cron – the easiest way is via crontab, which maintains a table where each row specifies a command, and how often it is to run.

You edit the crontab file not directly, but with **crontab -e**, which calls up the default editor to do it. To take an example from Michael Stutz's *Linux Cookbook* (No Starch Press), add an entry in the form:

```
45 05 * * 1-5 calendar | mail -s 'Your calendar'
me@myemailaddress.com
```

... which grabs the output from the venerable UNIX calendar program and emails it to you every morning. The first five crontab fields cover minute, hour, day of month, month, day of the week, and can all be replaced with a single special value, like **@daily** or **@hourly**. While **man crontab** tells you a little about crontab, **man 5 crontab** is far more useful as it covers the layout of the file, with examples. Run **man man** for more on the numbered sections available with **man** commands.

Note that traditional UNIX command-line mail is not installed by default on the Pi, so if you wanted to follow the example, you would need to install a simple mailer – we recommend **ssmtp**, and the Raspberry Pi forums contain plenty of tips on command-line mail, as it can be used in all sorts of projects.

## A fresh startup

Using the value **@reboot**, we can easily run our own scripts on startup, without messing about with system startup scripts. There are times when a full systemd startup script will be more appropriate, but for quickly getting something tested, put the script into crontab.

There are two things that may catch you out. Firstly, you might be running scripts out of a directory that you have in your \$PATH, which defines where Bash looks for commands. As \$PATH is only set once you log in and your personalised **.bashrc** file is read, scripts running from

```

pi@raspberrypi: ~
File Edit Tabs Help
pi@raspberr... x pi@raspberr... x
pi@raspberrypi:~ $ ls /etc/cron*
/etc/crontab

/etc/cron.d:

/etc/cron.daily:
apt aptitude bsdmainutils dpkg logrotate man-db ntp passwd

/etc/cron.hourly:
fake-hwclock

/etc/cron.monthly:

/etc/cron.weekly:
man-db
pi@raspberrypi:~ $ ls /etc/cron.daily
apt aptitude bsdmainutils dpkg logrotate man-db mlocate ntp passwd
pi@raspberrypi:~ $

```

crontab which are run immediately upon startup will not be aware of your \$PATH setting. So, you will need to express all commands by their full paths, such as `/home/pi/bin/test.sh` – as well as making sure that the permissions are sufficient.

Secondly, systemd’s parallel service starts also mean that some services, such as the network, as well as environment variables, may not be ready when your `@reboot` commands are called. If you have problems, try giving a short pause first. It’s ten seconds in this example crontab entry, but you could use the smallest time that consistently works on testing:

```
@reboot sleep 10; /usr/bin/python3 /home/pi/Documents/Python_Projects/hello_gpio.py
```

One of the scripts that you’ll see called by cron, is to run anacron, designed to periodically run tasks on machines that were not always switched on – so it is very useful on laptops, too – with tasks specified, in anacrontab, to run after so many days have passed since they last ran. You can also use the automation of crontab or anacron to run your own backup scripts, so we’ll discuss backup options in the next chapter.

Installing mlocate adds a script to `/etc/cron.daily` to update its database of files and folders every day

# [ CHAPTER TWELVE ] SAVE IT NOW!

Learn how to protect your data with backups and disk wipes

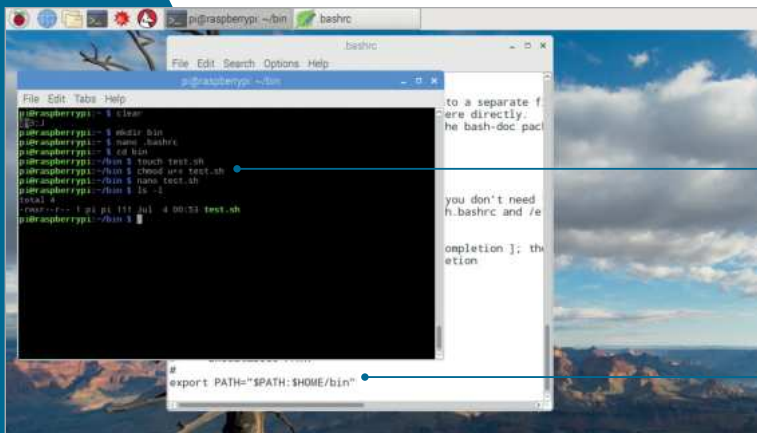


Perhaps you've got backups running automatically on your main computer, or perhaps you just back up what's important now and then – it's OK, we're not here to judge. But we will say that anything you don't have backed up, you don't have. Computers break. Hard disks and SSDs break. Accidents happen. The unexpected is somewhat inevitable.

IT professionals prepare as if they could lose everything at any moment, at least the best do. That might sound a tall order for a small single-board computer you bought at a disposable price to use in a hobby project, but your data is the most important thing on it, and good backups are a useful discipline to take elsewhere. Fortunately, the command line can actually make this easier; we'll show you some of your best options for (relatively) painless backups.

Simplest of all is to copy the data and move it elsewhere. It's labour-intensive, compared to automatic solutions, but for only-very-occasional backups it's certainly better than nothing, so we will not ignore this option.

Whether you're copying to disk, or moving the backup to another machine, it's best to make it as small as practically possible. The zip compression format may not compress as much as some specialist UNIX choices, but it does mean that other users should be able to open the file(s) with their standard compression software. You could go with LZMA or bzip2 compression (from tar, **-J** or **-j**, respectively) – both more widely installed than they used to be – for better compression to



By putting commands in a `.sh` file and making it executable, you create your own scripts

Making a `~/bin` folder and adding it to the `PATH` directive in `~/bashrc` means you'll be able to run your scripts by name

A single command wraps up all of the files and subfolders, then compresses the tar archive with gzip

```

pi@raspberrypi: ~
File Edit Tabs Help
pi@raspberr... x pi@raspberr... x
Documents/BlueJ Projects/LED-Button/README.TXT
Documents/BlueJ Projects/LED-Button/LED.java
Documents/BlueJ Projects/LED-Button/LEDTester.ctxt
Documents/BlueJ Projects/LED-Button/LEDTester.java
Documents/BlueJ Projects/LED-Button/package.bluej
Documents/BlueJ Projects/LED-Button/StringToMorse.ctxt
Documents/BlueJ Projects/LED-Button/StringToMorse.java
Documents/BlueJ Projects/LED-Button/LED.ctxt
Documents/BlueJ Projects/LED-Button/Button.java
Documents/BlueJ Projects/LED-Button/Button.ctxt
Documents/BlueJ Projects/people2/
Documents/BlueJ Projects/people2/README.TXT
Documents/BlueJ Projects/people2/package.bluej
Documents/BlueJ Projects/people2/Address.java
Documents/BlueJ Projects/people2/Person.java
Documents/BlueJ Projects/people2/Student.java
Documents/BlueJ Projects/people2/Database.java
Documents/BlueJ Projects/people2/Staff.java
Documents/BlueJ Projects/file-reader/
Documents/BlueJ Projects/file-reader/README.TXT
Documents/BlueJ Projects/file-reader/package.bluej
Documents/BlueJ Projects/file-reader/FileReader.java
Documents/BlueJ Projects/file-reader/test.txt
pi@raspberrypi:~ $ ls -l myprojectbackup.tgz
-rw-r--r-- 1 pi pi 2557667 Jul  4 01:10 myprojectbackup.tgz
pi@raspberrypi:~ $ tar czvf myprojectbackup.tgz Documents/

```

a smaller file size, but although alternatives to gzip save a little more space, they can take far longer to perform the compression.

Taking a directory of files that needs collecting together, then compressing it, can be done with a single tar command:

```
tar czvf mybackup.tgz myfolder
```

The **c** switch tells tar to create the archive, diving into all subfolders found; **f**, use the named file (here, **mybackup.tgz**), is necessary to direct the output away from the terminal, or – historically – a tape device. Yes, tape, for tar is short for ‘tape archive’, as telling a sign of its longevity as its frequent use without a dash in front of the command-line options. **v**, as with many commands, asks for more verbose output, so the program tells you what it is doing and what (if anything) has gone wrong.

Lastly, **z** invokes gzip compression – saving the extra step of creating a .tar archive, then running it through gzip. To unpack the archive, substitute **x** for extract in place of **c** – you can omit the **z**, as tar will recognise the compression type and automatically deal with it:

```
tar xvf mybackup.tgz
```

## A safe home

Often, using `tar` to back up `/home/pi` – with `cd /home`, then `tar` on the `pi` folder – will be all you need, but if you have data across directories, from `/etc` to `/var/www`, it's simplest to back up the entire microSD card. We looked at copying a new Raspbian image onto a card in chapter 10; backing up your disk is almost a mirror image of that process, which you can do on the Pi, with a USB card reader – with one important caveat.

You'll be creating a file as big as the entire SD card – usually 8GB or more – onto a Pi with a lot less space to spare. The solution is to compress the image file as it is created, which, for a Pi with a modest amount of data on it, will result in an image of around 2.5GB. Look back at chapter 10 for how to be sure which device the USB card reader is – for a card plugged in as `/dev/sdb`, and unmounted, do:

```
dd bs=4M if=/dev/sdb | gzip > back-raspbian.img.gz
```

Open another Terminal tab and monitor your disappearing disk space with `df` – if you don't think that it will fit, stop the `dd` operation with the usual `CTRL+C`, then `rm` the image file that you have partially created, and go and perform the backup on a computer with more disk space – or with a backup drive mounted, which you copy the archive to directly. Turning the backup into a usable microSD for the Pi means piping the other way, from `gzip` to `dd`:

```
gzip -cd back-raspbian.img.gz | dd bs=4M of=/dev/sdb
```

For disk operations like `dd`, you'll need root permissions: you can prefix `dd` with `sudo`, but for saving the file outside of `/home/pi` you may also need `sudo` – which means typing it in front of `gzip` as well.

This is only mildly inconvenient on the Pi, where `sudo` does not demand your password – but on a multi-user computer, or any setup with greater security, you need a reliable way of becoming the root user for every operation: running `sudo -s` will give you a shell with root permissions, but remember to `exit` afterwards. Alternatively, a chain of commands can be run with full admin permissions like so:

```
sudo bash -c "gzip -cd back-raspbian.img.gz | dd bs=4M of=/dev/sdb"
```

### [ TAPE ARCHIVE ]

`tar` dates from the days when computers backed up to big tape reels, those essential props of 1960s and 1970s sci-fi films. The lack of file structure on tapes means that `tar` can save all of the file system info such as ownership and timestamps.

## [ WHICH FILES? ]

Apart from `~`, your project may have config files in `/etc` or even `/opt`, and under `/var` are logs, web config and files – all of which you may have modified for a project.

## Remote copy

It's good to be able to make backups as required, using removable drives, but to move towards systematic backups you will need to copy across the network. We mentioned SCP in chapter 9. To copy your backup file to another machine, one that allows SSH login (so is running a SSH server), pass your login name with the command:

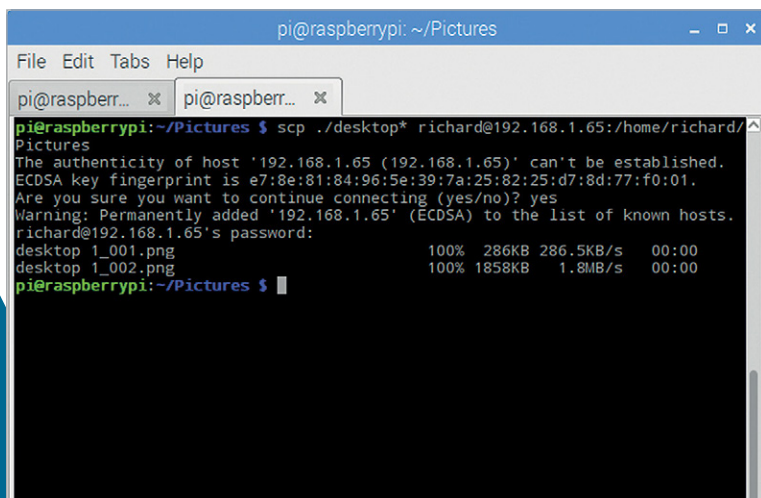
```
scp -p back-raspbian.img.gz pi@192.168.0.207:/home/pi/bak/
```

You will then be prompted for the user password. Change the **pi@** to whatever your user name is on the remote machine, not the Pi you're copying from. The **-p** preserves information such as when the files were last accessed. Note that **-P** (capital p) can be used to specify a particular port number.

We showed you how to set up a Pi with a fixed IP address in chapter 7; that Pi, with a plugged in USB disk drive, could be an inexpensive backup machine, as well as media server or whatever else your home or office needs.

Because you're sending these commands through the Bash shell, you get all the usual Bash advantages, from tab completion (just type **bac**, or however much of the file name is unique in your present working directory) to wildcards. If you have disparate archives

SCP makes command-line copying to remote machines as easy as moving files around on your Pi



in the same directory – such as **www-backup-20181225.gz** and **data-backup-20181226.gz** – then copy them all with:

```
scp -p /*backup*.gz pi@192.168.0.207:/home/pi/bak/
```

So far so good, but there are possibilities to automate your backup process later in the chapter, so the interactive element – having to give a password – would be better avoided. As long as you can maintain security in some other way, of course.

## Key to logins

Back in chapter 9 when we set up our SSH server, we generated keys with **ssh-keygen** – these keys can be used to provide passwordless login. You can copy them across to other machines manually, as we did earlier, but a handy shortcut is to use the command **ssh-copy-id**.

```
ssh-copy-id pi@192.168.0.207
```

If you have more than one key pair, use **-i** to specify which .pub file you're copying. **-p** allows you to specify an alternate port number – always a good thing in an internet-connected server, but not so necessary on a local network. Now we're all set for remote backups – but if you do them regularly, you'll waste a lot of disk space duplicating unchanging data.

**rsync** lets you copy data in much the same way as **scp**, but uses a delta-transfer algorithm, to only transfer the difference between the copies of the source file on your disk, and the remote, saved version. This both saves bandwidth used, and avoid cluttering up your backup disk with multiple near-identical versions of a file. It's also handy if you're paying a cloud provider for storage and data transfer.

If your version of Raspbian doesn't have rsync, it's just an **apt-get** away. Typically, rsync uses SSH for transport, but you can set up a server running an rsync daemon, and directly contact the **rsync://** URL over TCP (defaults to port 873). In this case, set an **RSYNC\_PASSWORD** environment variable or use the **--password-file** switch.

While rsync is not a built-in Bash command, we are highlighting it here as part of the array of command-line utility choices that users face when considering whether or not to simply employ

### [ WHY REMOTE? ]

As well as being able to centralise backups for more than one machine, a remote backup protects you from unexpected disasters such as fire or flood where the Pi is. Fairly unlikely? Yes, but that doesn't stop you insuring your house.

A timestamp in our script means that we are not producing a backup with the same name each day we run it

```

pi@raspberrypi: ~/bin
File Edit Tabs Help
pi@raspberr... x pi@raspberr... x
GNU nano 2.2.6 File: test.sh
#!/bin/sh
TODAY=$(date +%F)
cd /home/pi
tar czf mydocsbackup-"$TODAY".tgz Documents
scp mydocsbackup-"$TODAY".tgz pi@192.168.0.207:/home/pi/bak/
echo "done"
[ Read 9 lines ]
AG Get Help AO WriteOut AR Read File AY Prev Page AK Cut Text AC Cur Pos
AX Exit AJ Justify AW Where Is AV Next Page AU UnCut Text AT To Spell

```

built-in commands or to try something more complex instead. In addition, there is the possibility of using version control systems, such as git, for both backing up, and tracking changes on, important files.

## Script-it-yourself!

But let's row back to simpler commands. We have seen from early on how powerful Bash can be by chaining together a few commands; another way of putting commands together is to bundle them into a script – a short program simply comprising a small number of Bash commands, and known as a shell script. Take a look at this code – try typing it in to your favourite text editor, adjusting it for the IP address of your networked backup server, and backup folder location (or change the **scp** line to a **cp** to a plugged-in backup drive), and saving it as **test.sh**.

```

#!/bin/sh
cd /home/pi
tar czf mydocsbackup.tgz Documents
scp mydocsbackup.tgz pi@192.168.0.207:/home/pi/bak/

```

Then make the script executable with:

```
chmod u+x test.sh
```

...and run it with `./test.sh` – any problems, then check the names, network address, and did you perform the `ssh-copy-id` step?

Now we have a script that saves a folder, and copies it remotely, do you notice any potential problems? Each time you run it, it will overwrite the previous `mydocsbackup.tgz`, both locally and remotely. We need a way to put a timestamp on the backup name:

```
#!/bin/sh

TODAY=$(date +"%F")

cd /home/pi
tar czf mydocsbackup-"$TODAY".tgz Documents
scp mydocsbackup-"$TODAY".tgz pi@192.168.0.207:/home/pi/bak/
```

What we have done is set a variable – `TODAY` – to the current date, in YYYY-MM-DD format, which we can now access with `$TODAY` (remember, we permanently set variables for the shell named this way, when changing the prompt in chapter 5). You can run `date +%F` in the terminal – `date --help` will show you the many other format options. Now you can automate it by putting the script somewhere like `/usr/bin` (and with a better name than `test.sh`), and running it regularly with cron, as we covered in chapter 11.

Shell scripts tend to grow; there is always room for improvement. Here, you may want to back up more than one directory, for example, or use `echo` to let users know what the script is doing at each stage. You could even make it interactive, letting users choose which directories to back up.

There are plenty of shell scripting tutorials online, and great books (see *The MagPi* book reviews) to take it further, but Raspbian itself holds many great shell scripts, from which you can learn by example. To see how a script can be organised to still be maintainable with over a thousand lines of code, have a look at `/usr/bin/raspi-config`.

However grand or modest your scripting ambitions, don't be afraid to try things out: build up gradually, and test your code each time, so

## [ HASH BANG ]

The shebang, or hash bang – `#!` – at the start of the script is an instruction to the program loader to run the program immediately afterwards – in this case an interpreter directive to run `/bin/sh` – and pass the script as an argument to it.

## [ PASSWORD FREE ]

Using your key to log in is a convenience you quickly get used to – beyond `scp` to your backup server, try it on any machine under your control that you have to log in to.

that you know where to look for any errors that you introduce. Help is at hand from the Raspberry Pi forums, and pasting your code into **shellcheck.net** will give you valuable feedback – for example, advising you the `cd` line of our backup script should be:

```
cd /home/pi || exit
```

...in case the `cd` step fails – this is generally a good idea, although not so important in this particular case. Now that we have a choice of backup options, one task remains: securely getting rid of data from disks. This is a concern for anyone handling other people’s data, or just protecting their own privacy and security.

## Through the shredder

Back in the 1960s, if you wished to cover your tracks, *Mission Impossible* told us it was done by the show opening’s taped mission assignment finishing, “This message will self destruct in ten seconds...”, and boom went the tape player.

In these digital days, protecting privacy or security means understanding how a disk drive actually stores data, so that you don’t dispose of old disks under the mistaken impression that you’ve securely erased data, when you haven’t.

Disk space is collected into blocks, sized typically at 4096kB, that are indexed by the file system with inodes so that the disk controller knows where to send the read head to retrieve information. SSDs and flash drives don’t have read heads, but still organise the data in a similar fashion. Most disk operations occur at the inode level – so moving a file between directories on the same disk partition is simply done by relabelling an inode. `rm` does not delete the data stored, just the reference to its blocks on the inode.

Plug in a disk drive after someone has done `rm -rf` on it and it will look empty, but use a low-level utility and you’ll have access to all of the jigsaw puzzle pieces needed to put the data back together again. So far, so *NCIS*, but can this be significant to the average Pi user? Given the range of projects out there, and the multifaceted data that they collect, to stay on the right side of new and future data protection laws it will be useful to know how to securely remove data from your disks.



```

pi@raspberrypi: ~
File Edit Tabs Help
pi@raspberr... x pi@raspberr... x
pi@raspberrypi:~$ shred -v -n 5 top.secret
shred: top.secret: pass 1/5 (random)...
shred: top.secret: pass 2/5 (ffffff)...
shred: top.secret: pass 3/5 (random)...
shred: top.secret: pass 4/5 (000000)...
shred: top.secret: pass 5/5 (random)...
pi@raspberrypi:~$

```

Shred will securely overwrite (then optionally erase) files of sensitive data – or entire disk drives

## Caution

Before we start erasing disks, think of the carpentry maxim, ‘measure twice, cut once’. It’s easy to mistakenly erase the wrong disk or partition if you’re not paying attention. Given enough opportunities, most of us do it, and it’s often the lesson that teaches us to make proper backups! Running through other operations on the disk (**mount**, **df**, **ls**, **umount**), before erasing, works as a sanity check that you’re addressing the correct partition.

Now to those blocks. You can overwrite every bit of information, either with all 1 digits, or with totally random data, using **dd**. Even then, with magnetic disks, it’s theoretically possible to recover the data, and multiple overwrites will be necessary – but before you worry about writing a script to do that, let us introduce **shred**, a utility that does just that, overwriting with as many passes as you select as a command switch (or defaulting to three):

```
shred -vf -n 5 /dev/sdb
```

Adding a **-z** switch will overwrite the random data **shred** has used with zeroes, leaving a new-looking disk. **-u** will delete the file after the secure overwrite. **Shred** can also safely overwrite and/or remove individual files.

## [ WHERE AM I? ]

If you follow our tip on SSH keys everywhere, and end up hopping from machine to machine, remember to customise your Bash prompt so that you are always sure on which machine you’re about to erase a file.

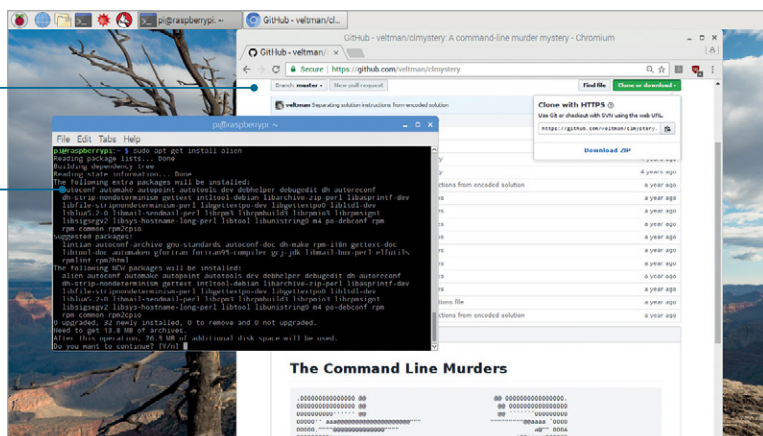
# [ CHAPTER THIRTEEN ]

# EASY COMPILATION

Here we look at how to install software  
and build it from the source code

Just paste the URI ending .git after git clone at the Terminal and you'll have the very latest source code

Install alien and you'll be able to swap packages between Raspbian, Fedora, and other systems' packages



It's so easy to install most software on Raspbian – provided it's mature enough to have been packaged as a .deb archive. Often, however, there's some great code available you'd love to try, but it asks you to compile it – sometimes after first cloning it from GitHub.

While not so straightforward as running an `apt-get install` command, there's little to fear in stepping through the decades-old ritual of compiling software – and when errors do occur, it's often quite easy to get back on track. We'll also look at scripted installs and Python packages, but first let's find out what to do if a package is in the wrong format.

Raspbian isn't the only distribution of GNU/Linux based upon Debian – **distrowatch.com** lists more than a hundred – and the ones you're most likely to have heard of include Linux Mint and Ubuntu. Ubuntu – the name derived from an Nguni Bantu term meaning 'humanity towards others' – is so popular that many projects, including ones not in the Debian and Raspbian repositories, maintain .deb packages for each version released.

Ubuntu also introduced the idea of Personal Package Archives (PPA) to the Debian family – special software repositories for uploading source packages to be built and published as an APT repository by Ubuntu's Launchpad software like the official Ubuntu repository, but for unofficial software from outside. You won't tend to find these with Pi software, but if you've been inspired to try one of Raspbian's relatives on your main PC, you'll find plenty of instructions on them in the Ubuntu community documentation.

Compilation can look complicated, but most projects provide a well-crafted make file for the process – a custom one even providing hints here

```

pi@raspberrypi: ~/Downloads/bashstyle-ng
File Edit Tabs Help
pi@raspberr...  pi@raspberr...
+ ffmpeg > Not Available
> movie2gif won't work

< Optional SuperUser Applications
+ dmidecode > OK

< Required Build Tools
+ msgfmt > OK

< Python Interpreter > 2.7.0 && < 3.9.0
+ Python > OK

< Required Python Modules
+ gettext > OK
+ Gtk (gi.repository) > OK
+ configobj > OK
+ shutils > OK

Notes from configure:
Prefix > /usr
Python > /usr/bin/python3
Post-Install Tasks > Enabled

You may want to continue with './make build'.
pi@raspberrypi:~/Downloads/bashstyle-ng $

```

When you run **apt-get**, or you **apt-cache search** to look for a package, APT quizzes its local record of what packages are available to it. The record of where it gets these packages from – the address of the repositories – is kept in the file **/etc/apt/sources.list** and at files in **/etc/apt/sources.list.d/** to which you can also add repositories by hand should a project you are interested in maintain one.

You can also edit **sources.list** (not advised) – editing all mentions of wheezy to jessie was an (unsupported) way of upgrading without overwriting your SD card when Raspbian updated. Meanwhile, plain old .deb files can be downloaded, and then installed with the command:

```
sudo dpkg -i example.deb
```

And any missing dependencies resolved with an:

```
apt-get install -f
```

There is another popular family of GNU/Linux distributions, based upon Red Hat, and including Fedora and CentOS. Fedora is available as an alternative to Raspbian (on the Pi 2 or 3), should you wish to try it out. What we're concerned with here is situations where you may need to install packages for one distribution, onto a system of the other type. Your friend

here is Raspbian's alien package, which will convert software between .debs and packages in .rpm (Red Hat Package Management) format.

```
alien some-package.rpm
```

...will convert from .rpm to .deb. You're more likely – on the Pi at least, where RPM-only packages are rarer – to need to convert the other way:

```
alien -r mysoftware.deb
```

## Friendly triad

Before package managers, it was normal to compile your own software, obtaining source code – usually written in the C or C++ languages – and running it through the GCC compiler, before linking libraries, and installing to the correct place on your disk drive.

Many of the headaches involved in the process are long gone as **configure** and **make** scripts do all of the hard work, checking dependencies are installed, then running the correct compiler flags for the project and the platform, and even installing the man page in the correct place.

Although most of the software you'll want to run will be available as a .deb to install with **apt** or **dpkg**, or come with a shell script which deals with installation (see below), plenty of projects, particularly those you'll find on GitHub or FreshCode, need unpacking then compiling.

After unpacking the archive with:

```
tar xvf latest-software.tgz
```

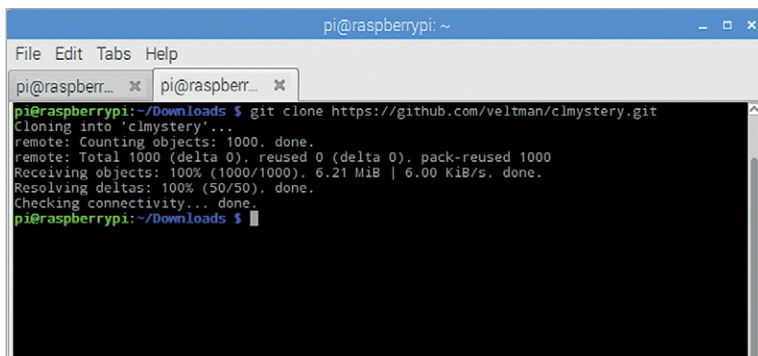
...**cd** into the directory created, and look for a file called **README** or **README.md**, or perhaps one called **INSTALL**. It will usually tell you to run a trio of commands that will become familiar – but read the instructions as there are variations, and some software even bypasses **make** with its own local version which you'll need to run as **./make**. The norm is:

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

## [ TRACK YOUR INSTALLS ]

Installing software from outside of Raspbian's repository means it must be looked after separately: updates, bug-fixes, security patches, and disaster recovery if something happens to your Pi. Keep track of what you install.

Git was developed by Linux creator Linus Torvalds to handle the complexities of multi-million lines of kernel code – yet makes it simple to maintain version control on the simplest of software.



```
pi@raspberrypi: ~
File Edit Tabs Help
pi@raspberr... x pi@raspberr... x
pi@raspberrypi:~/Downloads $ git clone https://github.com/veltman/clmystery.git
Cloning into 'clmystery'...
remote: Counting objects: 1000, done.
remote: Total 1000 (delta 0), reused 0 (delta 0), pack-reused 1000
Receiving objects: 100% (1000/1000), 6.21 MiB | 6.00 KiB/s, done.
Resolving deltas: 100% (50/50), done.
Checking connectivity... done.
pi@raspberrypi:~/Downloads $
```

However, on the Pi, `./configure` on its own may well result in programs (and particularly libraries) being built so they install in a not-quite-standard location – `/usr/local/lib` instead of `/usr/lib`. This can cause problems if these directories aren't on the library search path. In particular, if building a new version of a library that is already on the system, you will often end up with two versions of the library on the system – and the system will continue to use the old one. So, instead of plain `./configure`, we advise using the following to avoid issues:

```
./configure --prefix=/usr --libdir=/usr/lib/arm-linux-gnueabi
```

## Dependencies

Commercial software (open-source or proprietary) often comes as a large binary file containing all of the dependencies, statically linked into the application. With most non-commercial software projects, however, it is more common to just get a list of which version of which library will be needed to compile and run the software. Luckily, for a standard Debian package, there's a simple command that will install all the dependencies required to build it:

```
sudo apt-get build-dep <package name>
```

However, occasionally you'll need to compile and install something else first. Often the reason for compiling is to get the very latest

### [ GIT BOOK REVIEWS ]

We've reviewed a couple of useful books on Git in *The MagPi* – have a look at the reviews in issues 41 and 58.

version of the software from the developers, for new features, compatibility, or bug fixes. In the last few years, GitHub has become the default place to host free software projects; other repositories are available, but we'll just look at fetching software from GitHub, to quickly show you what you need to know.

Your first encounter with GitHub's existence may be seeing a banner on a project page inviting you to 'fork me on GitHub', or an invitation to 'clone' the software. Yes, we're in the world of a project big enough to sustain its own jargon. A fork is simply your own development copy to work on, after which you can offer the changes back to the project, or publish them (on GitHub or elsewhere) for others to try or build upon.

You don't even need to be able to code – many people are now using GitHub for collaborative development of documentation, including scientific research, and even fiction. But to simply download something, we don't need to worry about other Git methods – just 'clone' the application's source to your Pi with the following command:

```
git clone https://github.com/veltman/clmystery.git
```

...which will make a local directory containing all of the source files. **cd** into the subdirectory just created. Then follow the `configure/make` instructions as above.

## Nil desperandum

Sometimes, somewhere along the compilation, something goes wrong, and the script terminates with a complaint of some missing package. On a good day, you'll **apt-cache search** for the name of the missing dependency and there it will be, easy to install on Raspbian.

On a slightly-less-good day, you're going to have to dig around a bit to find the software, or the latest version that's being called for. Maybe having to go to GitHub or **SourceForge.net**. This is OK if the developer has a setup not too far removed from your own; the extra install steps may go smoothly. Otherwise, don't despair: help is available.

Most projects have one or more ways of reporting a problem and seeking help: a mailing list or Google Group; a wiki on the project SourceForge page; an email address for the lead developer; or even a Twitter account. Try to state your problem with plenty of detail, and

### [ IT'S A MYSTERY ]

The `clmystery` in the GitHub example is *The Command Line Mystery* – a text game which teaches you command line use. Entertaining and very useful.

[ GOING FOR  
UBUNTU ]

Ubuntu, can be installed on the Pi (2 or 3), as well as your PC, Mac or laptop.

remain polite and patient, and usually you'll find helpful people. Remember, although people want to be helpful, you are asking them to give up time to answer you – if you're frustrated with the installation process, and we've all been there, don't let that stop you being respectful of anyone going out of their way to help you.

If you don't have time for chasing up answers, but remain interested in a project, another option is to wait for the project's next release. By this time the dependencies may have become more widely available, even making their way into the Raspbian repositories, or the problem may simply no longer arise.

## A scripted install

You'll sometimes be asked to grab an installation script and run it directly – as is the case with the excellent EduBlocks, a half-way house between Scratch and Python which is helping young coders get over the large step between the two languages. The EduBlocks install wants you to run:

```
curl -sSL get.edublocks.org | bash
```

...which takes the shell script at [get.edublocks.org](http://get.edublocks.org) (you can look at it there in your browser) and pipes it through to a Bash process to run. The **-s** switch tells curl not to show a progress meter or error messages; **-S** overrides this to show an error message if it fails; while **-L** tells it to follow to wherever the site redirects to for the script.

If you don't like running software without knowing what it is doing to your Pi, or simply wish to take a look inside the script and see what the installation does, instead, download and save as **edu-install.sh** or just **install.sh** if that won't overwrite anything of the same name in your current working directory:

```
curl -o install.sh -L get.edublocks.org
```

...and you'll see the script downloads and unpacks a tarball with more than one script inside, first running the one that installs dependencies.

You can run the downloaded script – which you can do with **sh edu-install.sh**. If you don't read the script before running it, you're



```

pi@raspberrypi: ~/Downloads
File Edit Tabs Help
pi@raspberr... x pi@raspberr... x
GNU nano 2.2.6 File: install.sh
| echo "Removing old download..."
| rm -f edublocks-$ARCH.tar.xz
| fi
| if [ -d edublocks ]; then
| echo ""
| echo "Removing old extract..."
| rm -rf edublocks
| fi
| echo ""
| echo "Downloading package..."
| wget http://edublocks.org/downloads/edublocks-$ARCH.tar.xz
| echo ""
| echo "Extracting package..."
| tar -xvf edublocks-$ARCH.tar.xz
| echo ""
|
|_ Get Help WriteOut Read File Prev Page Cut Text Cur Pos
|_ Exit Justify Where Is Next Page UnCut Text To Spell

```

EduBlocks is a good example of a project that makes your life easy by putting every stage of the installation into a shell script, and giving you just one command to run the installation

placing a lot of trust not just in the developer(s) who wrote it, but every step of its journey before it reached your machine.

## Python, please

Over the years of installing software from various sources, you learn to recognise signs of whether or not it's likely to be a painless installation – and one cause for hope is something written in Python. Although both good and bad software can be written in any language, Python software and Python libraries just seem to be well-packaged and reliable.

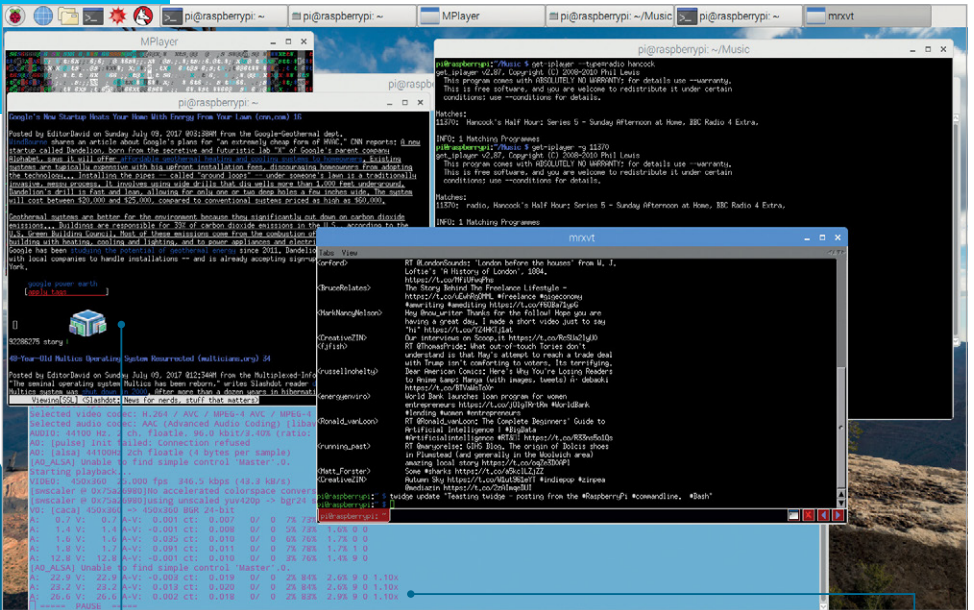
Although Debian-based GNU/Linux distributions like Raspbian come with the excellent APT and dpkg (Debian Package Manager), many popular programming languages have evolved their own ecosystems of packing tools and repositories – there are several for the Emacs text editor alone! Two that you are most likely to come across are JavaScript's npm and Python's pip. After using apt, the command format will be familiar, and:

```
sudo pip3 install numpy
```

...won't be a stretch to remember if the installation instructions tell you to install Python libraries such as NumPy. Happy installing, and above all, don't worry – it's quite hard to mess up the Raspbian installation, but if you do, at least those backups you made earlier will get tested out.

# [ CHAPTER FOURTEEN ] COMMANDING THE INTERNET

Yes, it is possible to get online from the command line. Here's how...



Text-mode browsing keeps the pics (but not the animated ads, thanks to w3m-img)

Keep up with Twitter in the terminal, or send tweets from shell scripts

**N**ow that you're getting comfortable at the command line, and perhaps finding it faster for some tasks, you may want to increase the amount of time you spend there, trying to improve your productivity on everyday tasks. Here we survey internet software – a diverse field, covering clients for protocols (IRC, Jabber), specific services (Twitter, BBC iPlayer), and tasks (search), as well as general web browsers, mail clients, and even surprising uses of the venerable Telnet client.

Don't get carried away; command-line alternatives vary from essentials you'll see often used in scripts and Pi projects, to less satisfactory alternatives you'd only use when there is no GUI available – so don't plan to switch all of your Facebook and Instagram use to the terminal. We concentrate here on internet interactions which will be useful for your Pi project, and sometimes you may be using the command line on the Pi and just need to look something up on the web, so welcome back to the Internet of Text.

If you've ever had to fall back on the Pi as a desktop machine, you'll know that capable as it is – particularly the four-core Pi 3 – you don't want to have too many browser tabs open at once. Some websites are so dependent upon JavaScript, though, that for all the memory that they hog on the Pi, you can only access them via Chrome (or Firefox). For others, try the retro world of the command-line web browser.

In the early 1990s, most people who had internet access browsed with a command-line browser called Lynx. After a quarter of a century, it's still in use! If not being able to see any pics makes the web pointless for you, consider the advantages. No tracking, no distractions from reading the text, and quicker page loads. Screen reader users and the mobility impaired can benefit from the simpler keyboard navigation, and as pages can be dumped to stdout, you can pipe it through other command-line applications:

```
lynx -dump https://duckduckgo.com/?q=gpio+bash | tail -n 30
```

Lynx isn't the only choice in command-line browsing – Raspbian also offers links2 and w3m; the latter is even usable from within the Emacs text editor. Install xterm (an alternative to Raspbian's default Terminal application) and w3m-img, and you can even view images when browsing from the terminal.

## Commanding the web

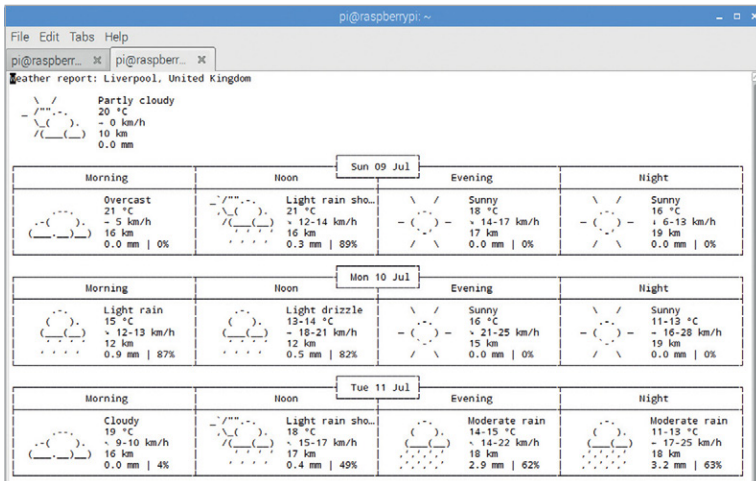
Telnet – short for teletype network – is the protocol developed at the end of the 1960s to provide two-way text communication between computers. It dates from an era without security concerns, and was long ago replaced by SSH as a means of connecting directly to other machines over open networks, but the Telnet client remains useful for interactively querying services such as mail or web for testing, by connecting to the appropriate port on the server, then issuing commands in the service's protocol:

```
telnet example.com 80
GET / HTTP/1.1
HOST: example.com
```

...which (after pressing **ENTER** again) should dump a webpage onto

### [ MOBILE ]

Responsive web design should mean that w3m works with more modern sites. While we wait for reality to catch up with the ideal, use mobile websites: w3m https://m.facebook.com



Command-line browsing is all about the information – mostly text, but not entirely

the terminal. It's a basic but useful diagnostic tool, but also gets used in some fun applications that we'll take a look at later in this chapter.

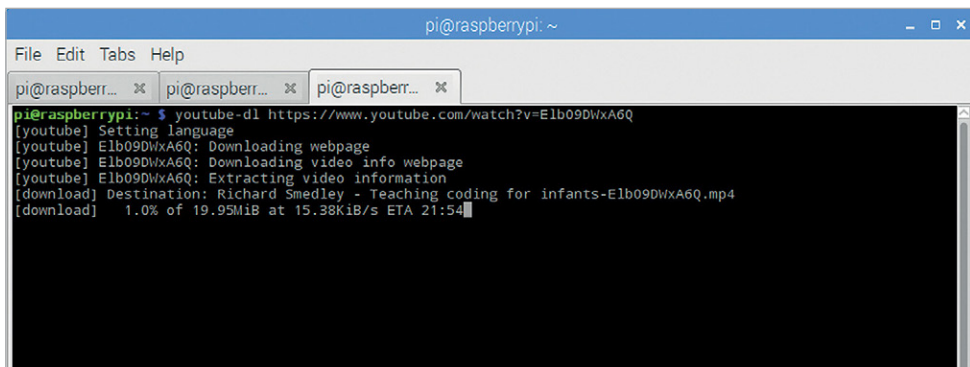
## Surfraw

As we have seen earlier, with `wget` and `curl`, functions such as downloading can be removed from the web browser. Another case where specialist commands seem most appropriate is searching. Looking things up from the command line can be done directly with a nifty little program called `surfraw`, which uses helpers called `elvi` to tackle different search tasks – see them all with:

```
surfraw -elvi
```

As a good citizen of the command-line world, `surfraw` pays due regard to your time and your tendons, and can be called simply with `sr`.

```
sr google -results=2 raspberry pi clojure gpio
sr translate peloton
sr gutenberd dickens
sr wikipedia permaculture
sr bbcnews tim peake
sr rhyme orange
```



No spare memory to open a YouTube link? Download the video to watch later

The results are displayed in your browser, which you can set in `/etc/xdg/surfwav/conf`:

```
def SURFWAV_text_browser /usr/bin/lynx  
defn SURFWAV_graphical no
```

## The Beeb

YouTube videos also won't work from a text mode browser, but it's easy enough to grab them with `youtube-dl`.

```
sudo apt-get install youtube-dl  
youtube-dl https://www.youtube.com/watch?v=Elb09DwxA6Q
```

Each downloaded video is saved as an MP4 file which, thanks to MPlayer's ability to output video through a choice of libraries, can even be rendered in ASCII through the terminal, using the `-vo caca` option. It can also be output via an SSH session, although forwarding the sound is an exercise we leave the reader to research. ASCII rendering is far from high-resolution, but if you run `mplayer` from one of the virtual consoles (which we visited back in chapter 1), it will default to a framebuffer output, for regular-quality video overlaying the command line.

You can also access BBC content from the command line. Although television programmes are only available from IP addresses located in the UK, radio shows – from Radio 3 concerts to classic comedy

and drama on Radio 4 Extra – can be downloaded for a few days after they have been broadcast, by the `get-iplayer` script; this saves them in the AAC format for listening to later. To keep within reasonable fair use terms – or at least be equivalent to the BBC’s own iPlayer service – the programmes should be deleted after 30 days.

At the time of writing, the programme search function no longer seems to work properly. So instead, you’ll need to use the iPlayer website to find the PID (Programme Identifier) of the desired show and then use it to ‘record’ (i.e. download) it. For example:

```
get_iplayer --pid=b0b95311
```

You can also use a comma-separated list of PIDs to download more than one show at a time.

```
get_iplayer --pid=b0b95311,b0b94zn8,b0b950c9
```

Calling with `--stream` sends the output to stdout, for redirecting via a pipe to an audio player (for radio shows). The `--stdout` switch does this in addition to recording.

## Communication

We mentioned command-line mail – scripted directly – in the calendar example in chapter 11. `ssmtp` makes a fine choice for the `mail` command, to use in scripts that email you when your Pi does something. Nowadays, getting the Pi to post to a microblog is a far more popular choice – at least if the microblog is Twitter, although other choices are available.

There are instructions all over the web for using Python modules to tweet when your Pi detects various events on the GPIO pins. If Python suits your project, then go ahead, but for the simplest setups all that’s needed is the easily scriptable `twidge`:

```
sudo apt-get install twidge
twidge setup
```

Setup is easy: `twidge` will give you a URL to visit, where you can authorise `twidge` to access your Twitter account; it passes you an

## [ EMAIL ]

If you struggle to keep up with email using a webmail interface, challenge yourself to use Mutt for two weeks: the command line may let you finally tame your inbox.

authorisation code to use, after which `twidge` is ready to go. `twidge lscommands` will list the available commands. To post from the command line (or a script), try something like:

```
twidge update "Testing twidge - tweeting from the  
#RaspberryPi #commandline."
```

The `update` command will also read from stdin. The helpful manual on the project's GitHub page will get you started. For an interactive (and very colourful) Twitter client on the command line, install Rainbowstream, which you can do with `pip`:

```
sudo pip3 install rainbowstream
```

## More than just gimmicks

Command-line apps are also available for older means of internet communication, from IRC to Jabber, and Mutt is a powerful enough email client to keep people (who otherwise don't spend much time in the terminal) using it. Serious work gets done in a shell, but coders keep churning out command-line apps that are just for fun, too.

MapSCII is a Braille and ASCII map renderer for your console, using OpenStreetMap data, written by Michael Straßburger ([magpi.cc/znMzWa](https://magpi.cc/znMzWa)). To connect from a remote computer:

```
telnet mapscii.me
```

ASCII Star Wars appeared at the end of the last century, and is still available at [asciimation.co.nz](https://asciimation.co.nz). Ironically, you'll need a modern graphical browser to view it.

Some sites are designed to look good in text-mode browsers. A favourite is [wtrtr.in](https://wtrtr.in), the weather website. Enter your location, and open in any of the text-mode browsers that we've mentioned; for example:

```
w3m wtrtr.in/Liverpool
```





THE Official  
**RASPBERRY PI**  
**PROJECTS BOOK**

VOLUME 4

**200 PAGES OF  
 IDEAS & INSPIRATION**

THE OFFICIAL RASPBERRY PI PROJECTS BOOK VOLUME 4



DIY Games Console



Set up a Spy Cam



**55 PROJECTS & GUIDES**



Build a Robot



Make a Magic Mirror

200 pages of Ideas & Inspiration

FROM THE MAKERS OF *MagPi* THE OFFICIAL RASPBERRY PI MAGAZINE

£12.99

200 pages of  
Raspberry Pi

THE *Official*

# RASPBERRY PI PROJECTS BOOK

VOLUME 4

Amazing hacking and making projects  
from the makers of *MagPi* magazine

## Inside:

- How to get involved with the Pi community
- The most inspirational community projects
  - Essential tutorials, guides, and ideas
  - Expert reviews and buying advice

Available  
now

[magpi.cc/store](http://magpi.cc/store)

plus all good newsagents and:

WHSmith

BARNES&NOBLE



Available on the  
App Store



GET IT ON  
Google Play

# THE OFFICIAL Raspberry Pi Beginner's Guide

The only guide you  
need to get started  
with Raspberry Pi

## Inside:

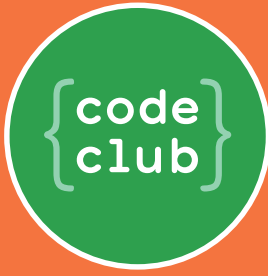
- Learn how to set up the Raspberry Pi, install an operating system, and start using it
- Follow step-by-step guides to code your own animations and games, using both the Scratch and Python languages
- Create amazing projects by connecting electronic components to the Pi's GPIO pins

**Plus much, much more!**

**£10 with FREE**  
worldwide delivery



Buy online: [magpi.cc/BGbook](http://magpi.cc/BGbook)



# Book of Scratch

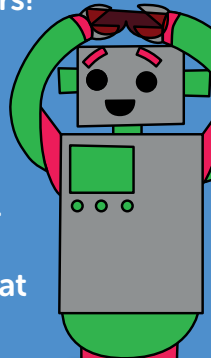
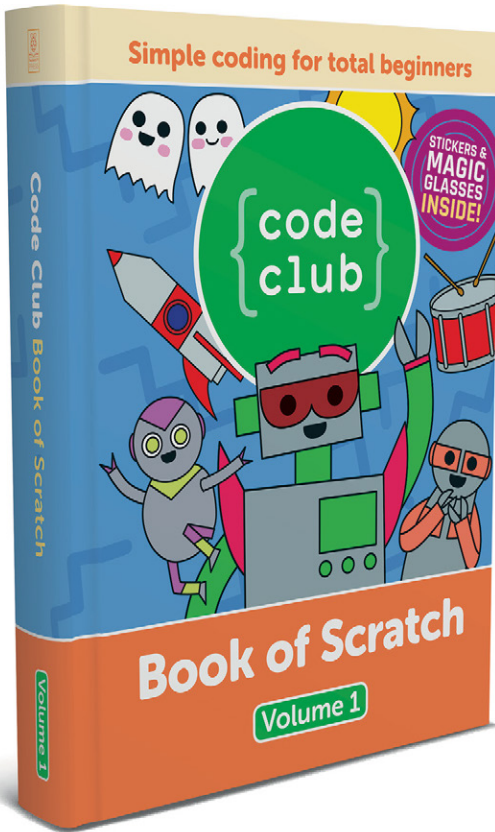
£9.99  
Free worldwide shipping!

Volume 1

Simple coding for total beginners

## The first Code Club book has arrived!

- Learn to code using Scratch, the block-based language
- Follow step-by-step guides to create games and animations
- Includes 24 exclusive Code Club stickers!
- Use the magic glasses to reveal secret hints
- The special spiral binding allows the book to lay flat

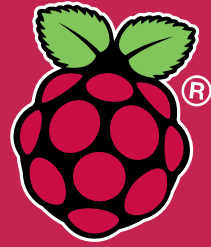


Available at: [magpi.cc/CCbook1](http://magpi.cc/CCbook1)



# The MagPi

raspberrypi.org/magpi Magazine



**SUBSCRIBE TODAY  
FROM ONLY £5**

**SAVE  
UP TO 35%**



#### Subscriber Benefits

- ▶ **FREE Delivery**  
Get it fast and for FREE
- ▶ **Exclusive Offers**  
Great gifts, offers, and discounts
- ▶ **Great Savings**  
Save up to 35% compared to stores

#### Rolling Monthly Subscription

- ▶ **Low Monthly Cost** (from £5)
- ▶ **Cancel at any time**
- ▶ **Free delivery to your door**
- ▶ **Available worldwide**

#### Subscribe for 12 Months

£55 (UK)      £90 (USA)  
£80 (EU)      £95 (Rest of World)

Free Pi Zero W Kit with 12 Month upfront subscription only  
(no Pi Zero Kit with Rolling Monthly Subscription)

 **Subscribe online: [magpi.cc/subscribe](https://magpi.cc/subscribe)**

JOIN FOR 12 MONTHS AND GET A

# FREE Pi Zero W Starter Kit

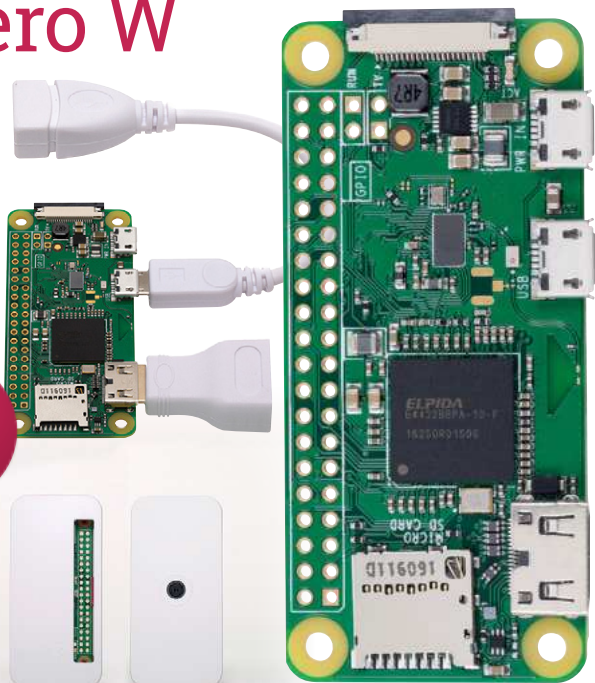
WITH YOUR SUBSCRIPTION

Subscribe in print for 12 months today and you'll receive:

- ▶ Pi Zero W
- ▶ Pi Zero W case with three covers
- ▶ USB and HDMI converter cables
- ▶ Camera Module connector

Offer subject to change or withdrawal at any time

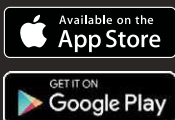
WORTH  
**£20**



 Buy now: [magpi.cc/subscribe](http://magpi.cc/subscribe)

**SUBSCRIBE**  
on app stores

From **£2.29**



# *The* *MagPi* ESSENTIALS

[raspberrypi.org/magpi](http://raspberrypi.org/magpi)